# Optimal Fair Ranking: Real Performances and Critical Parameters

Daniela A. Parletta[1,*], Fabio Napoli[1]

[1]Akkodis, Technology Innovation Hub, Artificial Intelligence Unit

**Abstract**

Ranking is the problem of retrieving the most relevant $m$ items for a query in a pool of $n$ items. Since ranking algorithms are extensively used in socially sensitive applications (e.g., candidate selections and search engines), it becomes critical to take into account an adequate notion of fairness. In this paper, we consider a flexible optimization-based fair ranking framework and we analyze optimal state-of-the-art algorithms on real-world data. We identify the merits as well as bottlenecks of existing methods and point out directions for future research.

**Keywords**

Ranking, Fairness, Optimization

## 1. Introduction

In this paper, we are concerned with the following fundamental problem. We are given a pool of $n$ *items*, and upon receiving a query, we have to retrieve the best $m \leq n$ of them. This problem has numerous applications in Search Engines (e.g., Google), Recommendation Systems (YouTube and Netflix), Targeted advertisement (Amazon and eBay), Social Networks (Facebook, Instagram, or Twitter), Matching Systems (e.g., Tinder), and so on. Given a query associated with each item, there is a notion of relevance or *utility* that depends on that query. For example, if the query is to present the top 10 movies on Netflix based on Bob's tastes, then each movie's utility will be an aggregation (based on Bob's preferences) of the genre components of that movie. In this work, we assume that a query is given and all the utilities are already determined.

Ranking is also employed in settings where social/demographic/sexual discrimination may happen. This is the case of HR departments performing recruiting campaigns and ranking the best candidates for the open positions. A second example is ranking the best students to be admitted to the college. In all these settings, there may be bias toward some groups of individuals (e.g., Black-African people or female individuals). This bias may be unwillingly injected into an automated ranking procedure, for example, relaying on sensitive attributes when computing the utility of an item. The approach taken in this work to enforce *fairness* is to directly account for it in the formalization of ranking problems that lead to a notion of **fairness by design**.

In this paper, we focus on the fair ranking framework proposed in [1], and provide a critical evaluation of state-of-the-art methods. Our main goal is to evaluate the maturity of existing

*AIMMES '24: Workshop on AI bias: Measurements, Mitigation, Explanation Strategies, March 20, 2024, Amsterdam, NL*
*Corresponding author.
daniela-angela.parletta@akkodis.com (D. A. Parletta); fabio.napoli@akkodis.com (F. Napoli)

methods for concrete applications. As a byproduct of our evaluation, we also point toward future research directions. In particular, we provide the following contributions:

- An explicit description of two state-of-the-art methods appeared in past literature only implicitly and in the proofs of runtime bounds.
- A discussion on the benefits of these methods as well as their drawbacks.
- A numerical evaluation of real-world datasets aimed at assessing the maturity of these solutions for concrete deployment into real-world applications. To the best of our knowledge, this is the first numerical evaluation of these methods.

**Related work.** Ranking with fairness constraints has recently received much attention from the research community, and a complete account is given in a recent survey [2, 3]. One can distinguish at least two different approaches to enforce fairness constraints in a ranking: *post-hoc adjustments* (e.g., [4, 5, 6, 7]) and *fairness by design* (e.g., [1, 8, 9]). The former, aim takes as input a (possibly unfair) ranking, produced by an algorithm, and applies the minimum amount of modifications to satisfy the fairness constraints. The latter, instead, are methods designed to provide output rankings that already satisfy the fairness constraints. We notice that while post-processing adjustments approaches can be applied to any ranking algorithm, even one that is already developed and deployed, to enforce fairness. On the other hand, fairness by design typically allows one to obtain rankings of superior quality.

In [1] authors propose an optimization framework for fair ranking problems and develop several polynomial time algorithms for it. This model has been generalized in [8] to account for noisy sensitive attributes. In [9] authors further extend the framework to aggregate multiple rankings optimally. We notice that the specific problem formulation described in this paper may be in principled approaches with the algorithms proposed in [10, 11]. However, the method proposed in [10] features an exponential dependence on the number of constraints, which in the case of fair ranking may be very large. On the other hand, the algorithms in [11] are only guaranteed to satisfy an $\sqrt{m}$-additive perturbation of the constraints, which is unsatisfactory for even moderately large $m$.

