Online Post-Processing In Rankings For Fair Utility Maximization

Ananya Gupta*
UMass Amherst

Eric Johnson*
UMass Amherst

Justin Payan†
UMass Amherst

Aditya Kumar Roy
UMass Amherst

Ari Kobren
Oracle Labs

Swetasudha Panda
Oracle Labs

Jean-Baptiste Tristan
Boston College

Michael Wick
Oracle Labs

ABSTRACT

We consider the problem of utility maximization in online ranking applications while also satisfying a pre-defined fairness constraint. We consider batches of items which arrive over time, already ranked using an existing ranking model. We propose online post-processing for re-ranking these batches to enforce adherence to the pre-defined fairness constraint, while maximizing a specific notion of utility. To achieve this goal, we propose two deterministic re-ranking policies. In addition, we learn a re-ranking policy based on a novel variation of learning to search. Extensive experiments on real world and synthetic datasets demonstrate the effectiveness of our proposed policies both in terms of adherence to the fairness constraint and utility maximization. Furthermore, our analysis shows that the performance of the proposed policies depends on the original data distribution w.r.t the fairness constraint and the notion of utility.

ACM Reference Format:
Ananya Gupta, Eric Johnson, Justin Payan, Aditya Kumar Roy, Ari Kobren, Swetasudha Panda, Jean-Baptiste Tristan, and Michael Wick. 2021. Online Post-Processing In Rankings For Fair Utility Maximization . In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437963.3441724

1 INTRODUCTION

Ranking models are ubiquitous and support high stakes decisions in a variety of application contexts, e.g., online marketing, job search and candidate screening, loan applications, etc. Depending on the application, these models are used to rank products, job candidates, credit profiles, etc. Ultimately, these models facilitate selection of specific items from the ranked list.

Many practical instantiations of ranking applications are online processes where there is an incoming stream of batches of items to be ranked. For example, if we consider a hiring application, a job advertisement elicits applicants which naturally arrive over time. The hiring entity processes these applications in batches to screen and select candidates for job interviews. Unlike a static ranking application, such an online system necessitates proactive decision making so as to maximize long-term utility. In the context of hiring, this translates to the selection of qualified candidates given an unknown distribution over the batches of future applicants.

The position of an item in a ranking directly influences its visibility or *exposure* [23], thereby directly affecting whether or not the item is eventually selected. Standard techniques for ranking often involve ordering items in descending order of *relevance* (as defined in information retrieval literature [22]). For example, in hiring, relevance can be quantified as the degree to which an applicant's qualifications match the job requirements. These standard techniques are referred to as *utility* maximizing ranking algorithms. However, recent literature has shown that utility maximization can lead to representation disparities in the generated rankings [14, 23], either in the static or online environment.

Recent work in machine learning fairness attempts to alleviate discrimination by enforcing adherence to specific fairness criteria or equivalently, fairness constraints [8, 9]. While a large proportion of this research focuses on supervised classification [27, 28], the idea of fairness in ranking is relatively less explored [3, 6, 23, 29]. In previous work, algorithms have been proposed to satisfy a variety of such fairness constraints (e.g., parity of exposure). However, these algorithms are primarily aimed at static rather than online ranking problems. One technique for ensuring adherence to a fairness constraint in classification is to post-process the decisions from a trained (black-box) classifier [11]. In the offline variant of this technique, a learned classifier is modified offline to generate a derived classifier which is then deployed to make the predictions in real time. One issue with this approach is that adherence to the fairness constraints hold in expectation with respect to the training data distribution. Consequently, this might lead to sporadic violations of the fairness constraints on the test data distribution.

In this work we consider the problem of satisfying fairness constraints while maximizing utility in online ranking applications. We do not advocate for a specific fairness constraint. Instead, we assume that a constraint is assigned prior to deployment, and our goal is to ensure that it is always satisfied. While our approach can incorporate general constraints, in this paper, we define the *demographic disparity* criterion to ensure parity of pairwise *exposures* of the different groups of items over an aggregate of observed ranking batches. Parity constraints are important in ranking applications beyond fairness considerations, e.g., to incorporate diversity in rankings.

*These authors contributed equally to this research.
†Work completed prior to author's internship with Amazon.

WSDM '21, March 8–12, 2021, Virtual Event, Israel

ACM ISBN 978-1-4503-8297-7/21/03…$15.00
https://doi.org/10.1145/3437963.3441724

We consider an incoming stream of batches of items that need to be ranked for a specific application. An existing ranking model generates a ranking from each batch at a given timestep. The goal is to decide whether (and how) to re-rank the batch in order to maximize cumulative utility while enforcing the fairness criteria. We post-process or re-rank the decisions generated by the initial (fixed) ranking model by deploying a *re-ranking* algorithm that guarantees that the fairness constraints are satisfied. Unlike a static intervention, an online post-processor is better equipped to handle concept drift in the test data distribution. It can address instantaneous fairness constraint violations at any given timestep, thereby satisfying the fairness constraint pro-actively through the time steps, while maximizing a predefined utility notion.

We begin by proposing two different deterministic policies: Fair Queues and Greedy Fair Swap. Since our framework involves an unknown distribution over ranking batches, we also propose to learn a re-ranking policy via a novel variation of learning to search [7], dubbed *locally optimal learning to search with queues* (L2SQ). L2SQ creates a priority queue for each group in a batch, within which items are ordered according to relevance scores from the initial ranking model. The learned policy creates a re-ranking by repeatedly deciding which queue to pull from until all queues are empty. The learned policy's action space is defined at each position in the re-ranking to be the non-empty queues which, upon being pulled from, can result in a ranking that satisfies the fairness constraint.

In summary, we make the following contributions: a) we present an algorithmic framework for online post-processing of ranking batches according to a given fairness criteria, b) we propose two deterministic policies, and a novel approach to learn a re-ranking policy which maximizes utility while respecting the fairness criteria, and c) we present extensive experimental results on various real world (German Credit, AirBnb, StackExchange and Resume) as well as a synthetic dataset to demonstrate the effectiveness of the proposed approaches in terms of utility and adherence to the fairness constraint. We release our processed versions of the datasets for the online re-ranking setting. In addition, our experiments demonstrate that the performance of the policies depends on the original data distribution w.r.t the given fairness constraint and the notion of utility. For concreteness and based on available datasets, we consider specific instances of ranking applications, but our methodology is generally applicable.

