Fairness-Aware Ranking in Search & Recommendation Systems with Application to LinkedIn Talent Search

Sahin Cem Geyik, Stuart Ambler, Krishnaram Kenthapadi
LinkedIn Corporation, USA

ABSTRACT

We present a framework for quantifying and mitigating algorithmic bias in mechanisms designed for ranking individuals, typically used as part of web-scale search and recommendation systems. We first propose complementary measures to quantify bias with respect to protected attributes such as gender and age. We then present algorithms for computing fairness-aware re-ranking of results. For a given search or recommendation task, our algorithms seek to achieve a desired distribution of top ranked results with respect to one or more protected attributes. We show that such a framework can be tailored to achieve fairness criteria such as *equality of opportunity* and *demographic parity* depending on the choice of the desired distribution. We evaluate the proposed algorithms via extensive simulations over different parameter choices, and study the effect of fairness-aware ranking on both bias and utility measures. We finally present the online A/B testing results from applying our framework towards representative ranking in LinkedIn Talent Search, and discuss the lessons learned in practice. Our approach resulted in tremendous improvement in the fairness metrics (nearly three fold increase in the number of search queries with representative results) without affecting the business metrics, which paved the way for deployment to 100% of *LinkedIn Recruiter* users worldwide. Ours is the first large-scale deployed framework for ensuring fairness in the hiring domain, with the potential positive impact for more than 630M LinkedIn members.

1 INTRODUCTION

Ranking algorithms form the core of search and recommendation systems for several applications such as hiring, lending, and college admissions. Recent studies show that ranked lists produced by a biased machine learning model can result in systematic discrimination and reduced visibility for an already disadvantaged group [17, 23, 35] (e.g., disproportionate association of higher risk scores of recidivism with minorities [3], over/under-representation and racial/gender stereotypes in image search results [31], and incorporation of gender and other biases as part of algorithmic tools [9, 11]). One possible reason is that machine learned prediction models trained on datasets that exhibit existing societal biases end up learning them and can reinforce such bias in their results, potentially even amplifying the effect.

In this paper, we present a framework for quantifying and mitigating algorithmic bias in systems designed for ranking individuals. Given fairness requirements expressed in terms of a desired distribution over protected attribute(s) (e.g., gender, age, or their combination), we propose algorithms for re-ranking candidates scored/returned by a machine learned model to satisfy the fairness constraints. Our key contributions include:

- Proposal of fairness-aware ranking algorithms towards mitigating algorithmic bias. Our methodology can be used to achieve fairness criteria such as *equality of opportunity* [26] and *demographic parity* [17] depending on the choice of the desired distribution over protected attribute(s).
- Proposal of complementary measures for quantifying the fairness of the ranked candidate lists.
- Extensive evaluation of the proposed algorithms via simulations over a wide range of ranking scenarios and attributes with different cardinalities (possible number of values).
- Online A/B test results of applying our framework for achieving representative ranking in LinkedIn Talent Search, and the lessons learned in practice. Our approach resulted in tremendous improvement in the fairness metrics (nearly three fold increase in the number of search queries with representative results) without statistically significant change in the business metrics, which paved the way for deployment to 100% of *LinkedIn Recruiter* users worldwide.

The rest of the paper is organized as follows. We first provide measures for evaluating bias and fairness in ranked lists in §2. Next, we present fairness-aware re-ranking algorithms in §3, followed by their extensive evaluation and results from deployment in LinkedIn Talent Search in §4. We discuss related work as well as a comparison of our approach to previous work on fairness-aware ranking in §5. We conclude the paper and present future work in §6.

2 MEASURING ALGORITHMIC BIAS

We first discuss the intuition underlying our bias measurement approach, and then present measures for quantifying bias in ranking that are complementary to each other.

2.1 Intuition Underlying Bias Quantification

Our bias measurement and mitigation approach assume that in the ideal setting, the set of top ranked results for a search or recommendation task should follow a desired distribution over a protected attribute such as gender or age. This desired distribution can be computed in many ways including, but not limited to, adhering to the corresponding distribution over a baseline population, a legal mandate, or a voluntary commitment (e.g., [1, 2, 38]). Note that our framework allows fairness-aware re-ranking over multiple attributes by considering the cross-product of possible values, e.g., adhering to a desired distribution over all possible (gender, age group) pairs. As we discuss in §3.3, we can achieve fairness criteria such as *equal opportunity* [26] and *demographic parity* [17] depending on the choice of the desired distribution.

2.2 Measures for Bias Evaluation

We next describe measures for evaluating bias in recommendation and search systems. We use the notations listed in Table 1.

2.2.1 Measure based on Top-k Results. Our first measure computes the extent to which the set of top k ranked results for a search or

This paper has been accepted for publication at ACM KDD 2019.

Table 1: Key Notations

Notation	Represents
r	A search request or a recommendation task
$A = \{a_1, \ldots, a_l\}$	Set of disjoint protected attribute values (each candidate has exactly one value in A); Note that we denote the attribute value for candidate x as $A(x)$, by abuse of notation.
τ_r	Ranked list of candidates for r; $\tau_r[j]$ denotes j^{th} candidate; τ_r^k denotes the first k candidates in τ_r
p_{q,r,a_i}	Desired proportion of candidates with attribute value a_i that should be in the ranked list
p_{τ_r,r,a_i}	Proportion of candidates in τ_r with value a_i, i.e., $\frac{\{x \in \tau_r \mid A(x)=a_i\}}{\lvert\tau_r\rvert}$

recommendation task differ over an attribute value with respect to the desired proportion of that attribute value.

Definition 2.1. Given a ranked list τ_r of candidates for a search request r, the *skew* of τ_r for an attribute value a_i is:

$$Skew_{a_i}@k(\tau_r) = \log_e\left(\frac{p_{\tau_r^k,r,a_i}}{p_{q,r,a_i}}\right). \quad (1)$$

In other words, $Skew_{a_i}@k$ is the (logarithmic) ratio of the proportion of candidates having the attribute value a_i among the top k ranked results to the corresponding desired proportion for a_i. A negative $Skew_{a_i}@k$ corresponds to a lesser than desired representation of candidates with value a_i in the top k results, while a positive $Skew_{a_i}@k$ corresponds to favoring such candidates. We utilize the log to make the skew values symmetric around origin with respect to ratios for and against a specific attribute value a_i. For example, the ratio of the proportions being 2 or $\frac{1}{2}$ corresponds to the same skew value in magnitude, but with opposite signs. Note that the calculation might need some adjustment to prevent a case of divide-by-zero or log(0).

Consider the gender attribute (with values {a_1 = male, a_2 = female}) as an example. Suppose that, for a given search task, the desired proportions are obtained based on the set of qualified candidates which consists of $32K$ males and $48K$ females ($80K$ total, hence desired ratios are $p_{q,r,\text{male}} = 0.4$ and $p_{q,r,\text{female}} = 0.6$). If the set of top 100 ranked results for this task consists of 20 males and 80 females, then, $Skew_{male}@100 = \log_e(\frac{20}{100} / \frac{32K}{80K}) = \log_e(0.5) \approx -0.3$.

$Skew_{a_i}@k$ measure is intuitive to explain and easy to interpret. In the above example, we can infer that males are represented 50% less than the desired representation. However, $Skew_{a_i}@k$ has the following disadvantages. (1) It is defined for a single attribute value, and hence we may need to compute the skew value for all possible values of the protected attribute. (2) It depends on k and has to be computed for different k values to fully understand the extent of the bias. While certain choices of k may be suitable based on the application (e.g., $k = 25$ may be meaningful to measure skew in the first page of results for a search engine that displays 25 results in each page), a measure that takes into account all candidates in a ranked list may be desirable to provide a more holistic view of fairness.

To deal with the first problem above, we introduce two more measures which give a combined view of $Skew@k$ measure:

- *MinSkew@k*: For a search request r, *MinSkew@k* provides the minimum skew among all attribute values,

$$MinSkew@k(\tau_r) = min_{a_i \in A} Skew_{a_i}@k(\tau_r). \quad (2)$$

- *MaxSkew@k*: For a search request r, *MaxSkew@k* provides the maximum skew among all attribute values,

$$MaxSkew@k(\tau_r) = max_{a_i \in A} Skew_{a_i}@k(\tau_r). \quad (3)$$

MinSkew and MaxSkew have the following interpretation. MinSkew signifies the *worst disadvantage in representation* given to candidates with a specific attribute value while MaxSkew signifies the *largest unfair advantage* provided to candidates with an attribute value. Since both $\sum p_{\tau_r^k,r,a_i} = 1$ and $\sum p_{q,r,a_i} = 1$, it follows that for any ranked list, and for any k, $MinSkew@k \leq 0$ and $MaxSkew@k \geq 0$.

Next, we present a ranking measure that addresses the second problem with skew measure as presented above.