**Structure of the paper.** The remaining of this paper is structured as follows. Section 2 introduces the problem setting. Section 3 details the considered methods and Section 4 present the numerical evaluation. Finally, Section 5 draws some conclusions and sketches future directions for research.

## 2. Problem Setting

**Notation.** $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ denote the set of natural, integer, and real numbers. We define $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, $\mathbb{R}_+ := \mathbb{R} \cap [0, \infty)$ and $\mathbb{Z}_+ := \mathbb{Z} \cap \mathbb{N}_0$. For every $n \in \mathbb{N}$, $[n]$ will denote the set $\{1, 2, \dots, n\}$. Given a set $A$ we denote its cardinality by $|A|$. Given $m, n \in \mathbb{N}$ and a set $A$, $A^{m \times n}$ denotes the set of $m$ by $n$ matrices with entries in $A$. We will denote vectors with lowercase bold letters and matrices with uppercase bold letters. Given a vector $\mathbf{a}$ we denote its $i$-th entry with $a_i$. The zero norm of a vector $\mathbf{a}$ is defined as $\|\mathbf{a}\|_0 := |\{i : a_i \neq 0\}|$. Given a matrix $\mathbf{A}$ we denote its $(i, j)$-th entry with

$A_{ij}$. Given a logical predicate $p$, we denote its indicator function by $\mathbb{I}(p)$, notice that $\mathbb{I}(p) = 1$ when $p$ is true and 0 otherwise.

**Setting.** Given $m, n \in \mathbb{N}$, the ranking problem consists informally of retrieving (and sorting) the $m$ *items* in a pool of $n$ upon receiving a query. The *utility* of an item depends on the specific query and maybe the level of the required skill in a job offer, or the h-index in a Google Scholar search. In this work, we assume that the query is given and fixed, so that we collect the utility of placing a item $i \in [n]$ in position $j \in [m]$ in a **utility matrix** $\mathbf{U} \in R_+^{m \times n}$. We stress the importance of accounting for the position of a item in a given ranking: in a real-world setting, the perceived utility is inversely proportional to the item's position in the ranking, a phenomenon known as **position bias**. A ranking is a vector $\mathbf{r} \in [n]^m$ s.t. $r_j = k$ if item $k$ is placed in position $j$, and $r_i \neq r_j$ for each $i \neq j$. For $k \in [m]$ the $k$-**prefix** of a ranking $\mathbf{r}$ is the vector $\mathbf{r}_k = (r_1, \dots, r_k)$. We introduce the **assignment matrix** of a ranking $\mathbf{A} \in \{0,1\}^{m \times n}$ where i) $A_{ij} = 1$ if item $i$ is assigned to position $j$ in the ranking, ii) $\sum_{j=1}^{m} A_{ji} \leq 1$ for each $i \in [n]$ and $\sum_{i=1}^{n} A_{ji} = 1$ for each $j \in [m]$. We denote the set of all such matrices by $\mathscr{A}$. Notice that there is a bijection between the set of the rankings and the set of the assignment matrices, thus we can speak about a ranking or its assignment matrix interchangeably. The *optimal ranking problem* is defined as

$$\max_{\mathbf{A} \in \mathscr{A}} \sum_{j=1}^{m} \sum_{i=1}^{n} A_{ji} U_{ji} \ . \tag{1}$$

As the focus of this paper is on fair rankings, we consider a more constrained version of Equation (1). In particular, we assume that there is a set of $p \in \mathbb{N}$ properties w.r.t. achieve fairness and denote with $P_\ell \subseteq [n]$ the set of items having the property $\ell \in [p]$: examples of properties are the gender, the ethnicity or the age. Fairness may be expressed via lower and upper bounds on the number of items featuring a given properties in each prefix of the ranking. For example, we may require that in each even prefix of the ranking, there should be an equal number of men and women. Formally, the matrices $\mathbf{F}^{(l)}, \mathbf{F}^{(u)} \in \mathbb{Z}_+^{m \times p}$ specifies the minimum $\mathbf{F}^{(l)}_{k\ell}$ and the maximum number $\mathbf{F}^{(u)}_{k\ell}$ of items having the property $\ell$ in the $k$-prefix of a feasible ranking, that is

$$F^{(l)}_{k\ell} \leq \sum_{j=1}^{k} \sum_{i \in P_\ell} A_{ji} \leq F^{(u)}_{k\ell}, \quad \forall k \in [m], \forall l \in [p] \ . \tag{2}$$