2 RELATED WORK

There are several directions of ongoing research at the intersection of algorithmic fairness and rankings. Yang and Stoyanovich [26] propose various logarithmic discounting based measures for fairness and extend the approach on learning fair representations in [31] by redefining the loss function to be appropriate for ranked outputs. Several novel metrics for fairness auditing on rankings are proposed in [15, 21]. There is previous work on mitigating bias in the relevance scores, which are often used for generating ranking models. This includes work on inverse propensity scoring to estimate true relevance scores produced using click data [13] and learning a fair relevance model [30]. Kulshrestha et al. investigate and distinguish between sources of bias arising from the data and the ranking model respectively [16]. Asudeh et al. and Guan et al. break the ranking score into a linear combination of component scores, with weights on each component given by the user. If the user-provided weights produce an unfair ranking, their algorithm proposes the closest weight vector which produces a fair ranking [1, 10]. There is previous research on learning ranking models which satisfy specific fairness constraints. Beutel et al. introduce a set of novel metrics for fairness auditing in recommendation systems and improve fairness criteria during model training using pairwise regularization [2]. Singh et al. propose a learning to rank approach for utility maximization with fairness constraints [24].

Unlike the above previous work, in our problem setting, an existing ranking model generates a ranking of items (or a sequence of rankings). Our approach performs a re-ranking of the items and constructs a new ranking so as to maximize a given utility notion while satisfying a fairness constraint. Consequently, we do not directly learn a ranking model or intervene during the training process of a ranking model.

One direction of previous research most relevant to our approach is utility maximization in rankings subject to fairness constraints. Singh et al. define the concept of exposure and optimize for fair probabilistic rankings [23]. While their algorithm satisfies the fairness constraints in expectation, the constraints might not hold on the individual rankings sampled from the optimized probability distribution. Zehlike et al. [29] present a statistical test based approach which operates on a series of rankings in an online fashion. However, they consider a specific fairness criterion based on the proportion of certain group members at each position in the ranking. Biega et al. [3] propose the idea of amortized fairness, but the analysis is specifically for individual fairness criteria. Celis et al. [5] propose a constrained maximum weight matching algorithm for utility maximization with a fairness constraint. However, the approach does not consider aggregate of rankings in an online setting. Panda et al. [18] explore audit and control of fairness measures in ranking batches. We present a concrete formulation of the online post-processing problem for ranking batches and analyze deterministic as well as learned policies as solution approaches.

There is significant previous research on diversity in information retrieval [4, 17] where the goal is to not present similar items in the rankings. Stoyanovich et al. study the online, diverse top-k set selection problem which is different from our problem setting [25]. Sakai and Song present a metric for auditing rankings for diversity measures [20]. In contrast, our approach reconstructs rankings to satisfy given diversity criteria. Moreover, unlike the majority of previous work on ranking diversity which are based on similarity of items, our approach operates on a discrete set of items as groups.

3 PROBLEM DEFINITION

In this section we describe the problem setting of online fair utility maximization in ranked batches. We begin with a brief review of standard definitions from information retrieval. Consider a *batch* of n items $i \in 1, 2, \ldots n$. Let $r(i)$ denote the rank of item i in ranking r. The exposure [23] of i under ranking r is defined as the following.

$$\text{Exposure}(i|r) = \frac{1}{\log(r(i)+1)} \tag{1}$$

Let $q(i)$ be the relevance of item i. The discounted cumulative gain (DCG) of a ranking r is defined as $DCG(r) = \sum_{i=1}^{n} \frac{2^{q(i)}-1}{\log(r(i)+1)}$ and the normalized DCG (nDCG) of r as $\frac{DCG(r)}{DCG(r^\star)}$, where $r^\star$ ranks items in decreasing relevance order.

We assume that each item i in the batch is a member of a pre-defined group, $g(i)$. In this work, as a concrete example of a fairness constraint, we aim to generate rankings so as to equalize exposure across groups [23, 30]. Specifically, let $G_j = \{i \in [n] \mid g(i) = j\}$ be a group of items. Then the exposure of G_j in ranking r is given by $\text{Exposure}(G_j|r) = \sum_{i \in G_j} \text{Exposure}(i|r)$.

We denote our fairness constraint as the *demographic disparity* (DDP) constraint that bounds the difference in mean exposures between all pairs of groups. We define the DDP of a ranking r as

$$DDP(r) = \max_{\{G_j, G_{j'}\}} \frac{\text{Exposure}(G_j|r)}{|G_j|} - \frac{\text{Exposure}(G_{j'}|r)}{|G_{j'}|}. \quad (2)$$

Our fairness constraint ensures that the DDP is less than a predetermined threshold α. Note that DDP is analogous to the demographic parity constraint in classification. It also relaxes a previously proposed demographic parity constraint on rankings [23]. Although we focus on DDP here, our approach can be adapted to account for any general fairness constraint.

3.1 Online Post-Processing For Rankings

Unlike previous work, we focus on the online setting where the batches of items arrive over time. Specifically, at each timestep $t \in \{1, \ldots, T\}$ we receive a batch of items initially ranked by a fixed ranking model in descending score order (ranking denoted $r^{(t)}_{\text{init}}$).