2.2.2 Ranking Measure. Several measures for evaluating the fairness of a ranked list have been explored in the information retrieval literature [40]. In this paper, we adopt a ranking bias measure based on Kullback-Leibler (KL) divergence [33]. Let $D_{\tau_r^i}$ and D_r denote the discrete distribution assigning to each attribute value in A, the proportion of candidates having that value, over the top i candidates in the given ranked list τ_r and over the desired distribution respectively. Given these two distributions, we compute the KL-divergence and then obtain a normalized discounted cumulative variant, similar to [40]. This measure is non-negative, with a larger value denoting greater divergence between the two distributions. It equals 0 in the ideal case of the two distributions being identical for each position i.

Definition 2.2. Given a ranked list τ_r of candidates for a search request r, the *normalized discounted cumulative KL-divergence* (NDKL) of τ_r is:

$$NDKL(\tau_r) = \frac{1}{Z} \sum_{i=1}^{|\tau_r|} \frac{1}{\log_2(i+1)} d_{KL}(D_{\tau_r^i} || D_r), \quad (4)$$

where, $d_{KL}(D_1||D_2) = \sum_j D_1(j) \log_e \frac{D_1(j)}{D_2(j)}$ is the KL-divergence of distribution D_1 with respect to distribution D_2 and $Z = \sum_{i=1}^{|\tau_r|} \frac{1}{\log_2(i+1)}$.

Note that $d_{KL}(D_{\tau_r^i}||D_r)$ corresponds to a weighted average of $Skew@i$ over all attribute values. While having the benefit of providing a single measure of bias over all attribute values and a holistic view over the whole ranked list, the NDKL measure has the following disadvantages. (1) It cannot differentiate between bias of equal extent, but in opposite directions. For example, given an equal desired proportion of males and females (i.e., $p_{q,r,\text{male}} = p_{q,r,\text{female}} = 0.5$), NDKL would be the same irrespective of whether males or females are being under-represented in the top ranked results by the same extent. Thus, the measure does not convey which attribute value is being unfairly treated (Skew measure is more suitable for this). (2) It is not as easy to interpret as the skew measure.

3 FAIRNESS-AWARE RANKING ALGORITHMS

We next present a discussion of the desired properties when designing fair ranking algorithms, followed by a description of our proposed algorithms.

3.1 Discussion of Desired Properties

As presented in §2, we assume that for each attribute value a_i, it is desirable for a fair ranking algorithm to include candidates possessing a_i with a proportion as close as possible to p_{q,r,a_i} (for brevity, we also use the term p_{a_i} to mean the desired proportion of candidates possessing attribute value a_i). While one can argue

that for a representation proportion of $p_{\tau_r,r,a_i} > p_{q,r,a_i}$, we are still "fair" to a_i, a model that achieves such a recommendation proportion causes unfairness to other $a_j \in A$ where $a_j \neq a_i$, since $\sum_{a\in A} p_{q,r,a} = \sum_{a\in A} p_{\tau_r,r,a} = 1$. This is the case because the attribute values are disjoint, i.e., each candidate possesses exactly one value of a given attribute.

Furthermore, it is not enough for a ranked list to have a representation proportion (actual proportion with which we show candidates for this attribute value, i.e., p_{τ_r,r,a_i} from Table 1) close to desired proportions in the top-k candidates, but rather should have these proportions for all k, computed over candidates up to k^{th} index (e.g. for top-1, top-2, ... , top-k candidates etc.), since recommending a candidate earlier vs. later has a significant effect on the response of the user [29]. Therefore, we suggest that a ranking algorithm should generate a ranked list τ_r for a recommendation task r, with the following properties:

$$\forall k \leq |\tau_r| \;\&\; \forall a_i \in A,\ \lceil p_{a_i} \cdot k \rceil \geq count_k(a_i)\,,\ and, \tag{5}$$

$$\forall k \leq |\tau_r| \;\&\; \forall a_i \in A,\ count_k(a_i) \geq \lfloor p_{a_i} \cdot k \rfloor\,, \tag{6}$$

where $count_k(a_i)$ represents the number of candidates from attribute value a_i recommended among the top k spots. Among the above two conditions, the more important one for fairness purposes is Eq. 6, since it guarantees a minimum representation for an attribute value (Eq. 5 is in general used as a guidance so that no disproportionate advantage is given to a specific attribute value, since this would cause disadvantage to other attribute values). Due to this, we further define a measure of *feasibility* for a ranking algorithm as follows:

Definition 3.1. A mitigation algorithm is infeasible if:

$$\exists\, r\ s.t.\ \exists\, k \leq |\tau_r| \;\&\; a_i \in A,\ count_k(a_i) < \lfloor p_{a_i} \cdot k \rfloor\,. \tag{7}$$

This means that for the mitigation algorithm, there is at least one search request r, such that the generated ranking list τ_r breaks the condition $\lfloor p_{a_i} \cdot k \rfloor \leq count_k(a_i)$ for at least one k. A new set of measures similar to those introduced in §2.2 can also be defined for the feasibility condition as follows:

- *InfeasibleIndex:* For τ_r, in how many indices $k \leq |\tau_r|$, is (6) violated?

$$\text{InfeasibleIndex}_{\tau_r} = \sum_{k\leq|\tau_r|} \mathbb{1}(\exists a_i \in A,\ s.t.\ count_k(a_i) < \lfloor p_{a_i} \cdot k \rfloor). \tag{8}$$

  While this value depends on the size of the ranked list τ_r, it can be normalized based on the length.
- *InfeasibleCount:* For τ_r, for how many attribute values $a_i \in A$, and in how many indices $k < |\tau_r|$, is (6) violated?

$$\text{InfeasibleCount}_{\tau_r} = \sum_{k\leq|\tau_r|} \sum_{a_i\in A} \mathbb{1}(count_k(a_i) < \lfloor p_{a_i} \cdot k \rfloor)\,. \tag{9}$$

  While this value depends on the size of the ranked list τ_r, as well as the number of possible attribute values, i.e. $|A|$, it can again be normalized.

Next, we present our proposed set of algorithms for obtaining fair re-ranked lists. Note that the proposed algorithms assume that there are enough candidates for each attribute value, which may not always be the case in search and recommendation systems. However, it would be easy to modify all the proposed algorithms to have a fallback mechanism to choose another candidate from the next-best attribute value (for fairness purposes). Finally, to avoid repetition, we have listed the combined set of inputs and outputs for all the algorithms in Table 2.

Table 2: Collective Inputs and Outputs of Algorithms 1 through 3

Inputs	$\boldsymbol{a}$: Possible attribute values indexed as a_i, with each attribute value having n candidates with scores $s_{i,j}$. Candidate list for each attribute value is assumed to be ordered by decreasing scores, i.e., $a_{i,j}$ refers to j^{th} element of attribute value a_i, with score $s_{i,j}$. $\forall k, l : k \leq l \iff s_{i,k} \geq s_{i,l}$
	$\boldsymbol{p}$: A categorical distribution where p_{a_i} indicates the desired proportion of candidates with attribute value a_i
	recMax: Number of desired results
Output	An ordered list of attribute value ids and scores

Algorithm 1 Score Maximizing Greedy Mitigation Algorithm (DetGreedy)

counts = [0 ∀ $a_i \in a$]
rankedAttList = [ ]
rankedScoreList = [ ]
for index ind ∈ [1:recMax] **do**
  belowMinimums = [a_i where counts[a_i] < $\lfloor ind \cdot p_{a_i} \rfloor$]
  belowMaximums = [a_i where counts[a_i] ≥ $\lfloor ind \cdot p_{a_i} \rfloor$ and counts[a_i] < $\lceil ind \cdot p_{a_i} \rceil$]
  if belowMinimums ≠ ∅ **then**
    nextAtt = argmax$_{a_i \in \text{belowMinimums}}$ $s_{i,counts[i]}$
  else
    nextAtt = argmax$_{a_i \in \text{belowMaximums}}$ $s_{i,counts[i]}$
  rankedAttList[ind] = nextAtt
  rankedScoreList[ind] = $s_{nextAtt,\ counts[nextAtt]}$
  counts[nextAtt]++
return [rankedAttList, rankedScoreList]

3.2 Ranking Algorithms

3.2.1 Baseline algorithm with no mitigation (Vanilla). Our baseline ranking approach orders candidates in the descending order of score assigned by the ML model.

We next present four deterministic algorithms towards the goal of satisfying the conditions given in Eq. 5 and Eq. 6.