These matrices are called **lower fairness constraint matrix** and **upper fairness constraint matrix** respectively. The **Optimal Fair Ranking** (OFR) problem is thus defined as

$$\begin{aligned} \max_{\mathbf{A} \in \mathscr{A}} \quad & \sum_{j=1}^{m} \sum_{i=1}^{n} A_{ji} U_{ji} \\ \text{s.t.} \quad & F^{(l)}_{k\ell} \leq \sum_{j=1}^{k} \sum_{i \in P_\ell} A_{ji} \leq F^{(u)}_{k\ell}, \quad \forall k \in [m], \forall l \in [p] \ . \end{aligned} \tag{3}$$

We notice that this problem has binary $m \cdot n$ decision variables and $m \cdot (p + n)$ constraints. The optimal value of the above problem will be denoted with OPT.

To set up a bottom line, notice that even just checking the feasibility of Equation (3) is NP-Hard in the general case (see Theorem 10.1, 10.3 and 10.4 in [1]). Moreover, in the general case Equation (3) is APX-Hard, and it thus not not admit a polynomial-time approximation scheme (see Theorem 10.2 in [1]).

**Utility and constraints models.** In what follows we list a few popular models for the positions bias. We denote with $u_i$ the *absolute utility* of the item $i$. We consider the following models for the utility $U_{ij}$ of placing $i \in [n]$ in position $j \in [m]$

$$\underbrace{u_i \log_2^{-1}(1+j)}_{logarithmic}, \quad \underbrace{u_i p(1-p)^{j-1}}_{geometric}, \quad \underbrace{u_i \mathbb{I}(j=1)}_{singular} \tag{4}$$

Notice that while this specific bias is uniform across the items in the pool, the general utility model presented above does not need to be. Further, notice that the geometric model is parametrized by a *discount factor* $p \in [0,1]$ controlling how quickly the utility of an item decays w.r.t. its position in the ranking.

We will assume the following conditions on the utility $\mathbf{U}$

$$U_{i_1 j_1} \geq U_{i_2 j_1}, \quad U_{i_1 j_1} \geq U_{i_1 j_2}, \quad U_{i_1 j_1} + U_{i_2 j_2} \geq U_{i_1 j_2} + U_{i_2 j_1} \tag{5}$$

where the two leftmost inequality are called monotonicity and the rightmost is known as **Monge condition**. We notice that is the users are sorted in descending order of absolute utilities, then any of the above models for the position bias satisfies the Equation (5).

As for the constraints, they are usually specified in terms of a maximum number of items belonging to a certain category that can appear in a given prefix. This can be done by setting $\mathbf{F}^{(l)}$ to the zero matrix and only specifying $\mathbf{F}^{(u)}$. We name this problem **Optimal Fair Ranking with Upper constraints** (OFRU).

## 3. Algorithms

In this section, we describe two algorithms that solve the OFR problem. Each method features optimality and runtime guarantees that depend on certain parameters, making them better suited under different circumstances. We notice that these methods first appeared in [1], but only implicitly with the proofs of the main theorems. We notice that this is the first explicit description of such algorithms.

**Dynamic programming.** We define the type of an item $i$ as a binary vector $\mathbf{t}(i) \in \{0,1\}^p$ such that $t_\ell(i) = 1$ if and only if $i$ has the property $\ell \in [p]$. Let $T := \{\mathbf{t}(i) : i \in [n]\}$ the set of different types in the data, and let $t := |T|$. We let the elements of $T$ be $v^{(1)}, \dots, v^{(t)}$ and define $T_\ell := \{i \in [m] : \mathbf{t}(i) = v^{(\ell)}\}$ - the set of the first $m$ items of type $\ell$ - for each $\ell \in [t]$. We denote with $t_\ell$ the cardinality of $T_\ell$ and its elements with $i_1^{(\ell)}, \dots, i_{t_\ell}^{(\ell)}$.