We will define a post-processing policy π to re-rank the items in each batch according to a new ranking $r^{(t)}$ containing group populations $G^{(t)}_j$, such that nDCG is maximized and the fairness constraint is satisfied. However, in this setting, the nDCG and the fairness constraint apply in aggregate over batches. In particular, at a given timestep t, we define the nDCG at t for some sequence of rankings $R = \{r^{(1)}, \ldots, r^{(t)}\}$, $nDCG(R, t)$, as

$$\frac{1}{t} \sum_{s=1}^{t} nDCG(r^{(s)}) \quad (3)$$

The DDP at t, $DDP(R, t)$, is

$$\max_{\{G_j, G_{j'}\}} \frac{\sum_{s=1}^{t} \text{Exposure}(G^{(s)}_j|r^{(s)})}{\sum_{s=1}^{t} |G^{(s)}_j|} - \frac{\sum_{s=1}^{t} \text{Exposure}(G^{(s)}_{j'}|r^{(s)})}{\sum_{s=1}^{t} |G^{(s)}_{j'}|} \quad (4)$$

π uses all rankings $\{r^{(s)}\}_{s=1}^{t-1} \cup \{r^{(t)}_{\text{init}}\}$ to compute the utility and the constraints in aggregate. However, it can only re-rank the current batch and not any of the previous batches. By re-ranking the current batch, our post-processing policy aims to satisfy the fairness constraint (in aggregate) while maximizing cumulative utility over the batches observed so far. Consequently, our goal in online post-processing is given by

$$\begin{aligned} &\text{maximize } nDCG(R, T) \\ &\text{subject to } \max_{1 \le t \le T} DDP(R, t) \le \alpha \end{aligned} \quad (5)$$

Henceforth, we use the term fair ranking to denote an aggregate of ranking batches up to a given (current) time step which satisfy our DDP fairness constraint (and the term unfair ranking otherwise).

4 DETERMINISTIC POLICIES

A deterministic policy re-ranks a batch at a given time-step so as to obtain a solution to the objective in Equation 5. In this section, we describe two deterministic re-ranking policies based on different heuristics: Fair Queues and Greedy Fair Swap.

4.1 Fair Queues

We begin by modifying the FA*IR algorithm proposed in [29]. The original FA*IR algorithm creates a priority queue for each group, sorted in decreasing order of relevance. This is followed by construction of a new ranking as follows. To fill each position in the new ranking, it identifies the queue with the most-qualified (top) item and pops from that queue. If that selection results in a sub-ranking which violates the fairness constraint, it identifies the queue with the next most-qualified top item and pops from that queue instead.

We denote our modification of FA*IR as Fair Queues. This algorithm works in a similar fashion to FA*IR, but there are two key differences. FA*IR models each sub-ranking on n' items using a binomial distribution $p(k; n', p)$ and checks that $p(k; n', p) > \alpha$, while Fair Queues applies our non-probabilistic DDP constraint on full rankings. Second, our fairness definition is based on group exposure in aggregate across multiple time steps and applies to multi-group settings, while the fairness definition in [29] only applies to single rankings with two groups (usually denoted as the protected and non-protected groups).

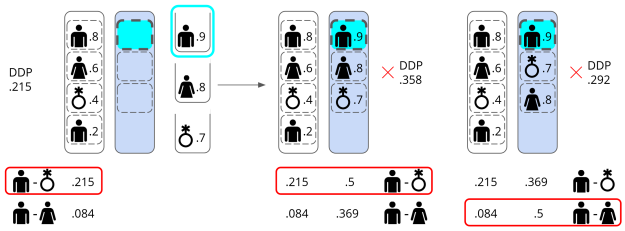

Figure 1: An example illustrating the *can be fair* pruning for the action space. There are 3 groups: male, female, and non-binary. The DDP threshold is 0.25. The model checks if it can select from the male queue and still create a fair ranking. Both possible ranking completions are unfair after selecting a male item, indicating that selection from the male queue is not a valid action for this time step. The red highlights indicate the group pairs which attain the maximum average exposure difference (DDP).

We denote the subroutine that checks if a ranking can be completed while satisfying the fairness constraint as *can be fair* (illustrated in Figure 1). A naive approach would examine every completion of the ranking until it finds a fair completion or until it exhaustively examines all enumerations of ranking completions. Since there are $n!$ rankings on n elements, we use a heuristic (Section 4.1.1) to find a single ranking completion. If that heuristic-based

ranking completion is unfair, *can be fair* fails, and we restrict the algorithm from selecting from the queue in consideration. In case all queues are eliminated, we select from the queue of the group with the minimum exposure.

Using the heuristic for *can be fair*, Fair Queues has a worst-case complexity of $\Theta(gn^2)$ for a ranking with n items and g groups. The use of a heuristic rather than an exact method for *can be fair* implies that we might sometimes over-restrict the action space. However, even with the heuristic, *can be fair* never incorrectly allows for selecting a queue which precludes a final fair ranking, as long as there is a queue which allows for a final fair ranking. Therefore, while the reconstructed ranking might be sub-optimal with respect to nDCG, it will be fair whenever possible.

4.1.1 Can Be Fair Heuristic. Our heuristic completes a ranking using the same basic framework as Fair Queues - it selects a queue to draw from at each step. However, rather than selecting from the queue with the most relevant top item, it selects from the queue with the least expected exposure if we fill each remaining position by selecting from a *random* queue . To calculate the expected exposure, we first calculate the average exposure for all remaining open slots in the ranking. We then compute the expected exposure for a group by assuming that each remaining item in the group's queue receives the average remaining exposure. We can then average the exposures for each group under this assumption, and select from the queue with the lowest expected exposure. We do not claim this heuristic creates the optimally fair ranking completion, but rather treat it as a reasonable approximation.

4.2 Greedy Fair Swap

Our second deterministic policy denoted as Greedy Fair Swap aims to promote members of groups with lower exposure within a single ranking $r^{(t)}$. The algorithm (Algorithm 1) iteratively selects the most highly ranked member (highest relevance score) of a lower-exposure group which is still below a member of a higher-exposure group, and swaps them. Note that this swap is greedy because it minimally lowers nDCG while guaranteeing a lower DDP. The algorithm terminates when the rankings up to time t meet the DDP threshold α. Since there are $\binom{n}{2}$ possible swaps, Greedy Fair Swap is $O(n^2)$ for re-ranking a batch with n items. This algorithm does not necessarily produce a ranking with optimal nDCG under the fairness constraint.