3.2.2 Score maximizing greedy mitigation algorithm (DetGreedy). Deterministic Greedy (DetGreedy) algorithm (Alg. 1) works as follows: If there are any attribute values with a representation below their minimum requirements (Eq. 6), choose the one with the highest next score among them. Otherwise, choose the attribute value with the highest next score among those that have not yet met their maximum requirements (Eq. 5). In Alg. 1, *counts* keep the number of candidates shown from attribute value a_i up until the current index ($count_k(a_i)$), $s_{i,counts[i]}$ basically means the score of the next candidate (not yet shown) for attribute value a_i, since as given in the **Inputs** (Table 2), the set of candidates is already ordered by decreasing score, and bucketized per attribute value a_i.

3.2.3 Score maximizing greedy conservative mitigation algorithm (DetCons) and its relaxed variant (DetRelaxed). While DetGreedy generates rankings with as high score candidates as possible in the ranked list, it may easily fall into an infeasible state (Definition 3.1). Hence, it may be desirable to incorporate look-ahead to account for attribute values requiring one more element soon within the ranking, which is the basis for our next two algorithms. Deterministic Conservative (DetCons) algorithm and its relaxed version (DetRelaxed), described in Alg. 2, work as follows:

Algorithm 2 Score Maximizing Greedy Conservative Mitigation Algorithm (DetCons) and its Relaxed variant (DetRelaxed)

counts = [0 $\forall\ a_i\ \in\ a$]
rankedAttList = [ ]
rankedScoreList = [ ]
for index ind $\in$ [1:recMax] **do**
  belowMinimums = [a_i where counts[a_i] < $\lfloor ind \cdot p_{a_i} \rfloor$]
  belowMaximums = [a_i where counts[a_i] $\geq$ $\lfloor ind \cdot p_{a_i} \rfloor$ and counts[a_i] < $\lceil ind \cdot p_{a_i} \rceil$]
  if belowMinimums $\neq \emptyset$ **then**
    nextAtt = argmax$_{a_i \in \text{belowMinimums}}$ $s_{i,counts[i]}$
  else
    if ***DetCons*** **then**
      nextAtt = argmin$_{a_i \in \text{belowMaximums}}$ $\frac{\lceil ind \cdot p_{a_i} \rceil}{p_{a_i}}$
    else if ***DetRelaxed*** **then**
      minSteps = min($\lceil \frac{\lceil ind \cdot p_{a_i} \rceil}{p_{a_i}} \rceil\ \forall\ a_i\ \in$ belowMaximums)
      minStepAtt = [$a_i\ \in$ belowMaximums where $\lceil \frac{\lceil ind \cdot p_{a_i} \rceil}{p_{a_i}} \rceil$ == minSteps]
      nextAtt = argmax$_{a_i \in \text{minStepAtt}}$ $s_{i,counts[i]}$
  rankedAttList[ind] = nextAtt
  rankedScoreList[ind] = $s_{nextAtt,\ counts[nextAtt]}$
  counts[nextAtt]++
return [rankedAttList, rankedScoreList]

(1) If there are any attribute values that are below their minimum requirements (Eq. 6), choose the one with the maximum next score among them.
(2) Otherwise, among those attribute values that are not yet violating their maximum requirements (Eq. 5), choose the one which minimizes $\frac{\lceil p_{a_i} \cdot k \rceil}{p_{a_i}} - k$ (in Alg. 2, $-k$ is omitted since it is a constant for the current index). The minimizer a_i of this equation is the one that will reach its minimum requirement in the least amount of fractional recommendations (for DetCons, we will next introduce DetRelaxed which works on the fact that we do not do fractional recommendations). This is due to the fact that, without loss of generality: $\lceil p_{a_i} \cdot k \rceil - \lfloor p_{a_i} \cdot k \rfloor \leq 1$, hence the current maximum ($\lceil p_{a_i} \cdot k \rceil$) of an attribute value a_i will be the next minimum representation requirement (Eq. 6).

The relaxed version of DetCons algorithm uses the integer nature of recommendation indices to increase the overall score ordering in the output, and is called Deterministic Relaxed (DetRelaxed) algorithm. Contrary to DetCons, in the second step above, it chooses, among those attribute values that have not yet met their maximum requirements (Eq. 5), a subset which minimizes $\lceil \frac{\lceil p_{a_i} \cdot k \rceil}{p_{a_i}} \rceil - k$. This gives the number of steps till an attribute value a_i will reach a next step in its minimum requirement, i.e. find $k' - k$ where $\lfloor p_{a_i} \cdot k' \rfloor = \lfloor p_{a_i} \cdot k \rfloor + 1$. Among those that minimize this condition, we choose the next attribute value as the one which has the highest score for the next candidate. The algorithm is more relaxed in a sense that the attribute value that would have been utilized in DetCons is definitely one of the candidates in DetRelaxed, but we may choose a candidate with a higher score from those attribute values that are now equivalent in terms of the number of indices to the next minimum requirements, since the condition to be chosen is relaxed using the integer upper bound.

While the above three algorithms are designed towards meeting the conditions given in Eq. 5 and Eq. 6, we can show that DetGreedy is not feasible in certain settings. Although we have not been able to prove that DetCons and DetRelaxed are always feasible, our simulation results (§4) suggest that this may indeed be the case.

Theorem 3.2. *The algorithms DetGreedy, DetCons, and DetRelaxed* ***are feasible*** *whenever the number of possible attribute values for the protected attribute is less than 4, i.e., for* $|A| \leq 3$. *DetGreedy is* ***not feasible*** *for* $|A| \geq 4$.

Proof is available in the appendix (§A.3). Next, we present a provably feasible algorithm for fairness-aware ranking, which follows a constrained sorting scheme.

3.2.4 Feasible mitigation algorithm which employs interval constrained ordering (DetConstSort). Deterministic Constrained Sorting (DetConstSort) algorithm (Alg. 3) also aims to enforce the conditions given in Eq. 5 and Eq. 6. However, contrary to the three greedy approaches listed previously, DetConstSort waits for multiple indices of recommendation before deciding on the next attribute value to get a candidate from, and may change its previous decisions to improve the score ordering. The algorithm works as follows:

(1) Increase a counter value (starting with 0) until at least one attribute value has increased its minimum value requirement per Eq. 6. If there is more than one such attribute value, order them according to descending score of their next candidates.
(2) Go over the set of ranked attribute values which have increased their minimum requirement, and for each one:
  (a) Insert the next candidate from the attribute value to the next empty index in the recommendation list.
  (b) Swap this candidate towards earlier indices in the list until:
    - Either the score of the left candidate (candidate in the earlier index) is larger, or,
    - Maximum index of the left candidate will be violated due to swap (maximum index of a candidate is the maximum index which the candidate from this attribute value can be placed at so that we meet the condition of feasibility, i.e. $count_k(a_i) \geq \lfloor p_{a_i} \cdot k \rfloor$ per Definition 3.1.

DetConstSort algorithm also solves a more general interval constrained sorting problem where we want to maximize the sorting quality, subject to constraints that some elements cannot go beyond a specific index, as long as there is a solution that satisfies constraints. If we describe sorting quality with a measure such as discounted cumulative gain [28] (or any monotonic function that captures the idea that higher scoring candidates should be ranked higher), then a more formal definition of interval constrained sorting can be given as below:

Definition 3.3. Given a set of objects o_i from the set o, maximum index constraints m_i, and values v_i belonging to each object o_i, interval constrained sorting solves the following:

$$maximize \sum \frac{v_i}{\log(r_i + 1)}\ ,\ s.t.$$
$$\forall j \text{ and } k,\ r_j \neq r_k,$$
$$\forall j,\ r_j \leq |O| \text{ and } r_j \leq m_j\ .$$

Theorem 3.4. *DetConstSort algorithm is feasible per Definition 3.1.*

Proof is available in the appendix (§A.4).

3.3 Mapping from Fairness Notions to Desired Attribute Distributions

Our fairness-aware ranking approach aims to achieve representativeness as determined by the desired distribution over a protected attribute (or multiple attributes, by considering the cross-product

Algorithm 3 Feasible Mitigation Algorithm Based on Interval Constrained Sorting (DetConstSort)

counts = [0 ∀ $a_i \in a$]
minCounts = [0 ∀ $a_i \in a$]
rankedAttList = [ ]
rankedScoreList = [ ]
maxIndices = [ ]
lastEmpty = 0
ind = 0
while lastEmpty ≤ recMax **do**
  // Fill the array until max number of recommendations
  ind++
  tempMinCounts = [$\lfloor ind \cdot p_{a_i} \rfloor$ ∀ a_i]
  changedMins = [a_i where minCounts[a_i] < tempMinCounts[a_i]]
  if changedMins ≠ ∅ **then**
    ordChangedMins = sort changedMins BY $s_{a_i,\ counts[a_i]}$ descending
    for a_i ∈ ordChangedMins **do**
      rankedAttList[lastEmpty] = a_i
      rankedScoreList[lastEmpty] = $s_{a_i,\ counts[a_i]}$
      maxIndices[lastEmpty] = ind
      start = lastEmpty
      while start > 0 & maxIndices[start - 1] ≥ start & rankedScoreList[start-1] < rankedScoreList[start] **do**
        swap(maxIndices[start - 1], maxIndices[start])
        swap(rankedAttList[start - 1], rankedAttList[start])
        swap(rankedScoreList[start - 1], rankedScoreList[start])
        start−−
      counts[a_i]++
      lastEmpty++
    minCounts = tempMinCounts
return [rankedAttList, rankedScoreList]

of possible values). Next, we discuss how our framework can be used to achieve fairness notions such as equal opportunity [26] and demographic parity [17] through a careful selection of the desired distribution.