It is convenient to perform a pre-processing step that only retains the first (at most) $m$ items (i.e., the items with the smallest indices) for each type. In light of Equation (3), among these (at

**Algorithm 1** Dynamic programming for OFR

**Input**: the sets $T_1, \dots, T_t$, the utility matrix $\mathbf{U}$, the constraints $\mathbf{F}^{(l)}, \mathbf{F}^{(u)}$, and the set of properties $\{1, \dots, p\}$.

Initialize $U[0, \dots, 0] \leftarrow 0, \quad s_1, \dots, s_t \leftarrow 0, \quad r = \text{EMPTY LIST}$
**while** $s_1 + \cdots + s_t < m$ **do**
  $\mathbf{u} \leftarrow (0, \dots, 0) \in \mathbb{N}_0^t$
  **for** $\ell = 1, \dots, t$ **do**
    **if** $\text{FEASIBLE}(s_1, \dots, s_\ell + 1, \dots, s_t)$ **then**
      $u_\ell \leftarrow U[s_1, \dots, s_t] + U_{i_{s_\ell}^{(\ell)} j}$
    **else**
      $u_\ell \leftarrow -\infty$
    **end if**
  **end for**
  Get the index $\ell^*$ and the value $u^*$ of $\max_{\ell \in [t]} u_\ell$
  Increment $s_{\ell^*} \leftarrow s_{\ell^*} + 1, U[s_1, \dots, s_t] \leftarrow u^*, r.\text{APPEND}(i_{s_{\ell^*}}^{(\ell)})$
**end while**

**Output**: the optimal ranking $r$, the optimal utility $U[s_1, \dots, s_\ell]$.

most) $m \cdot t$ items there will be the optimal ranking. Notice step can be done in order of $m \cdot t$ time and produces the sets $T_1, \dots, T_t$.

The idea of the dynamic programming approach is the following. We can partition the set of all possible rankings of a certain length $k \leq m$ according to the number of items of each type they have. That is, for $s_1, \dots, s_t \in \mathbb{N}_0$ s.t. $k = s_1 + \cdots + s_t$ we denote with $(s_1, \dots, s_t)$ the set of all rankings of length $k$ made of $s_j$ items of type $\mathbf{v}^{(j)}$ for each $j \in [t]$. Furthermore, with $U[s_1, \dots, s_t]$ we denote the value of the feasible rankings of the highest value in $(s_1, \dots, s_t)$. Thus, finding the optimal solution to an OFR problem corresponds to identifying a feasible ranking of utility $U[s_1, \dots, s_t] = \text{OPT}$, when $s_1 + \cdots + s_t = m$. Checking feasibility of the rankings in $(s_1, \dots, s_t)$ is simple: it is enough to check that

$$F_{k\ell}^{(l)} \leq \sum_{h=1}^{t} s_h v_\ell^{(h)} \leq F_{k\ell}^{(u)}, \quad \forall \ell \in [p],$$

which takes order of $t \cdot p$ time.

We start initializing $U[0, \dots, 0] = 0$ and notice that by Equation (5), it either holds that

$$U[s_1, \dots, s_t] = \max_{\ell \in [t]} U[s_1, \dots, s_\ell - 1, \dots, s_t] + U_{i_{s_\ell - 1}^{(\ell)} j},$$

or that $(s_1, \dots, s_t)$ is infeasible in which case we set $U[s_1, \dots, s_t] = -\infty$. As a result, we can build the optimal solution with a bottom-up approach: we start from $(0, \dots, 0)$ and we build the ranking by adding an item of maximal utility that keeps the constraints satisfied at each step. As we have seen, checking the feasibility takes the order of $t \cdot p$ and there are at most order of $m^t$ rankings that needed to be considered (all the ways to chose $t$ natural numbers so that they

sum to $m$). The overall algorithm is described in Algorithm 1. Below we report the theorem establishing its theoretical performances.

**Theorem 1** (Theorem 3.1 in [1])**.** *Algorithm 1 finds an optimal solution to the OFR problem, if there is one, in order of $ptm^t$ time. Otherwise, it return $U[s_1, \dots, s_t] = -\infty$. The pre-processing runs in order of $m \cdot t$ time.*

**Remark 1** (On the complexity of Algorithm 1)**.** We make the following comments:

- We notice that an exhaustive search among all possible rankings of length $m$ in a pool of $m \cdot t$ takes the order of $t^m$ time. On the other hand, the proposed method takes the order of $m^t$ time, which is better: in real-world applications, $t$ should be thought of as a constant determined by the fairness constraints, while $m$ is a parameter that can be as large as $n$.
- The practicality of Algorithm 1 depends on the value of $t$. We notice that $t \geq p$, implies that when fairness is defined using more than 2-3 properties, the algorithm becomes unpractical.