5 POLICY BASED ON LEARNING TO SEARCH

Both Greedy Fair Swap and Fair Queues have a potentially undesirable property by definition, which is they only act when it is absolutely required. Consequently, the DDP measure stays very close to the threshold at all times. If these policies suddenly have to re-rank a batch with highly relevant items from the group with the highest exposure, these will be forced to take a large penalty on nDCG to maintain the DDP under the fairness threshold. This weakness motivates a more pro-active learned policy. In this section, we describe a learned policy based on a variation of learning to search.

Algorithm 1: Greedy Fair Swap

input : Initial ranking r_{init} on items $\{i_1, \dots i_n\}$, group membership function g, threshold α'
output : Ranking r on $\{i_1, \dots i_n\}$, with $DDP(r) \leq \alpha'$

Initialize $r = r_{\text{init}}$
while $DDP(r) > \alpha'$ **do**
  Identify the group with highest exposure G_h
  Identify the group with lowest exposure G_l
  Set $l = \underset{i_j \in G_l \mid \exists i_{j'} \in G_h, r(i_{j'}) < r(i_j)}{\arg\min} r(i_j)$
  Set $h = \underset{i_{j'} \in G_h \mid r(i_{j'}) < r(l)}{\arg\max} r(i_{j'})$
  Swap l and h in r
return r

5.1 Background

Locally optimal learning to search (LOLS) [7] learns a policy by imitating and extending a *reference policy*. Since the learned policy provably has low regret on deviations from the reference, it is possible to improve upon the performance of the reference [7]. The learned policy can be trained so as to predict an action from features derived from the *state space* at a given timestep. Below we summarize the concept at a very high level. LOLS constructs a training example by "rolling in" up to a given number of time steps according to the learned policy. For every action in the action space, LOLS "rolls out" using the reference policy (or possibly a mixture of the reference and the learned policy). This roll out terminates at an end state, and a score can be assigned to that end state. Using these scores, the model learns to prioritize actions which led to high scoring end states at a given time step.

5.2 LOLS With Queues (L2SQ)

Our proposed approach, Locally Optimal Learning to Search with Queues, merges the learning to search algorithm reviewed in Section 5.1 with the queue-based ranking procedure described in Section 4.1. A detailed description of L2SQ can be found in Algorithm 2. Concretely, we create a scoring model (a feedforward neural network) that maps from a partial ranking and a collection of queues (one per group) to a score for each queue. We select from the queue with the top score from the model, rather than the queue with the most-relevant item. Intuitively, we would like the L2SQ model to learn to maintain a fairness buffer well below the DDP threshold, allowing the model to take advantage of incoming batches with highly relevant items from a high-exposure group.

To implement the LOLS framework, we must define a reference policy, a parametrization of the state and action spaces, and a cost function to be applied at the end of roll-outs. At training time, we construct training examples where each example consists of a rolled-in set of rankings up to some timestep (described below) and a choice of queues from which to select the next element of the current ranking. We then roll-out for each possible choice of queue to obtain costs for each queue. From this pairing of state and costs, we construct multiple training examples to update the scoring model. To construct a set of rankings at inference time, we apply the scoring model for each slot of each ranking, filling in slots with the top item from the highest-scoring queue at each step.

Algorithm 2: L2SQ Training

input : Sets of initial rankings $\{R_{\text{init}}^{(n)}\}_{n=1}^{N}$, mixture parameter $\beta \geq 0$, and roll-out horizon h

for $n \in \{1, 2, \ldots, N\}$ **do**
  $R_{\text{init}} \leftarrow R_{\text{init}}^{(n)}$
  for $t \in \{1, 2, \ldots, T\}$ **do**
    Roll-in $t-1$ rounds to reach $r_{\text{init}}^{(t)} \in R_{\text{init}}$
    Create priority queue Q_g (ordered by decreasing relevance) for all groups g
    Initialize $r^{(t)} = \emptyset$
    while $|Q_g| > 0$ for at least 2 groups g **do**
      for $g \in \{1, 2, \ldots, G\}$ **do**
        if $|Q_g| > 0$ and can_be_fair($r^{(t)}, Q_g$) **then**
          Copy $r'^{(t)} \leftarrow r^{(t)}$
          Insert Q_g.pop() into $r'^{(t)}$
          Apply roll-out policy to fill $r'^{(t)}$ and the next h batches $r'^{(t+1)}, \ldots r'^{(t+h)}$
          Compute cumulative nDCG after roll-out for group g
      For all rolled-out g, compute cost = max nDCG for any group minus nDCG of g
      Construct training example from groups g
      Compute BPR loss
      Apply roll-out policy to insert Q_g.pop() into $r^{(t)}$
  Update model with total BPR loss

We parametrize the search space over queues (rather than over items) because DDP is based on groups and is agnostic to the choice of individual elements within a group.

Reference Policy Any ranking policy can be used as the reference policy. All our results use Fair Queues as the reference policy, since L2SQ did better with Fair Queues as a reference than with Greedy Fair Swap in early experiments.

Parametrization of State and Action Spaces We encode the state space using 17 features per group: mean exposure and percentage of the group in previous batches, total number of items in current ranking, statistics of relevance scores and ranks for items which have already been ranked (min, max, mean, standard deviation), the relevance score of the top item in the queue, size of the queue, and statistics of relevance scores for the queue (min, max, mean, standard deviation). We parametrize the model using a feedforward neural network, which takes as input all features for all groups and outputs a vector of scores, one per group.

The action space consists of all selections from non-empty queues which can result in a ranking that satisfies our constraint. We use the *can be fair* subroutine described in Section 4.1 to restrict the action space for L2SQ.

Roll-out and Cost Computation To create training examples, we roll-in up to a certain time step, simulate selecting from each non-restricted queue, then roll-out from each simulated choice to compute a loss function. The policy used for roll-out is a mixture of the learned policy and the reference policy, where the reference policy is selected with probability β. We calculate the score of each queue using the average nDCG over all batches after roll-out. An illustration of roll-out with two groups (male/female), four timesteps, and a DDP threshold of 0.25 is shown in Figure 2.