3.3.1 Achieving Equal Opportunity. A predictor function $\hat{Y}$ is said to satisfy equal opportunity [26] with respect to a protected attribute A and true outcome Y, if the predictor and the protected attribute are independent conditional on the true outcome being 1 (favorable). That is,

$$p(\hat{Y} = 1|A = a_1, Y = 1) = \ldots = p(\hat{Y} = 1|A = a_l, Y = 1)\ . \quad (10)$$

For a search or recommendation task, we can roughly map our framework to the above fairness notion by assuming that the set of candidates that match the criteria (either explicitly specified in the search request or implicitly for the recommendation task) as "qualified" for the task. The true outcome being positive ($Y = 1$) corresponds to a candidate matching the search request criteria (or equivalently, being "qualified" for the search request), while the prediction being positive ($\hat{Y} = 1$) corresponds to a candidate being presented in the top ranked results for the search request. The equal opportunity notion requires that the fraction of qualified candidates that are included in the top ranked results does not depend on the protected attribute, or equivalently that the proportion of candidates belonging to a given value of the attribute does not vary between the set of qualified candidates and the set of top ranked results. In our framework, this requirement can be met by selecting the desired distribution to be the distribution of the *qualified* candidates over the protected attribute. Further, since the top ranked results are chosen from the set of qualified candidates (that is, $\hat{Y} = 0$ whenever $Y = 0$), the above choice of the desired distribution can also be viewed as meeting the requirement of equalized odds [26].

3.3.2 Achieving Demographic Parity. Demographic parity (or statistical parity) [17] requires that the predictor function $\hat{Y}$ be independent of the protected attribute A, that is,

$$p(\hat{Y} = 1|A = a_1) = \ldots = p(\hat{Y} = 1|A = a_l), \text{ and,}$$
$$p(\hat{Y} = 0|A = a_1) = \ldots = p(\hat{Y} = 0|A = a_l)\ . \quad (11)$$

In our framework, we can show that this requirement can be met by selecting the desired distribution to be the distribution of *all* candidates over the protected attribute (following a similar argument as in §3.3.1). Demographic parity is an important consideration in certain application settings, although it does not take qualifications into account and is known to have limitations (see [17, 26]). For example, in the case of gender, demographic parity would require that the top results always reflect the gender distribution over all candidates, irrespective of the specific search or recommendation task.

4 EVALUATION AND DEPLOYMENT IN PRACTICE

In this section, we evaluate our proposed fairness-aware ranking framework via both offline simulations, and through our online deployment in *LinkedIn Recruiter* application.

4.1 Simulation Results

Next, we present the results of evaluating our proposed fairness-aware re-ranking algorithms via extensive simulations. Rather than utilizing a real-world dataset, we chose to use simulations for the following reasons:

(1) **To be able to study settings where there could be several possible values for the protected attribute.** Our simulation framework allowed us to evaluate the algorithms over attributes with up to 10 values (e.g., <gender, age group> which could assume 9 values with three gender values (male, female, and other/unknown) and three age groups), and also study the effect of varying the number of possible attribute values. In addition, we generated many randomized settings covering a much larger space of potential ranking situations, and thereby evaluated the algorithms more comprehensively.
(2) **Evaluating the effect of re-ranking on a utility measure in a dataset collected from the logs of a specific application is often challenging due to *position bias* [29].** Utilizing a simulation framework allows random assignment of relevance scores to the ranked candidates (to simulate the scores of a machine learned model) and directly measure the effect of fairness-aware re-ranking as compared to score based ranking.

Our simulation settings are listed in the appendix (§A.1). Figures 1 through 4 give the bias and utility results as a function of the number of attribute values for the proposed algorithms per the simulation framework.

From Figure 1, we can see that all our proposed algorithms are feasible for attributes with up to 3 possible values (which is in confirmation with our feasibility results (§3)). We observed similar results for InfeasibleCount measure (Eq. 9; results given in §A.2). We observe that DetConstSort is also feasible for all values of $|A|$

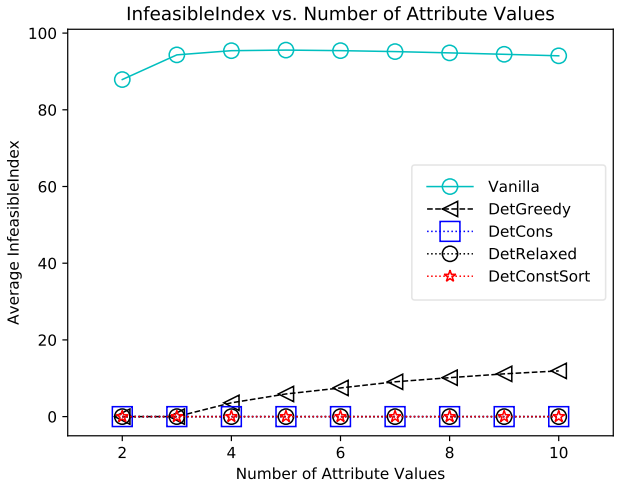

Figure 1: InfeasibleIndex Measure Results

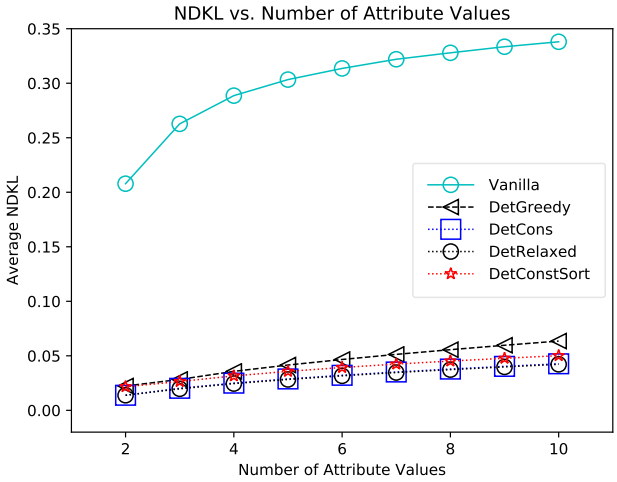

Figure 3: NDKL Measure Results

(in agreement with the theorem in §3). Furthermore, for DetGreedy, InfeasibleIndex measure increases with the number of possible attribute values, since it becomes harder to satisfy Eq. 6 for a large number of attribute values. We can also see that both DetCons and DetRelaxed are feasible for all values of $|A|$, which, although not proven, give strong evidence to their general feasibility.

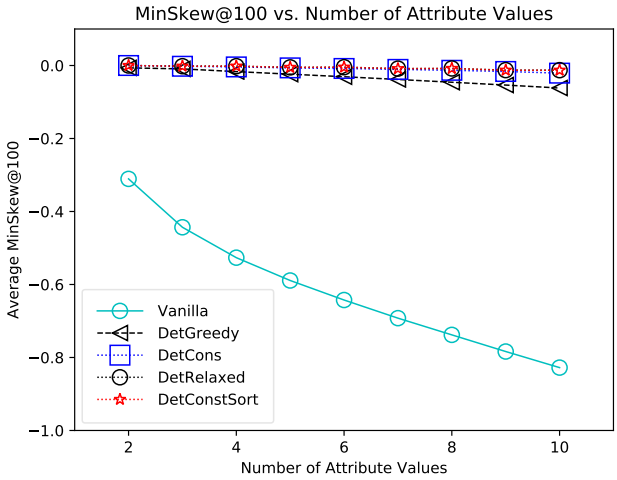

Figure 2: MinSkew@100 Measure Results

Figure 2 presents the results for MinSkew@100 measure. We observed similar results for MaxSkew measure (Eq. 3; results given in §A.2). DetCons, DetRelaxed, and DetConstSort algorithms perform quite similarly, and overall better than DetGreedy, as expected. All the fairness-aware algorithms perform much better compared to the baseline score-based (vanilla) ranking.

The results for NDKL measure, presented in Figure 3, show that the look-ahead algorithms, DetCons and DetRelaxed, perform slightly better than DetConstSort.