**Greedy algorithm.** To overcome the limitations discussed in Remark 1, we consider an alternative greedy approach.

The algorithm is simple. Similarly to Algorithm 1, it keeps the items grouped into the set $T_1, \dots, T_t$. At each step $j$, the algorithm greedily takes the item with the highest utility - let $\mathbf{v}^{k(j)}$ its type - and adds it to the ranking if this addition does not violate the constraints. Otherwise, it seeks the next item among the types $\{\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(t)}\}/\{\mathbf{v}^{k(j)}\}$. If no feasible item can be found, the problem is declared to be infeasible. Since there are $m$ positions to fill and each item can be checked for feasibility in order of $p$ time, the runtime of this algorithm is the order of $m \cdot p$. The algorithm is also described in Algorithm 2.

Despite its simplicity, this algorithm provably finds an optimal solution in the following class of problems. Let define

$$\Delta \coloneqq \max_{v \in T} \|v\|_0 \tag{6}$$

that is, $\Delta$ is the maximum number of properties each type of item can exhibit. Then the following holds.

**Theorem 2** (Theorem 3.2 in [1])**.** *Algorithm 2 finds an optimal solution to any OFRU problem as long as $\Delta = 1$. For those problems, if the algorithm returns infeasible, then the problem does not admit a solution. It runs in order of $m \cdot p$ time. The pre-processing runs in order of $m \cdot t$ time.*

**Remark 2** (On the complexity of Algorithm 2)**.** We make the following comments:

- Theorem 2 guarantees optimality of Algorithm 2 only when the problem is an instance of the OFRU problem. This assumption is in contrast to Algorithm 1 that instead can solve the more general OFR problem. Nevertheless, OFRU models many natural settings, since fairness is usually expressed only by limiting the number of items in the unprotected classes (thus only specifying the matrix $\mathbf{F}^{(u)}$).
- The assumption $\Delta = 1$ is satisfied in many practical settings, including the important case of mutually exclusive properties (e.g. male vs. female, White vs Black vs Asian, protected workers categories). For the sake of comparison, notice that in this setting, $t = p$ and when $p$ is even moderately large (e.g. $t \in \{4, 5\}$) Algorithm 1 is unpractical.

**Algorithm 2** Greedy Algorithm for OFR

**Input**: the sets $T_1, \dots, T_t$, the utility matrix $\mathbf{U}$, the constraints $\mathbf{F}^{(l)}, \mathbf{F}^{(u)}$, and the set of properties $\{1, \dots, p\}$.

Initialize $U \leftarrow 0, \quad s_1, \dots, s_t \leftarrow 0, \quad r = \text{EMPTY LIST}$
**for** $j = 1, \dots, m$ **do**
  Sort by decreasing utility $i_{s_1}^{(1)}, \dots, i_{s_t}^{(t)}$ # We denote with $\pi$ the sorted indices
  **for** $\ell = 1, \dots, t$ **do**
    **if** $\text{FEASIBLE}(s_1, \dots, s_\ell + 1, \dots, s_t)$ **then**
      $u \leftarrow U + U_{\pi_{s_\ell}^{(\ell)} j}, \quad r.\text{APPEND}(\pi_{s_\ell}^{(\ell)}), \quad T_\ell \leftarrow T_\ell / \{\pi_{s_\ell}^{(\ell)}\}, \quad \text{FLAG} \leftarrow \text{TRUE}$
    **end if**
  **end for**
  **if** FLAG == FALSE **then return** INFEASIBLE
  **end if**
**end for**

**Output**: a fair ranking $r$ and its utility $U$.

- The runtime of this greedy method is exponentially (in $t$) smaller than that of Theorem 1.
- When $\Delta \geq 3$, Theorem 10.2 in [1] establish that it is NP-Hard to find a solution that is within a (multiplicative) factor larger than $\Delta / \log(\Delta)$.