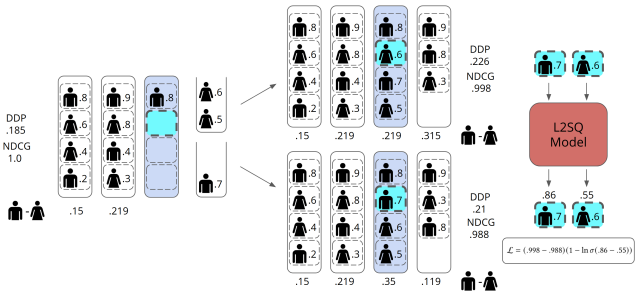

Figure 2: Roll-out and loss function computation at a single time step. We display the relevance of each item, as well as the difference in male and female exposures for each completed batch. We roll-out after selecting from each queue to calculate the post-roll-out nDCGs. The loss is a function of the post-roll-out nDCGs and the model's scores. Note the model's scores encode the model's preferences for selecting from each group's queue, not the relevances of particular items.

Training Examples and Loss Function We create multiple pairwise examples per state, comparing each queue to the queue with the highest post-roll-out nDCG. The model assigns each queue in the pair a score, and we compute the Bayesian Personalized Ranking loss [19] based on the pair of scores and final nDCGs for the two queues. If Q_1 is the queue with the highest final nDCG, then for every non-restricted Q_2 at a given timestep t, we calculate the loss as $l(Q_2, t) = (nDCG(Q_1) - nDCG(Q_2))(1 - \ln \sigma(f(Q_1) - f(Q_2)))$, where σ is the logistic function and $f(\cdot)$ is the score of the model for a queue. Note that we do not calculate any losses for actions which are restricted by *can be fair*. An example of loss function computation is in Figure 2 (right).

Inference At inference time, we apply the scoring model for each slot of each ranking, filling in slots with the top item from the highest-scoring queue at each step. We apply the *can be fair* restriction on the action space at inference time as well, to ensure that the generated rankings are fair (if possible). Because of the *can be fair* restriction on the action space, the L2SQ model has a worst-case complexity of $\Theta(gn^2)$ at inference time for a ranking with n items and g groups.

6 EXPERIMENTS

6.1 Data

We evaluate our proposed approach on the following four real world datasets: UCI German Credit, StackExchange, AirBnB, and a new dataset Resume. In addition, we analyse our algorithms on a synthetic dataset described below. In each case, we construct training examples with 10 batches each, and validation and test examples with 25 batches each.

Synthetic Data We construct a synthetic dataset to study the behavior of our proposed algorithms. We define four groups for generating this synthetic dataset. We generate the data by sampling an initial component $\sim U(0, 1)$ and adding a value sampled from a Gaussian random variable with standard deviation $\sigma = 0.1$ to each

group. We set the Gaussian random variable to have a negative mean for two of the four groups, where $\mu \sim U(-.75, -.25)$ is sampled for each batch, and zero mean for the remaining two groups. For each batch, we select a random number of elements from each group (specifically, between 3 and 7 items per group). We sample 100 training examples, 50 validation examples, and 50 test examples.

German Credit Hans Hofmann [12] introduced the German Credit dataset which maps applicants to credit rating/scores generated by the private German credit agency Shufa. Each applicant has features such as age, gender, martial status, etc. There are 1,000 applicants overall. Our data was created similar to [29] who assigned a score to each applicant as the sum of the relevant numeric or ordered attributes[1]. We rank the applicants by this score. For our experiments we take the cross-product of gender and age to create four groups. To convert these rankings we shuffle the data and then split into batches of size 20. We sample 100 training examples, 50 validation examples, and 50 test examples.

AirBnB Although their setting is different, we take inspiration from [3] to create a ranking dataset from AirBnB listings. Note that our results are not directly comparable to [3], because they consider individual fairness criteria while we consider DDP which is defined for groups. We create a dataset using AirBnB data[2] for the cities Boston, Seattle, and New York. There are 37,171 listings in our dataset. Each listing is considered an item, and we obtain item scores by summing ratings for each of 7 attributes: cleanliness, check-in, communication, location, review score, value, and accuracy. Ratings are generally very close to the maximum of 10. Batches (of size 20) are uniformly sampled at random from the total population. We create four groups of listings based on price per night: \$0-\$50, \$51-\$100, \$101-\$200, and \$201-\$1000. Although listing price is not a sensitive attribute, our fairness constraint will ensure that some listings from each price range appear in the results, providing cheaper listings more opportunity to be seen while giving the user diversity of choice. From this construction, we sample 300 training examples, 50 validation examples, and 50 test examples.

Stack Exchange Also inspired by [3], we create a realistic query-answer dataset from the StackExchange online archive[3]. As with AirBnB, our results on this dataset are not directly comparable to their results due to the difference in problem setting. Each ranking is composed of answers to a question asked on the Unix, AskUbuntu, and Academia StackExchange websites. We restrict ourselves to questions with at least 4 answers. There are 28,026 questions that meet this criterion. For each question, we rank the question's answers using cosine similarity between Latent Semantic Indexing vectors[4]. Answers are assigned a group based on the reputation of the posting user. We split users into 3 bins: low reputation (1-50), medium reputation (51-2,500), and high reputation (>2,500), so each item falls into one of these three groups. User reputation is not typically a sensitive attribute, but applying our fairness constraint improves diversity of results and gives less-experienced users more chance to gain reputation. As with other datasets, a single "example" consists of 10 questions for training or 25 questions for validation/test. We sample 1000 training examples, 50 validation examples, and 50 test examples.

Resume Motivated by the setting of screening candidates for a job in an online, batched setting, we construct a dataset of batched resumes from a freely available list of 14,800 parsed resumes for software engineers, data scientists, and other computer science related professionals across India[5]. We trained a simple model to predict, from the body of the resume, whether the applicant should be considered for a software developer position. As a simple approximation of such a signal, we train a logistic regression model to predict (using only the resume's body) whether or not the resume's title contains the word "developer." We use the scores output by the logistic regression model as the relevance scores. To sort resumes into groups, we use the resume's "state" field to map each resume to one of four regions of the country: North, South, East, and West. We use a batch size of 20 items each. We sample 200 training examples, 50 validation, and 50 test examples.