For utility evaluation, we computed the NDCG@100 of the generated rankings to see whether re-ranking causes a large deviation from a ranking strategy based fully on the relevance scores. Figure 4 shows that DetGreedy performs significantly better compared to the rest of fairness-aware ranking algorithms in terms of utility. DetConstSort also performs slightly better compared to the look-ahead algorithms (DetCons and DetRelaxed). Note that the vanilla algorithm ranks purely based on scores, and hence has a constant NDCG of 1.

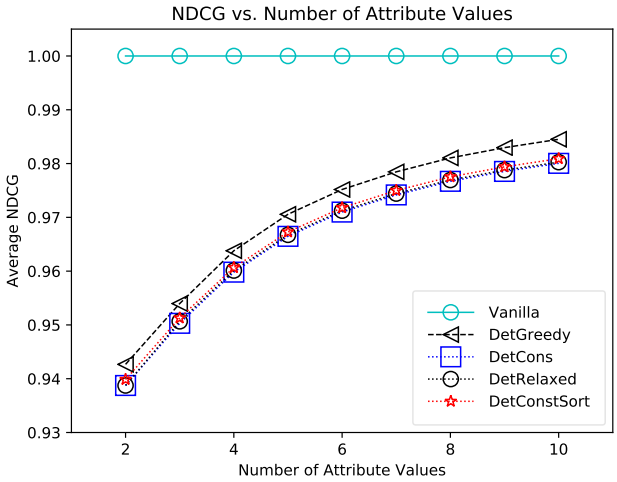

Figure 4: NDCG@100 Measure Results

Overall, DetGreedy has very competitive performance in terms of fairness measures and generates ranked lists with the highest utility. However, if the requirements of minimum representation for each attribute value are strict, we would be confined to DetCons, DetRelaxed, and DetConstSort (which happens to be the only algorithm we have theoretically proven to be feasible). Among those algorithms that did generate consistently feasible rankings in our simulations, DetConstSort performed slightly better in terms of utility. In terms of fairness measures though, we did not observe considerable difference amongst DetCons, DetRelaxed, and DetConstSort. In summary, there is no single "best" algorithm, and hence it would be desirable to carefully study the fairness vs. utility

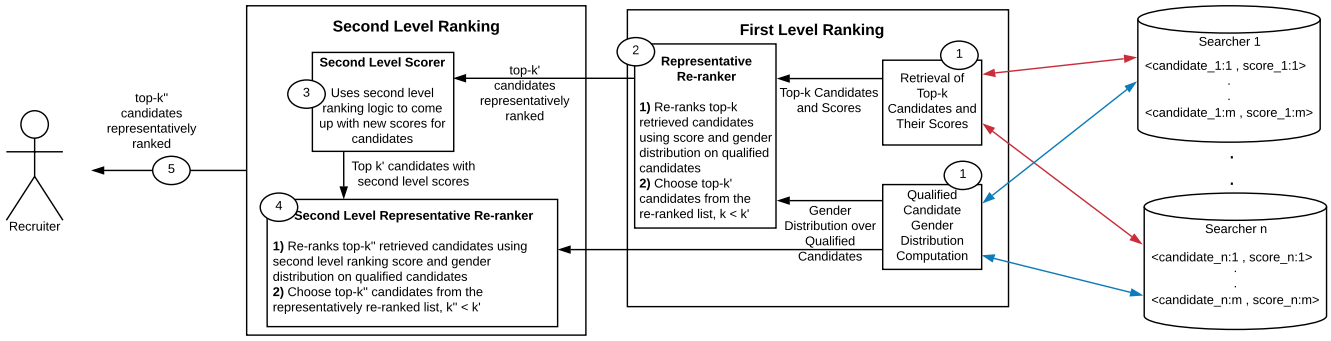

Figure 5: Online Architecture for Gender-Representative Ranking at LinkedIn

trade-offs in the application setting (e.g., by performing A/B testing) and thereby select the most suitable of these algorithms.

4.2 Online A/B Testing Results and Deployment in LinkedIn Talent Search

We have implemented the proposed framework as part of the *LinkedIn Recruiter* product to ensure gender-representative ranking of candidates. This product enables recruiters and hiring managers to source suitable talent for their needs, by allowing them to perform search and reach out to suitable candidates. Given a search request, this system first retrieves the candidates that match the request out of a pool of hundreds of millions of candidates, and then returns a ranked list of candidates using machine-learned models in multiple passes (see Figure 5, explained in §A.5). The desired gender distribution over the recommended candidate list is computed based on the number of members within each gender group among those candidates that meet (i.e., qualify for) the search criteria provided by the user of the system (recruiter) for each search request. The candidate set retrieval and scoring, as well as the computation of the desired distribution, is performed in a distributed manner using LinkedIn's *Galene* search engine [36]. Computing the desired distribution in this manner can be thought of as corresponding to achieving *equality of opportunity* per discussion in §3.3. We utilized Algorithm 1 (*DetGreedy*) in our online deployment due to its implementation simplicity and practicality considerations with A/B testing multiple algorithms (such as ensuring sufficient statistical power). Also, we observed in §4.1 that it provided the highest utility and good performance in terms of fairness measures, especially for protected attributes with low cardinality like gender (per Theorem 3.2, DetGreedy is feasible for attributes with up to three values, and gender fits this description).

The results of the A/B test which we performed over three weeks within 2018 with hundreds of thousands of Recruiter users is presented in Table 3. In this experiment, a randomly chosen 50% of Recruiter users were presented with results from the fairness-aware ranking approach while the rest were presented with the vanilla ranking of candidates based on the scores from the ML model optimized for the likelihood of making a successful hire. Please refer [21] and the references therein for a detailed description of ML models used in LinkedIn Talent Search. Our fair re-ranking approach has ensured that more than **95% of all the searches are representative** of any gender compared to the qualified population of the search (i.e., the ranking is *feasible* per Definition 3.1 in 95% of the cases), which is nearly a **3X improvement**. Furthermore, *MinSkew* (Skew for the most disadvantaged gender group within the results of a search query) over top 100 candidates, averaged over all search requests, is now very close to 0 (we achieved similar results over top 25, top 50, etc., and for other fairness measures). In other words, ranked lists of candidates presented are representative in practice. We did not observe any statistically significant change in business metrics, such as the number of inMails sent [messages from recruiters to candidates] or inMails accepted [messages from recruiters to candidates, answered back with positive responses] (only relative values are presented in Table 3), meaning that ensuring representation did not negatively impact the customers' success metrics or quality of the presented candidates for our application. Based on these results, we decided to ramp the re-ranking approach to 100% of Recruiter users worldwide. We direct the interested reader to our engineering blog post [22] for further details.

Table 3: Online A/B Test Results

Metric	Vanilla	Fairness-aware
Queries with representative results	**33%**	**95%**
Average MinSkew@100	-0.259	-0.011 (p-value < 1e-16)
InMails Sent	-	± 1% (p-value > 0.5)
InMails Accepted	-	± 1% (p-value > 0.5)

4.3 Lessons Learned in Practice

Post-Processing vs. Alternate Approaches: Broadly, there are three technical approaches for mitigating algorithmic bias in machine learning systems:

- *Pre-processing* includes the efforts prior to model training such as representative training data collection and modifying features or labels in the training data (e.g. [12]).
- *Modifying the training process* to generate a bias-free model (e.g., [5]).
- *Post-processing* includes the modification of the results of a trained machine learning model, using techniques such as calibration of regression or classification output and re-ranking of results (e.g., [42]).

We decided to focus on post-processing algorithms due to the following practical considerations which we learned over the course

of our investigations. First, applying such a methodology is agnostic to the specifics of each model and therefore scalable across different model choices for the same application and also across other similar applications. Second, in many practical internet applications, domain-specific business logic is typically applied prior to displaying the results from the ML model to the end user (e.g., prune candidates working at the same company as the recruiter), and hence it is more effective to incorporate bias mitigation as the very last step of the pipeline. Third, this approach is easier to incorporate as part of existing systems, as compared to modifying the training algorithm or the features, since we can build a standalone service or component for post-processing without significant modifications to the existing components. In fact, our experience in practice suggests that post-processing is easier than eliminating bias from training data or during model training (especially due to redundant encoding of protected attributes and the likelihood of both the model choices and features evolving over time). However, we remark that efforts to eliminate/reduce bias from training data or during model training can still be explored, and can be thought of as complementary to our approach, which functions as a "fail-safe".