**Practical considerations.** Both algorithms employ a pre-processing step that prunes the item pool to retain only the top $m \cdot t$ items. This step is justified by the Equation (5). In practice, even when the utilities are obtained from an application of the position-bias model to the absolute utilities, it is often not the case that the items are already sorted in a way that $U$ satisfies Equation (5). To overcome this limitation, it is necessary to perform a sorting of the items based on their absolute utilities, a step that requires order of $n \log n$ time. Overall the pre-processing runs in order of $n \cdot \log n + m \cdot t$ time. In the case of Algorithm 2 this pre-processing time dominates over the ranking time.

Furthermore, both methods require storage of the constraint matrices which occupy order of $m \cdot p$ space. In addition, Algorithm 1 also requires to store the array with the $U[s_1, \dots, s_t]$ values, which can require up to order of $m^t$ space. This results in a strong limitation - even disposing of time, when $t$ is moderately large it is not possible to run this method.

Finally, we notice that typically the fairness constraints are phrased in natural language. To work within this framework, it is necessary to translate this constraint into the matrices $\mathbf{F}^{(l)}$ and $\mathbf{F}^{(u)}$. This is not straightforward in general, but we will show an example in Section 4.

| Dataset | $n$ | $t$ | $\Delta$ |
|---|---|---|---|
| Chess Ranking | 3251 | 5 | 2 |
| Chess Ranking S | 3251 | 4 | 2 |
| Chess Ranking R | 3251 | 3 | 1 |
| Occupation G | 4494 | 2 | 1 |
| Occupation W | 4494 | 94 | 1 |

**Table 1**
Dataset parameters.

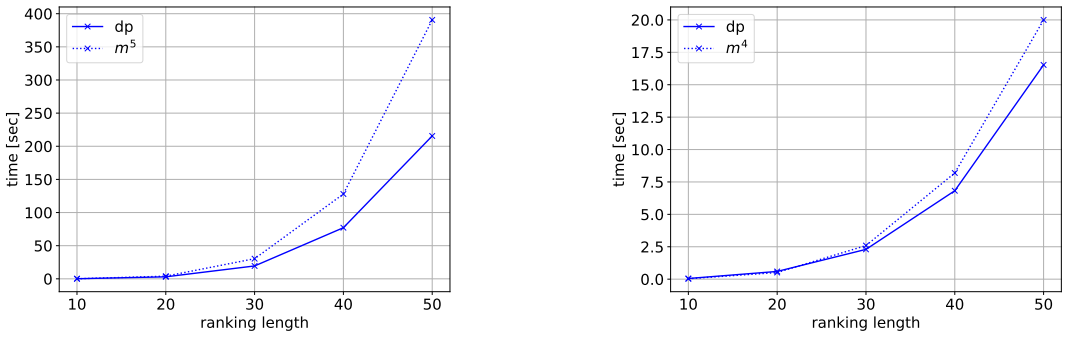
## 4. Numerical Experiments

In this section, we perform some experiments aiming at assessing the following questions:

- Exemplify some typical values that $t$ and $\Delta$ may take in real-world applications.
- What is the actual runtime of Algorithm 1? Does it follow the theoretical complexity?
- How does Algorithm 2 compare with Algorithm 1 even when the hypothesis of Theorem 2 are violated?

For all datasets, we consider the logarithmic position-bias model discussed in Section 2. For the sake of comparison, when setting the fairness constraints we always put $\mathbf{F}^{(l)} = \mathbf{0}$, thus restricting the problems to be instances of the OFRU problem. For $\mathbf{F}^{(u)}$ we consider the notion of fairness known as *proportional representation*, where the entry $(k, \ell)$ of the upper fairness constraint matrix is $k|I_p|/n$, where $I_p$ is the subset of items that have property $p$. Notice that this constraint imposes that each property, in the first $k$ positions, cannot be represented with more items than the overall population, fractionally. For each algorithm, we measure the runtime excluding the pre-processing that is common to both methods. For Algorithm 2 we report also the **competitive ratio** of the computed ranking $\hat{\mathbf{r}}$, this is defined as the ratio between the utility of $\hat{\mathbf{r}}$ and the optimal value OPT. This ratio takes value in $[0, 1]$ and follows the semantic that *the higher the better*. Notice that, due to its optimality, for Algorithm 1 this ratio always evaluates 1. In what follows, we refer to Algorithm 1 as DP and to Algorithm 2 as Greedy. All experiments are performed on an Intel i9 2.4 GHz with 8 cores and 16 GB 2.66 MHz DDR4 of RAM.

