Data Science and Engineering
https://doi.org/10.1007/s41019-023-00213-y

RESEARCH PAPER

A Framework to Maximize Group Fairness for Workers on Online Labor Platforms

Anis El Rabaa[1] · Shady Elbassuoni[1] · Jihad Hanna[2] · Amer E. Mouawad[1] · Ayham Olleik[2] · Sihem Amer-Yahia[3]

Received: 2 February 2023 / Revised: 17 March 2023 / Accepted: 6 April 2023

Abstract
As the number of online labor platforms and the diversity of jobs on these platforms increase, ensuring group fairness for workers needs to be the focus of job-matching services. Risk of discrimination against workers occurs in two different job-matching services: when someone is looking for a job (i.e., a job seeker) and when someone wants to deploy jobs (i.e., a job provider). To maximize their chances of getting hired, job seekers submit their profiles on different platforms. Similarly, job providers publish their job offers on multiple platforms with the goal of reaching a wide and diverse workforce. In this paper, we propose a theoretical framework to maximize group fairness for workers 1) when job seekers are looking for jobs on multiple platforms, and 2) when jobs are being deployed by job providers on multiple platforms. We formulate each goal as different optimization problems with different constraints, prove most of them are computationally hard to solve and propose various efficient algorithms to solve all of them in reasonable time. We then design a series of experiments that rely on synthetic and semi-synthetic data generated from a real-world online labor platform to evaluate our framework.

Keywords Group fairness · Online labor platforms · Crowdsourcing · Optimization · Job seeker · Job provider

1 Introduction

Online labor marketplaces such as TaskRabbit[1] and Upwork[2] are gaining popularity as platforms to hire workers to perform certain jobs. On these platforms, people can hire temporary workers in the physical world (e.g., someone to clean an apartment in New York City), or remote workers (e.g., someone to design a website) by submitting a description of the job and receiving a ranked list of potential workers deemed qualified for the job. A job seeker (i.e., a worker looking for a job) provides her job interests and skills and is matched to certain jobs available on the platform. A job provider (i.e., an employer looking for workers to perform a certain job) provides a description of the job and is matched to potential workers. In the majority of these platforms, such job-matching services are algorithmic and most of the time opaque. This raises fairness concerns. A ranking of workers will be considered unfair if it is biased toward certain groups of people, such as white males. This commonly happens since ranking usually depends on the social feedback received by workers in the form of reviews and ratings, and on the number of their past jobs, both of which perpetuate bias against certain groups of workers [9, 16, 17, 29]. In this paper, we propose the first theoretical framework that can be used to assess and compare worker fairness of multiple jobs on multiple platforms.

✉ Shady Elbassuoni
se58@aub.edu.lb

Anis El Rabaa
ase29@mail.aub.edu

Jihad Hanna
jgh20@mail.aub.edu

Amer E. Mouawad
aa368@aub.edu.lb

Ayham Olleik
abo00@mail.aub.edu

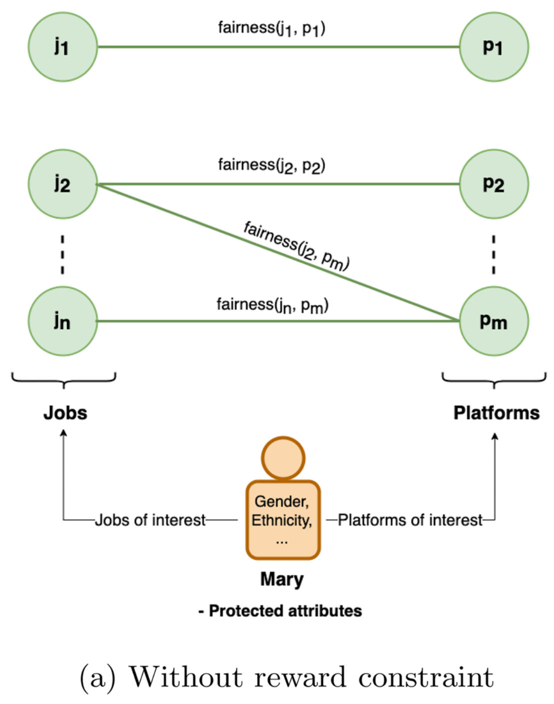
Sihem Amer-Yahia
sihem.amer-yahia@univ-grenoble-alpes.fr

[1] Computer Science Department, American University of Beirut, Beirut, Lebanon

[2] Electrical and Computer Engineering Department, American University of Beirut, Beirut, Lebanon

[3] CNRS, University Grenoble Alpes, Grenoble, France

[1] https://www.taskrabbit.com/.
[2] https://www.upwork.com/.