Socio-technical Dimensions of Bias and Fairness: Although our fairness-aware ranking algorithms are agnostic to how the desired distribution for the protected attribute(s) is chosen and treat this distribution as an input, the choice of the desired bias / fairness notions (and hence the above distribution) needs to be guided by ethical, social, and legal dimensions. As discussed in §3.3, our framework can be used to achieve different fairness notions depending on the choice of the desired distribution. Guided by LinkedIn's goal of creating economic opportunity for every member of the global workforce and by a keen interest from LinkedIn's customers in making sure that they are able to source diverse talent, we adopted a "diversity by design" approach for LinkedIn Talent Search, and took the position that the top search results for each query should be representative of the broader qualified candidate set [22]. The representative ranking requirement is not only simple to explain (as compared to, say, approaches based on statistical significance testing (e.g., [42])), but also has the benefit of providing consistent experience for a recruiter or a hiring manager, who could learn about the gender diversity of a certain talent pool (e.g., sales associates in Anchorage, Alaska) and then see the same distribution in the top search results for the corresponding search query. Our experience also suggests that building consensus and achieving collaboration across key stakeholders (such as product, legal, PR, engineering, and AI/ML teams) is a prerequisite for successful adoption of fairness-aware approaches in practice [8].

5 RELATED WORK

There has been an extensive study of algorithmic bias and discrimination across disciplines such as law, policy, and computer science (e.g., see [20, 23, 44] and the references therein). Many recent studies have investigated two different notions of fairness: (1) *individual fairness*, which requires that similar people be treated similarly [17], and (2) *group fairness*, which requires that the disadvantaged group be treated similarly to the advantaged group or the entire population [34, 35]. While some studies focus on identifying and quantifying the extent of discrimination (e.g., [3, 11, 34]), others study mitigation approaches in the form of fairness-aware algorithms (e.g., [10, 14–19, 24–27, 30, 32, 39, 41–43]) and inherent trade-offs and limitations in achieving different notions of fairness and non-discrimination [16–18, 32]. Formal definitions of group fairness include demographic parity [17] and equal opportunity [26]. As discussed in §3.3, our framework supports these two definitions through appropriate choice of the desired distribution. We remark that there is extensive work in social science, philosophy, and legal literature on discrimination and fairness. We defer the reader to [6] for a discussion from a legal perspective and [4] for a discussion of four different frameworks of equal opportunity.

Our work is closely related to recent literature on fairness in ranking [5, 7, 13, 15, 37, 40, 42]. A method to assist the algorithm designer to generate a fair linear ranking model has been proposed in [5]. With respect to a fixed set of items, given a weight vector for ranking, the method in [5] computes a similar vector that meets fairness requirements. This approach is not applicable in our setting since it assumes that the candidate set of items to be ranked is fixed, whereas this set depends on the query in our case. Further, since it is limited to linear models and requires modifying the weight vector, this approach would be hard to scale across different model choices in practice (see §4.3). The problem of achieving individual equity-of-attention fairness in rankings, along with a mechanism for achieving amortized individual fairness, has been proposed in [7]. While this work aims to achieve individual fairness amortized across many rankings, our focus is on ensuring that each ranking meets the group fairness requirements specified using a desired distribution. Our proposed algorithms are designed to mitigate biases in the ranked results for each query by achieving a desired distribution over a protected attribute. Algorithms for ranking in the presence of fairness constraints, specified as the maximum (or minimum) number of elements of each class that can appear at any position in the ranking, have been proposed in [15, 42]. Zehlike et al. [42] propose a fair top-k ranking algorithm focusing on a required representation proportion for a *single* under-represented group. Our proposed algorithms allow attributes with many possible values as opposed to just binary attributes, hence constituting a more general framework, and can handle representation constraints corresponding to an arbitrary, desired categorical distribution. FA*IR algorithm proposed in [42] can be thought of as similar to our DetGreedy method that works only for a binary protected attribute, with a considerable difference in the minimum representation requirement computation. Celis et al. [15] present a theoretical investigation of fair ranking computation with constraints on the maximum and the minimum number of elements possessing an attribute value that can be present at each rank. In contrast, we provide relatively easy-to-explain and easy-to-implement algorithms since our work is motivated by the desire to implement and deploy in practice (see §4.2 and §4.3). Further, by presenting an empirical evaluation of the trade-off between fairness and business metrics, we enable practitioners to select a suitable algorithm for their application needs. A framework for formulating fairness constraints on rankings, and an associated probabilistic algorithm for computing utility maximizing fair ranking have been proposed in [37]. This method requires solving a linear program with a large number of variables and constraints (N^2 where N denotes the number of candidates to be ranked), and hence does not seem feasible in a practical search/recommendation system with strict latency requirements. Fairness measures for ranking have been proposed in [40], which we have extended for our setting (§2). Finally, ours is the first large-scale deployed framework for ensuring fairness in ranked results.

6 CONCLUSION

Motivated by the desire for creating fair opportunity for all users being ranked as part of search and recommendation systems and the consequent need for measuring and mitigating algorithmic bias in the underlying ranking mechanisms, we proposed a framework for fair re-ranking of results based on desired proportions over one or more protected attributes. We proposed several measures to quantify bias with respect to protected attributes such as gender and age, and presented fairness-aware ranking algorithms. We demonstrated the efficacy of these algorithms in reducing bias without affecting utility, and compared their performance via extensive simulations. We also deployed the proposed framework for achieving gender-representative ranking in LinkedIn Talent Search, where our approach resulted in huge improvement in the fairness metrics (nearly 3X increase in the number of queries with representative results) without impacting the business metrics. In addition to being the first web-scale deployed framework for ensuring fairness in the hiring domain, our work contains insights and lessons learned in practice that could be of interest for researchers and practitioners working on various web-scale search and recommendation systems.

A potential direction for future work is to further study fairness and utility guarantees of the proposed algorithms, as well as to extend with other algorithmic variants. While we have experimented with synthetic datasets to capture a wide range of parameter choices compared to what a real-world dataset might include, experiments with real datasets could be performed to complement our empirical study. Our approach assumes that the protected attribute values of candidates as well as the desired proportions are provided as part of the input; these assumptions may not hold in certain application settings. A fruitful direction is to understand the social dimension of how to reliably obtain the protected attribute values of users, if not readily available, and how to design the fairness-aware ranking algorithms in an incentive-compatible fashion in case these values are self-reported. A related direction is to study the social question of how the desired proportions (or fairness notions) should be chosen for different classes of practical applications.

ACKNOWLEDGMENTS

The authors would like to thank other members of LinkedIn Careers and Talent Solutions teams for their collaboration while deploying our system in production, and in particular Patrick Driscoll, Gurwinder Gulati, Rachel Kumar, Divyakumar Menghani, Chenhui Zhai, and Yani Zhang for working closely with us during the development of the product. We would also like to thank Deepak Agarwal, Erik Buchanan, Patrick Cheung, Gil Cottle, Nadia Fawaz, Joshua Hartman, Heloise Logan, Lei Ni, Ram Swaminathan, Ketan Thakkar, Janardhanan Vembunarayanan, Hinkmond Wong, Lin Yang, and Liang Zhang for insightful feedback and discussions.