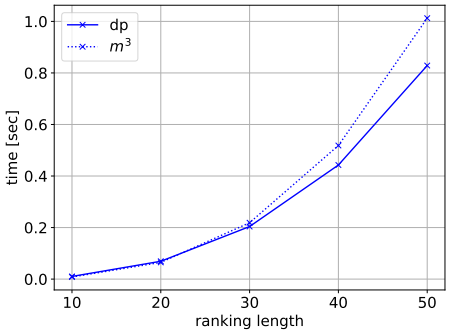
**Parameters.** We consider the following real-world datasets. **Chess Ranking** [12] consists of 3251 entries, one per player. For each player, we have the absolute utility as given by the FIDE rating, the gender (male or female), and the race (Asian, Hispanic-Latin, white). Notice that in this dataset there are no women of Hispanic-Latin ethnicity, so that $t = 5$ and $\Delta = 2$. We derive two more datasets from Chess Ranking. We consider a simplified version, that we name **Chess Ranking S**(implified), where each player has a gender and is either white or non-white, leading to $t = 4$ and $\Delta = 2$. Instead, we drop the gender attribute in **Chess Ranking R**(ace) and keep the ethnicity, leading to $t = 3$ and $\Delta = 1$. **Occupation** [8] consists of 4494 images of workers. These data are the results of Google queries each one reporting the top workers in a given professional category, among 94 options. The absolute utility is the position of a worker in the relative ranking. For each image, we also have the gender of the worker. We generate two datasets from Occupation. In **Occupation G**(ender) we drop the working category and



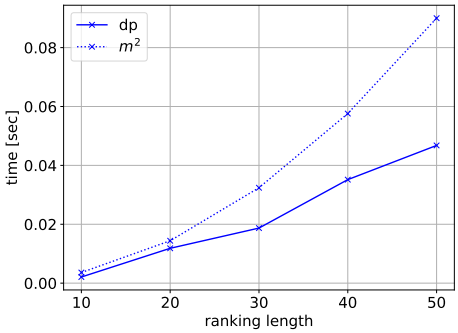

(a) Chess Ranking




(b) Chess Ranking S




(c) Chess Ranking R




(d) Occupation G

**Figure 1:** Runtime of Algorithm 1 across the considered datasets.

only consider the gender, leading to $t = 2$ types and $\Delta = 1$. In **Occupation W**(ork) we drop the gender and only the working category, leading to $t = 94$ and $\Delta = 1$. The relevant parameters of the dataset are reported in Table 1.

**Runtime.** To evaluate the runtime of Algorithm 1 we vary the ranking length $m$ in $\{10, 20, 30, 40, 50\}$ and measure it for each dataset (excluding Occupation work). For comparison, we also report the leading term $m^t$ in the worst-case bound of Theorem 1. This latter term is multiplied by a suitably chosen small constant, to make the quantities comparable. Results are shown in Figure 1. First, notice the qualitative differences among the plots: the actual runtime scales with a different law depending on the dataset. This is in line with the theory as the leading term in the bound takes on the expression $m^5, m^4, m^3, m^2$ respectively on the considered datasets. Second, we notice that on at least 3 out of 4 datasets, the behavior of DP closely follows the worst-case behavior predicted by the theory. In the case of Occupation G instead, it performs significantly better.

**Optimization.** We now fix $m = 100$ and compare DP to Greedy. Results are shown in Table 2. First, notice that Greedy is always *at least* 2 orders of magnitude faster than DP. In the case of

| Dataset | Algorithm | Runtime [sec] | Ratio [%] |
|---|---|---|---|
| Chess Ranking | DP | 6343.4 | 1 |
| | Greedy | 0.0287 | 0.9998 |
| Chess Ranking S | DP | 247.76 | 1 |
| | Greedy | 0.01593 | 0.99995 |
| Chess Ranking R | DP | 7.24895 | 1 |
| | Greedy | 0.01078 | 1 |
| Occupations G | DP | 0.14752 | 1 |
| | Greedy | 0.00627 | 1 |
| Occupation W | Greedy | 0.21901 | 1 |

**Table 2**
Experimental comparison between Algorithm 1 (DP) and Algorithm 2 (Greedy).

Chess Ranking, it is 6 orders of magnitude faster. Second, even when $\Delta > 1$, Greedy features a competitive ratio very close to 1, meaning that it essentially finds an optimal solution. Third, in the case of Occupation W Greedy finds an optimal solution in about 0.2 seconds, while it is not even possible to run DP due to the tremendous space required to store the table for $U[s_1, \dots, s_t]$: $100^{94}$ elements for this dataset.