6.2 Experimental Setup

For each dataset we split the items into two populations, one for training and validation and the other for test. We sample sets of rankings in a way that ensures no item is repeated within a single ranking, but may be repeated across rankings. For training we use 10 timesteps for each set of rankings, and for validation and test we use 25 timesteps. We use Fair Queues as a reference policy for L2SQ, rolling out for three timesteps before calculating the loss for each deviation. The DDP threshold α for all datasets is set to 0.1. We optimize the model with Adam for 20 epochs. The model is a feed forward neural network with 17 features for each group (listed in Section 5.2), 2 hidden layers of size $\lfloor \frac{17}{2} \rfloor$ and $\lfloor \frac{17}{4} \rfloor$ features per group, and an output layer of size equal to the number of groups. The model for each dataset was trained on a Xeon Gold 6240 CPU @ 2.60GHz with 192GB RAM, though in practice we found the model did not use more than 10 GB of memory.

We tune the mixture parameter β on the validation set and keep the best model for test. We set the learning rate at 0.001 for all datasets. The test data consists of 50 sets of rankings for each dataset. We evaluate each algorithm on the test data and calculate, for each timestep, the mean and 95% confidence interval for nDCG and DDP across all 50 sets of rankings.

We compare the initial rankings to re-rankings from Greedy Fair Swap, Fair Queues, and L2SQ. Although there is a large amount of prior work on fair ranking, we are specifically focused on post-processing approaches that guarantee fairness over time in an online, multi-batch setting. We are not aware of prior work that directly fulfills these criteria, and thus do not make any further comparisons.

Our code is available online[6].

6.3 Results

We begin with the synthetic data. In this setting L2SQ attains the highest nDCG on the test data while still meeting the fairness threshold (top row of Figure 3).

[1] https://www.github.com/MilkaLichtblau/FA-IR_Ranking
[2] http://insideairbnb.com/
[3] https://archive.org/details/stackexchange
[4] https://radimrehurek.com/gensim/models/lsimodel.html
[5] https://www.kaggle.com/avanisiddhapura27/resume-dataset
[6] https://github.com/ejohnson0430/fair_online_ranking

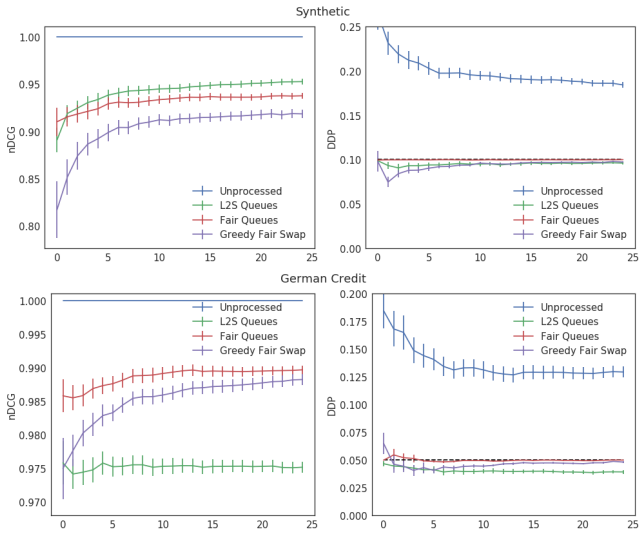

Figure 3: Results on synthetic data and German Credit data over 25 timesteps.

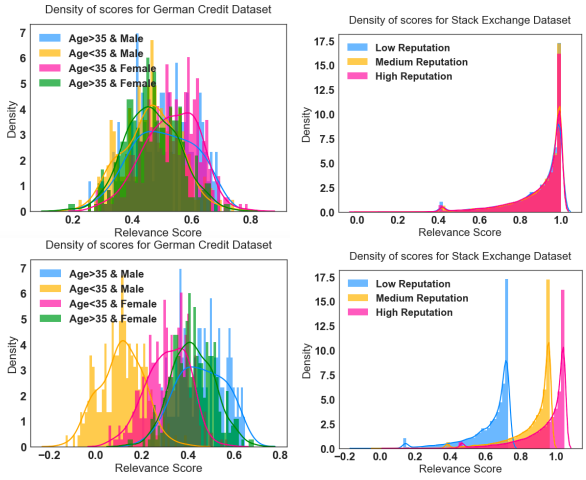

Figure 4: Distribution on GermanCredit dataset (left column) and StackExchange dataset (right column) with noise (top row) and without noise (bottom row).

On the real datasets, we find that our two deterministic approaches are effective at enforcing fairness while maintaining high nDCG. L2SQ enforces fairness, but its nDCG lags behind the nDCG of the deterministic algorithms. The results on the German Credit dataset are shown in the bottom row of Figure 3 (note that we lower the DDP threshold to $\alpha = 0.05$ since the unprocessed rankings already have low DDP). Results are similar on the other 3 real datasets and are omitted.

All our approaches produce fair rankings. However, our results suggest that the deterministic algorithms outperform L2SQ on the real datasets in terms of nDCG, but L2SQ excels on the synthetic dataset. By construction, the groups in the synthetic data have significant differences in mean relevance scores across groups, potentially requiring disruptive re-ranking to maintain fairness. The real datasets exhibit a low difference in mean relevance scores in the unprocessed rankings. We hypothesize that this discrepancy drives the behavior seen in the initial results.

We consequently run additional experiments with a wider spread of mean relevance scores for the real datasets, discussed in Sections 6.3.1 and 6.3.2. Section 6.4 provides further discussion on the algorithms' sensitivity to the difference in mean relevance scores across groups.

6.3.1 Score Distributions Across Groups. We find that for the real datasets, the distribution of scores is not significantly different across groups. We display the relevance score distributions for the German Credit and Stack Exchange datasets in the top row of Figure 4, though Airbnb and Resume show a similar pattern.