REFERENCES

[1] U.S. equal employment opportunity commission, 2017. https://www1.eeoc.gov/eeoc/newsroom/release/1-3-17.cfm.
[2] European commission diversity charters, 2018. https://ec.europa.eu/info/strategy/justice-and-fundamental-rights/discrimination/tackling-discrimination/diversity-management/diversity-charters_en.
[3] J. Angwin, J. Larson, S. Mattu, and L. Kirchner. Machine bias. *ProPublica*, 2016.
[4] R. Arneson. Four conceptions of equal opportunity. *The Economic Journal*, 2018.
[5] A. Asudeh, H. V. Jagadish, J. Stoyanovich, and G. Das. Designing fair ranking schemes. In *SIGMOD*, 2019.
[6] S. Barocas and A. D. Selbst. Big data's disparate impact. *California Law Review*, 104, 2016.
[7] A. J. Biega, K. P. Gummadi, and G. Weikum. Equity of attention: Amortizing individual fairness in rankings. In *SIGIR*, 2018.
[8] S. Bird, B. Hutchinson, K. Kenthapadi, E. Kiciman, and M. Mitchell. Tutorial: Fairness-aware machine learning: Practical challenges and lessons learned. In *WWW*, 2019. https://sites.google.com/view/fairness-tutorial.
[9] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *NIPS*, 2016.
[10] T. Calders and S. Verwer. Three naive bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery*, 21(2), 2010.
[11] A. Caliskan, J. J. Bryson, and A. Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334), 2017.
[12] F. Calmon, D. Wei, B. Vinzamuri, K. N. Ramamurthy, and K. R. Varshney. Optimized pre-processing for discrimination prevention. In *NIPS*, 2017.
[13] C. Castillo. Fairness and transparency in ranking. *ACM SIGIR Forum*, 52(2), 2018.
[14] L. E. Celis, A. Deshpande, T. Kathuria, and N. K. Vishnoi. How to be fair and diverse? In *FATML*, 2016.
[15] L. E. Celis, D. Straszak, and N. K. Vishnoi. Ranking with fairness constraints. In *ICALP*, 2018.
[16] S. Corbett-Davies, E. Pierson, A. Feller, S. Goel, and A. Huq. Algorithmic decision making and the cost of fairness. In *KDD*, 2017.
[17] C. Dwork, M. Hardt, T. Pitassi, and R. Z. Omer Reingold. Fairness through awareness. In *ITCS*, 2012.
[18] S. A. Friedler, C. Scheidegger, and S. Venkatasubramanian. On the (im) possibility of fairness. *arXiv:1609.07236*, 2016.
[19] S. A. Friedler, C. Scheidegger, S. Venkatasubramanian, S. Choudhary, E. P. Hamilton, and D. Roth. A comparative study of fairness-enhancing interventions in machine learning. In *FAT**, 2019.
[20] B. Friedman and H. Nissenbaum. Bias in computer systems. *ACM Transactions on Information Systems (TOIS)*, 14(3), 1996.
[21] S. C. Geyik, Q. Guo, B. Hu, C. Ozcaglar, K. Thakkar, X. Wu, and K. Kenthapadi. Talent search and recommendation systems at LinkedIn: Practical challenges and lessons learned. In *SIGIR*, 2018.
[22] S. C. Geyik and K. Kenthapadi. Building representative talent search at LinkedIn, 2018. https://engineering.linkedin.com/blog/2018/10/building-representative-talent-search-at-linkedin.
[23] S. Hajian, F. Bonchi, and C. Castillo. Algorithmic bias: From discrimination discovery to fairness-aware data mining. In *KDD Tutorial on Algorithmic Bias*, 2016.
[24] S. Hajian and J. Domingo-Ferrer. A methodology for direct and indirect discrimination prevention in data mining. *IEEE TKDE*, 25(7), 2013.
[25] S. Hajian, J. Domingo-Ferrer, and O. Farràs. Generalization-based privacy preservation and discrimination prevention in data publishing and mining. *Data Mining and Knowledge Discovery*, 28(5-6), 2014.
[26] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. In *NIPS*, 2016.
[27] S. Jabbari, M. Joseph, M. Kearns, J. Morgenstern, and A. Roth. Fairness in reinforcement learning. In *ICML*, 2017.
[28] K. Jarvelin and J. Kekalainen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. on Information Systems (TOIS)*, 2002.
[29] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*, 2005.
[30] F. Kamiran, T. Calders, and M. Pechenizkiy. Discrimination aware decision tree learning. In *ICDM*, 2010.
[31] M. Kay, C. Matuszek, and S. A. Munson. Unequal representation and gender stereotypes in image search results for occupations. In *CHI*, 2015.
[32] J. Kleinberg, S. Mullainathan, and M. Raghavan. Inherent trade-offs in the fair determination of risk scores. In *ITCS*, 2017.
[33] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 1951.
[34] D. Pedreschi, S. Ruggieri, and F. Turini. Discrimination-aware data mining. In *KDD*, 2008.
[35] D. Pedreschi, S. Ruggieri, and F. Turini. Measuring discrimination in socially-sensitive decision records. In *SDM*, 2009.
[36] S. Sankar and A. Makhani. Did you mean "Galene"?, 2014. https://engineering.linkedin.com/search/did-you-mean-galene.
[37] A. Singh and T. Joachims. Fairness of exposure in rankings. In *KDD*, 2018.
[38] T. Verge. Gendering representation in Spain: Opportunities and limits of gender quotas. *Journal of Women, Politics & Policy*, 31(2), 2010.
[39] B. Woodworth, S. Gunasekar, M. I. Ohannessian, and N. Srebro. Learning non-discriminatory predictors. In *COLT*, 2017.
[40] K. Yang and J. Stoyanovich. Measuring fairness in ranked outputs. In *SSDBM*, 2017.
[41] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *WWW*, 2017.
[42] M. Zehlike, F. Bonchi, C. Castillo, S. Hajian, M. Megahed, and R. Baeza-Yates. FA*IR: A fair top-k ranking algorithm. In *CIKM*, 2017.
[43] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork. Learning fair representations. In *ICML*, 2013.
[44] I. Žliobaitė. Measuring discrimination in algorithmic decision making. *Data Mining and Knowledge Discovery*, 31(4), 2017.

A APPENDIX

A.1 Simulation Setting for §4.1

Our simulation framework can be summarized as follows:

(1) For each possible number of attribute values ($2 \leq |A| \leq 10$):
  (a) Generate a set P of 100K random categorical probability distributions of size $|A|$ each. Each probability distribution $P_j \in P$ is generated by choosing $|A|$ i.i.d. samples from the uniform distribution over (0, 1) and normalizing the sum to equal 1. Each P_j represents a possible desired distribution over the set A of attribute values.
  (b) For each $P_j \in P$:
    (i) For each attribute value in A, generate 100 random candidates whose scores are chosen i.i.d. from the uniform distribution over (0, 1), and order them by decreasing scores. We replicate this step 10 times (resulting in 1M distinct ranking tasks for each choice of $|A|$).
    (ii) Run each proposed fairness-aware ranking algorithm to get a fairness-aware re-ranked list of size 100, with the desired distribution P_j and the generated random candidate lists for each attribute value as inputs.

For each ranking task generated within the above framework, we compute the proposed bias measures such as InfeasibleIndex (Eq. 8), MinSkew (Eq. 2), and NDKL (Eq. 4), as well as Normalized Discounted Cumulative Gain[2] (NDCG) [28] as a measure of the "ranking utility" where we treat the scores of candidates as their relevance. We report the results in terms of the average computed over all ranking tasks for a given choice of the number of attribute values.

A.2 Results for InfeasibleCount and MaxSkew Measures

For the continuation of §4.1, we present the results for InfeasibleCount (Eq. 9) and MaxSkew@100 (Eq. 3) in Figures 6 and 7.

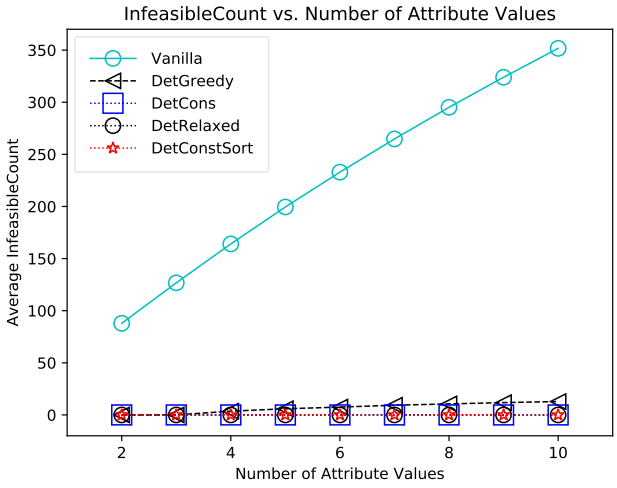

Figure 6: InfeasibleCount Measure Results

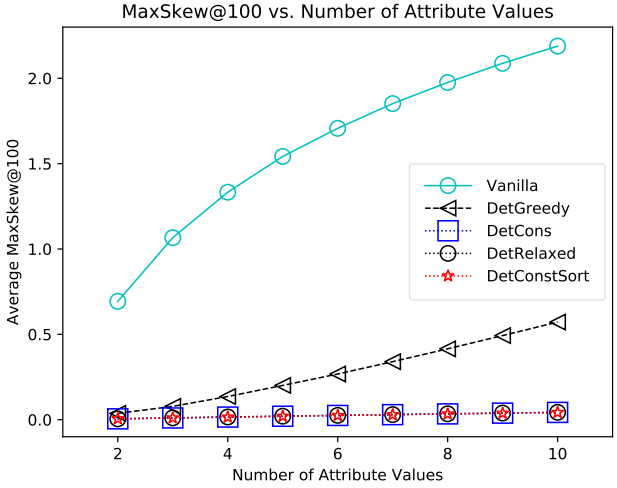

Figure 7: MaxSkew@100 Measure Results

A.3 Proof of Theorem 3.2

The algorithms DetGreedy, DetCons, and DetRelaxed ***are feasible*** *for cases when number of possible attribute values for the protected attribute is smaller than 4, i.e. for $|A| \leq 3$. DetGreedy is* ***not feasible*** *for $|A| \geq 4$.*

Proof. First, we will prove that all three algorithms are feasible for $|A| \leq 3$. For a ranking list to become infeasible, at a specific k^{th} index, there should be at least two attribute values, a_i and a_j, such that both have less than their minimum requirement shown so far excluding k^{th} index (hence both of them require a candidate from them to be shown in the index, and since this is impossible, then at least one of them will be below their minimum requirement, making the overall ranking infeasible), i.e. $count_{k-1}(a_i) < \lfloor p_{a_i} \cdot k \rfloor$, and $count_{k-1}(a_j) < \lfloor p_{a_j} \cdot k \rfloor$. For $|A| >= 2$, without loss of generality, we can claim that $1 > p_{a_i} > 0 \; \forall \; a_i$. Hence, the first ever index of a ranked list is always feasible, since for any a_i, the minimum requirement is $\lfloor p_{a_i} \cdot 1 \rfloor = 0$. Assume an index k where the ranking is still feasible, can there be an index $k+1$ such that upto k the ranking was feasible, and at $k+1$ it became infeasible? Suppose there are two attribute values a_1 and a_2 (we can use indices 1 and 2 without loss of generality) which met their minimum requirements at k^{th} index, but not $(k+1)^{th}$. Then:

$$\lceil p_{a_1} \cdot k \rceil = \lfloor p_{a_1} \cdot (k+1) \rfloor > count_k(a_1) \geq \lfloor p_{a_1} * k \rfloor \; and, \quad (12)$$

$$\lceil p_{a_2} \cdot k \rceil = \lfloor p_{a_2} \cdot (k+1) \rfloor > count_k(a_2) \geq \lfloor p_{a_2} * k \rfloor. \quad (13)$$

Above, $\lceil p_{a_1} \cdot k \rceil = \lfloor p_{a_1} \cdot (k+1) \rfloor$ (similarly for a_2), because the minimum requirement for any a_i, i.e. $\lfloor p_{a_i} \cdot k \rfloor$, increases by at most 1 at each index, due to the fact that $\forall \; a_i \; , \; p_{a_i} < 1$.