## 5. Conclusion and Future Directions

Ranking is a foundational problem in algorithms, with numerous applications where fairness constraints must be accounted for. In this paper, we present and discuss an important optimization-based framework for ranking with fairness constraints. We provide, for the first time, an explicit description of state-of-the-art algorithms and evaluate them critically on real-world datasets. By considering the critical parameters controlling the complexity of the fair ranking problem, we shed light on the limitations of the exact dynamic programming method. While it appears to be feasible in some circumstances, its runtime explodes as soon as more than 3 properties are used to define fairness. On the other hand, the greedy approach seems to offer a viable alternative even when the hypothesis that ensures its optimality is violated. Both these methods appear to feature a level of maturity that may already suffice for certain real-world applications such as candidate selection in job matching.

Future directions in this area may attempt to: design optimal linear time algorithms for the case $\Delta = 2$; re-parameterize the problem and design novel algorithms with better dependencies on the parameters than Algorithm 1; identify additional assumptions to place to escape NP-hardness results.

## 6. Acknowledgments

This paper was supported by the European Union's Horizon Europe research and innovation program under grant number 101070363 - AEQUITAS. Funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect

those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

## References

[1] L. E. Celis, D. Straszak, N. K. Vishnoi, Ranking with fairness constraints, in: I. Chatzigiannakis, C. Kaklamanis, D. Marx, D. Sannella (Eds.), 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, volume 107 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 28:1–28:15. URL: https://doi.org/10.4230/LIPIcs.ICALP.2018.28. doi:10.4230/LIPICS.ICALP.2018.28.

[2] M. Zehlike, K. Yang, J. Stoyanovich, Fairness in ranking, part I: score-based ranking, ACM Comput. Surv. 55 (2023) 118:1–118:36. URL: https://doi.org/10.1145/3533379. doi:10.1145/3533379.

[3] M. Zehlike, K. Yang, J. Stoyanovich, Fairness in ranking, part II: learning-to-rank and recommender systems, ACM Comput. Surv. 55 (2023) 117:1–117:41. URL: https://doi.org/10.1145/3533380. doi:10.1145/3533380.

[4] M. Zehlike, F. Bonchi, C. Castillo, S. Hajian, M. Megahed, R. Baeza-Yates, Fa* ir: A fair top-k ranking algorithm, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 1569–1578.

[5] A. Singh, T. Joachims, Fairness of exposure in rankings, in: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, 2018, pp. 2219–2228.

[6] M. Zehlike, C. Castillo, Reducing disparate exposure in ranking: A learning to rank approach, in: Proceedings of the web conference 2020, 2020, pp. 2849–2855.

[7] M. Zehlike, P. Hacker, E. Wiedemann, Matching code and law: achieving algorithmic fairness with optimal transport, Data Mining and Knowledge Discovery 34 (2020) 163–200.

[8] L. E. Celis, V. Keswani, Implicit diversity in image summarization, Proceedings of the ACM on Human-Computer Interaction 4 (2020) 1–28.

[9] N. Boehmer, L. E. Celis, L. Huang, A. Mehrotra, N. K. Vishnoi, Subset selection based on multiple rankings in the presence of bias: Effectiveness of fairness constraints for multiwinner voting score functions, in: Proceedings of the 40th International Conference on Machine Learning, 2023, pp. 2641–2688.

[10] F. Grandoni, R. Ravi, M. Singh, R. Zenklusen, New approaches to multi-objective optimization, Mathematical Programming 146 (2014) 525–554.

[11] A. Srinivasan, Improved approximations of packing and covering problems, in: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, 1995, pp. 268–276.

[12] A. Ghosh, R. Dutt, C. Wilson, When fair ranking meets uncertain inference, in: Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval, 2021, pp. 1033–1043.