In the previous section, we hypothesized that L2SQ outperforms our deterministic approaches when the initial rankings are very unfair. To test this on real data, we sample new relevance scores for all 4 real datasets by adding positive values to some groups and negative values to others. We then evaluate the L2SQ model and our baselines on these datasets.

While constructing the rankings for each dataset, we add values sampled from a Gaussian random variable with zero mean for groups with higher exposure and a negative mean for groups with lower exposure. These values are drawn from $\mathcal{N}(0, 0.1)$ and $\mathcal{N}(-0.25, 0.1)$ respectively for German Credit, Resume, and Stack Exchange, where relevance scores are between 0 and 1. The values are drawn from $\mathcal{N}(0, 0.25)$ and $\mathcal{N}(-1, 0.25)$ respectively for AirBnB, where the relevance scores are between 0 and 7.

As an example, we show the new distribution of scores on German Credit and StackExchange in the bottom row of Figure 4. The groups form separated modes in the space of scores, indicating that any rankings sampled from the perturbed data will likely not satisfy the fairness constraint without further processing.

6.3.2 Results With Modified Score Distributions. c We show results for all datasets in Figure 5. We see that L2SQ typically outperforms Fair Queues and Greedy Fair Swaps in terms of nDCG, and that all methods are successful at maintaining the DDP below the threshold at all steps, despite the fact that the unprocessed rankings nearly always exceed the threshold.

On Resume and StackExchange, L2SQ also visibly maintains a DDP significantly below the threshold and much lower than the other policies. L2SQ also displays a higher advantage in terms of nDCG over the other policies on these two datasets. As previously indicated, we believe that L2SQ's main advantage is its ability to provide a buffer well under the threshold, allowing it to take advantage of batches which provide high nDCG only if items with already high exposure are chosen. We find that on the German Credit and AirBnB datasets, L2SQ is difficult to distinguish from the other policies.

6.4 Sensitivity to Relevance Distributions

To determine how the L2SQ model performs with respect to the degree of inherent bias, we perform an experiment systematically varying the degree of inherent bias in the synthetic dataset. We train

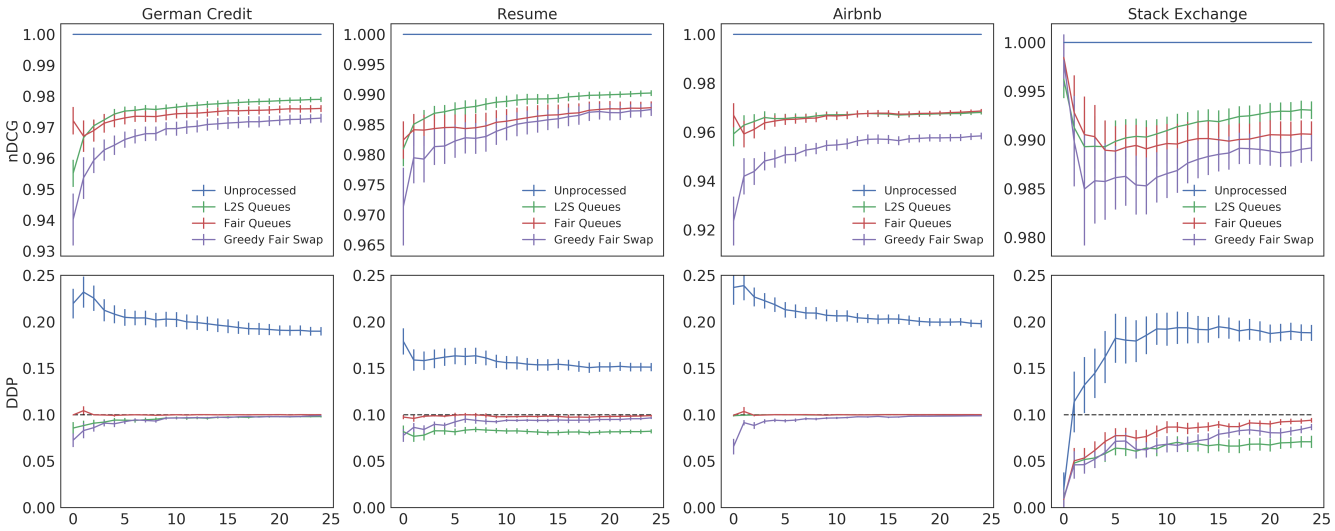

Figure 5: Results on all modified datasets over 25 timesteps.

L2SQ on 36 different synthetic datasets corresponding to six values (0.0 to 0.5 by 0.1) for the mean of the Gaussian random variable for groups 0 and 1, and the same six values for the negative mean of Gaussian random variable for groups 2 and 3. After training each model for 20 epochs, we take the model which scores best on the validation data and compare it to the other algorithms. Figure 6 visualizes the difference in the sum of nDCG scores of the batches re-ranked by each algorithm.

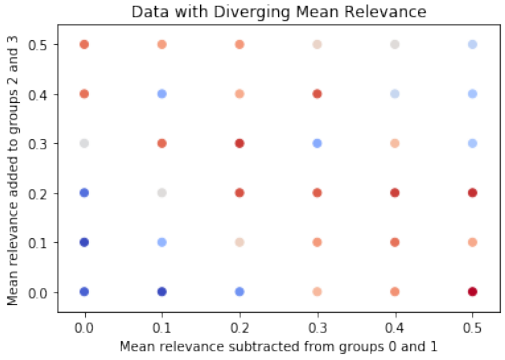

Figure 6: Red indicates L2S performed better, while blue indicates that either Greedy Fair Swap or Fair Queues performed better. The opacity of the color indicates degree of improvement.

L2SQ performs best when the groups have some spread between them, but its performance is not as good when the groups have very similar or very dissimilar scores. If we denote the total difference in means as the "spread," it seems to perform best with a spread of 0.4 - 0.7. We hypothesized that L2SQ would perform better with a wider spread, but it seems that as the spread becomes too large it struggles. In this range the model may have a limited state space to explore during training, which would inhibit its potential to learn a better policy.