If $|A| = 2$, then this means that at the time of the k^{th} index, we have $\lfloor p_{a_1} \cdot k \rfloor + \lfloor p_{a_2} \cdot k \rfloor$ elements in the k length ranked list. Since we know that $p_{a_1} + p_{a_2} = 1$ (due to $|A| = 2$, hence $A = \{a_1, a_2\}$), then we immediately have the fact that $\lfloor p_{a_1} \cdot k \rfloor + \lfloor p_{a_2} \cdot k \rfloor < k$ unless both $p_{a_1} \cdot k$ and $p_{a_2} \cdot k$ are exact integers. But, if $p_{a_1} \cdot k$ and $p_{a_2} \cdot k$ are exact integers, then we have $\lceil p_{a_1} \cdot k \rceil = \lfloor p_{a_1} \cdot k \rfloor$ and

[2] NDCG is defined over a ranked list of candidates τ_r as follows:

$$\text{NDCG}(\tau_r) = \frac{1}{Z} \times \sum_{i=1}^{|\tau_r|} \frac{u(\tau_r[i])}{\log(i+1)},$$

where $u(\tau_r[i])$ is the relevance for the candidate in i^{th} position of τ_r. In our simulations, we treat the score of each candidate as the relevance, whereas in real-world applications, relevance could be obtained based on human judgment labels or user response (e.g., whether or the extent to which the user liked the candidate). Z is the normalizing factor corresponding to the discounted cumulative gain for the best possible ranking $\tau_r{}^*$ of candidates, i.e., $Z = \sum_{i=1}^{|\tau_r{}^*|} \frac{u(\tau_r{}^*[i])}{\log(i+1)}$.

Table 4: Infeasibility Example for DetGreedy with $|A| = 4$

a_i, p_{a_i} candidates(a_i)	a_1 (0.4) [0.1]	a_2 (0.4) [0.2]	a_3 (0.1) [0.3]	a_4 (0.1) [0.4]	**DetGreedy Chooses**
	Min. Requirements				
Index 1	0	0	0	0	a_4
Index 2	0	0	0	0	a_3
Index 3	1	1	0	0	**Infeasible**

$\lceil p_{a_1} \cdot k \rceil = \lfloor p_{a_1} \cdot k \rfloor$ which violate the condition 12 and 13 above. Hence, we prove by contradiction that such a case can never occur for $|A| = 2$.

If $|A| = 3$ (i.e. $A = \{a_1, a_2, a_3\}$), then this means that at the time of the k^{th} index, we have $\lfloor p_{a_1} \cdot k \rfloor + \lfloor p_{a_2} \cdot k \rfloor + \lceil p_{a_3} \cdot k \rceil$ elements in the k length ranked list, since $\lfloor p_{a_1} \cdot k \rfloor + \lfloor p_{a_2} \cdot k \rfloor + \lfloor p_{a_3} \cdot k \rfloor < k$, we don't go over $\lceil p_{a_i} \cdot k \rceil$ for any a_i per all three algorithms due their logic and because $\sum \lceil p_{a_i} \cdot k \rceil \geq \sum p_{a_i} \cdot k = k$. We also know that $\lfloor p_{a_1} \cdot k \rfloor + \lfloor p_{a_2} \cdot k \rfloor + \lfloor p_{a_3} \cdot k \rfloor \neq k$ since this could only happen if all $p_{a_i} \cdot k$ were exact integers and in that case a_1 and a_2 cannot reach a new minimum at index $k+1$ since both $p_{a_1} < 1$ and $p_{a_2} < 1$. Therefore:

$$(p_{a_1} \cdot k - \lfloor p_{a_1} \cdot k \rfloor) + (p_{a_2} \cdot k - \lfloor p_{a_2} \cdot k \rfloor) = \lceil p_{a_3} \cdot k \rceil - p_{a_3} \cdot k < 1 \text{ , and ,}$$

$$p_{a_1} + (p_{a_1} \cdot k - \lfloor p_{a_1} \cdot k \rfloor) + p_{a_2} + (p_{a_2} \cdot k - \lfloor p_{a_2} \cdot k \rfloor) < 1 + (p_{a_1} + p_{a_2}).$$

However, we know that $p_{a_1} + (p_{a_1} \cdot k) = p_{a_1} \cdot (k+1) \geq 1 + \lfloor p_{a_1} \cdot k \rfloor$ (per conditions in 12 and 13). Hence,

$$2 \leq p_{a_1} + (p_{a_1} \cdot k - \lfloor p_{a_1} \cdot k \rfloor) + p_{a_2} + (p_{a_2} \cdot k - \lfloor p_{a_2} \cdot k \rfloor) < 1 + (p_{a_1} + p_{a_2}) \text{ , then,}$$

$$1 < p_{a_1} + p_{a_2} \text{ ,}$$

which is a contradiction, since we know that $\sum p_{a_i} = 1$. This proves that there can be no case for $|A| = 3$ where the ranking is feasible at index k, and becomes infeasible at index $k+1$.

We will prove that the DetGreedy algorithm is not feasible for $|A| \geq 4$ via a simple counter-example. In Table 4, we list four attribute values with their corresponding desired proportions, and one candidate from each of them (we only give the scores in the brackets). Per the algorithm, at index 3, both a_1 and a_2 require a candidate from them to be shown, which makes the ranked list infeasible. Since any A where $|A| > 4$ covers the set of A with $|A| = 4$ (when $p_{a_i} = 0$ for $i > 4$), then we have proven that DetGreedy algorithm is not feasible for $|A| \geq 4$. □

A.4 Proof of Theorem 3.4

DetConstSort algorithm is feasible per Definition 3.1.

Proof. Since $\sum p_{a_i} = 1$, this means that by index k, we would have at most k candidates that came from attribute values that met their feasibility requirements per Eq. 6 and Definition 3.1, since:

$$\sum \lfloor p_{a_i} \cdot k \rfloor \leq \sum (p_{a_i} \cdot k) = k \text{ .}$$

This proves that each time a new candidate has to be inserted (i.e. since the attribute value a_i that it belongs to has reached its minimum representation condition for feasibility), there is enough empty space in the ranked list before and including k^{th} index. The algorithm also never swaps a candidate to cause its attribute value a_i to violate the feasibility condition 6. Since the feasibility constraint is not violated at any step of the algorithm, then we have proven that DetConstSort is indeed feasible. □

A.5 Description of Representative Ranking System Architecture

Figure 5 details our two-tiered ranking architecture for achieving gender-representative ranking for LinkedIn Talent Search systems. There are three primary components via which we attempt to achieve representative ranking:

(1) Computation of the gender distribution on the qualified set of candidates, along-side our first-level ranking.
(2) Re-ranking of the candidate list utilizing our first-level machine-learned ranking model's scores and the desired gender distributions. The top-k′ candidates, ranked in a representative manner are then sent to the second-level ranking.
(3) Re-ranking of the candidate list utilizing our second-level machine learned ranking model's scores and the desired gender distributions. The top-k′′ candidates, ranked in a representative manner are then presented to the recruiter.

In the figure, each *Searcher* has access to a subset of LinkedIn members, hence the search and retrieval is applied simultaneously on these sub-partitions. We also count the number of members falling into each inferred gender within each searcher (that pass the search criteria as pure filtering conditions), and then combine these counts to get the overall distribution of members to gender values. Since the *Recruiter* is presented the candidates in a page-by-page manner, always a top subset (k → k′ → k′′) of the candidates (after ranking, and representative re-ranking) are sent to the next stage (first-level → second-level → recruiter).