7 CONCLUSION

We propose an algorithmic framework for post-processing of ranking batches while satisfying a given fairness constraint, when operating in an online environment. We evaluate two deterministic policies as well as a novel learning to search based policy L2SQ. Extensive experiments on synthetic as well as real world data sets demonstrate the effectiveness of our proposed approaches. The experiments also demonstrate that our approach can be generalized to consider multiple groups i.e., the synthetic dataset with different group sizes, StackExchange with three groups, and other datasets with four groups. Our approach is also robust to the batch size which varies among the different datasets. While we present our results using the DDP criterion, our approach can incorporate general fairness criteria and can operate on ranking models for a variety of applications.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback, and Andrew McCallum for introducing us as part of the DS696: Industry Mentorship Independent Study at UMass. This work was supported in part by the Chan Zuckerberg Initiative. The work reported here was performed in part using high performance computing equipment obtained under a grant from the Collaborative RD Fund managed by the Massachusetts Technology Collaborative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Abolfazl Asudeh, HV Jagadish, Julia Stoyanovich, and Gautam Das. 2019. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data*. 1259–1276.

[2] Alex Beutel, Jilin Chen, Tulsee Doshi, Hai Qian, Li Wei, Yi Wu, Lukasz Heldt, Zhe Zhao, Lichan Hong, Ed H. Chi, and Cristos Goodrow. 2019. Fairness in Recommendation Ranking through Pairwise Comparisons. arXiv:1903.00780 [cs.CY]

[3] Asia J Biega, Krishna P Gummadi, and Gerhard Weikum. 2018. Equity of attention: Amortizing individual fairness in rankings. In *The 41st international acm sigir conference on research & development in information retrieval*. 405–414.

[4] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. 335–336.

[5] L Elisa Celis, Anay Mehrotra, and Nisheeth K Vishnoi. 2020. Interventions for ranking in the presence of implicit bias. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 369–380.

[6] L Elisa Celis, Damian Straszak, and Nisheeth K Vishnoi. 2017. Ranking with fairness constraints. *arXiv preprint arXiv:1704.06840* (2017).

[7] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Hal Daumé III. 2015. Learning to search better than your teacher. In *International Conference on Machine Learning*.

[8] Alexandra Chouldechova and Aaron Roth. 2018. The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810* (2018).

[9] Sam Corbett-Davies and Sharad Goel. 2018. The measure and mismeasure of fairness: A critical review of fair machine learning. *arXiv preprint arXiv:1808.00023* (2018).

[10] Yifan Guan, Abolfazl Asudeh, Pranav Mayuram, HV Jagadish, Julia Stoyanovich, Gerome Miklau, and Gautam Das. 2019. Mithraranking: A system for responsible ranking design. In *Proceedings of the 2019 International Conference on Management of Data*. 1913–1916.

[11] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of Opportunity in Supervised Learning. *CoRR* abs/1610.02413 (2016).

[12] Hans Hofmann. 1994. Statlog (german credit data) data set. *UCI Repository of Machine Learning Databases* (1994).

[13] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2016. Unbiased Learning-to-Rank with Biased Feedback. arXiv:1608.04468 [cs.IR]

[14] Matthew Kay, Cynthia Matuszek, and Sean A Munson. 2015. Unequal representation and gender stereotypes in image search results for occupations. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 3819–3828.

[15] Caitlin Kuhlman, MaryAnn VanValkenburg, and Elke Rundensteiner. 2019. FARE: Diagnostics for Fair Ranking using Pairwise Error Metrics. In *The World Wide Web Conference*. 2936–2942.

[16] Juhi Kulshrestha, Motahhare Eslami, Johnnatan Messias, Muhammad Bilal Zafar, Saptarshi Ghosh, Krishna P Gummadi, and Karrie Karahalios. 2017. Quantifying search bias: Investigating sources of bias for political searches in social media. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. 417–432.

[17] Matevž Kunaver and Tomaž Požrl. 2017. Diversity in recommender systems–A survey. *Knowledge-Based Systems* 123 (2017), 154–162.

[18] Swetasudha Panda, J. Tristan, M. Wick, Haniyeh Mahmoudian, and P. Kanani. 2019. Using Bayes Factors to Control for Fairness Case Study on Learning To Rank.

[19] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[20] Tetsuya Sakai and Ruihua Song. 2011. Evaluating diversified search results using per-intent graded relevance. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 1043–1052.

[21] Piotr Sapiezynski, Wesley Zeng, Ronald E Robertson, Alan Mislove, and Christo Wilson. 2019. Quantifying the Impact of User Attention on Fair Group Representation in Ranked Lists. In *Companion Proceedings of The 2019 World Wide Web Conference*. 553–562.

[22] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge.

[23] Ashudeep Singh and Thorsten Joachims. 2018. Fairness of exposure in rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2219–2228.

[24] Ashudeep Singh and Thorsten Joachims. 2019. Policy learning for fairness in ranking. In *Advances in Neural Information Processing Systems*. 5426–5436.

[25] Julia Stoyanovich, Ke Yang, and HV Jagadish. 2018. Online set selection with fairness and diversity constraints. In *Proceedings of the EDBT Conference*.

[26] Ke Yang and Julia Stoyanovich. 2017. Measuring fairness in ranked outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. 1–6.

[27] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P Gummadi. 2019. Fairness Constraints: A Flexible Approach for Fair Classification. *J. Mach. Learn. Res.* 20, 75 (2019), 1–42.

[28] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rogriguez, and Krishna P Gummadi. 2017. Fairness constraints: Mechanisms for fair classification. In *Artificial Intelligence and Statistics*. PMLR, 962–970.

[29] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. 2017. Fa*ir: A fair top-k ranking algorithm. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1569–1578.

[30] Meike Zehlike and Carlos Castillo. 2020. Reducing disparate exposure in ranking: A learning to rank approach. In *Proceedings of The Web Conference 2020*. 2849–2855.

[31] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning fair representations. In *International Conference on Machine Learning*. 325–333.