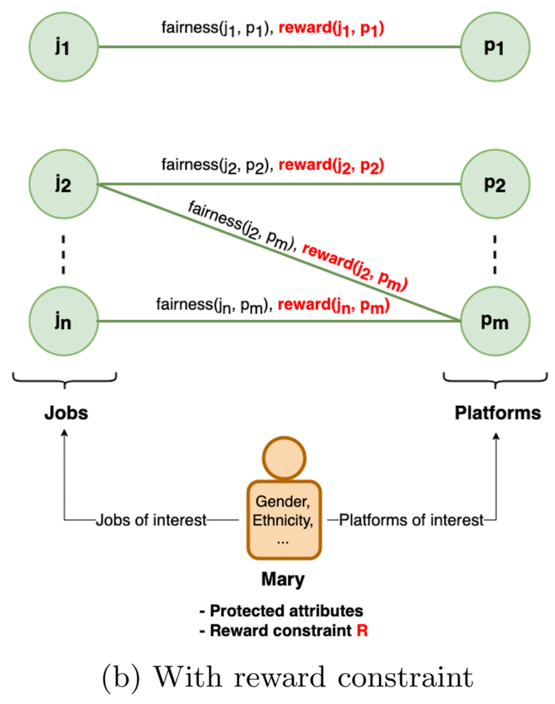

fairness(j1, p1), reward(j1, p1)
fairness(j2, p2), reward(j2, p2)
fairness(j2, pm), reward(j2, pm)
fairness(jn, pm), reward(jn, pm)
Jobs
Platforms
Jobs of interest
Platforms of interest
Gender, Ethnicity, ...
Mary
- Protected attributes
- Reward constraint R
(b) With reward constraint

Fig. 1 Motivating example: Mary the job seeker

Motivating Examples

We consider the example of Mary, a website developer, who is seeking some gigs in the summer (Fig. 1). Mary would like to know on which platform (e.g., TaskRabbit, Fiverr) and for which jobs (e.g., website design, website portability, website translation) she is most likely to land a gig. Today, Mary would need to apply to multiple jobs on several platforms. Mary would benefit from a tool that will provide her with the (job,platform) pairs where she is most likely to get hired. This could be made possible by analyzing how people like Mary, in terms of her demographics, skills and other characteristics, are treated on existing platforms for website development jobs. Moreover, Mary may also wish to make at least a certain amount of money for her summer vacation. This could be achieved by setting a constraint on the cumulated reward of the set of (job,platform) pairs she is most likely to get. Additional constraints such as the total time required to complete all gigs could also be desired and incorporated. Therefore, with the proposed tool, a job seeker such as Mary will not only save time in her search but also be empowered to only target jobs and platforms she is most likely to get selected for and for which she wishes to set her own conditions.

We now consider the case of Angela who wishes to deploy some jobs on several platforms (Fig. 2). Angela wants to treat job seekers fairly and at the same time stay within her budget. Without a dedicated tool, a job provider like Angela would not be able to assess fairness on different platforms and make an informed decision on where to deploy which jobs. Such a tool would optimize worker fairness and also incorporate her constraints. Job providers may have a specific per-platform budget. That is the case when platforms operate with different currencies or when a job provider already has some jobs running on a platform and wants to cap the number of jobs. For example, Angela may already have some jobs deployed on TaskRabbit and some funds on Prolific Academic. In that case, she will deploy her remaining jobs in such a way that she maximizes worker fairness and satisfies platform-specific budgets. Providing such expressive tools for job seekers and job providers is the topic of our work.

We focus on group fairness, which is defined as the fair treatment of all groups of people [4, 37], where groups are defined using protected attributes such as gender, age and ethnicity. For example, the worker groups could be males, asians, black females, young white males, etc. Our framework can accommodate multiple group fairness definitions as long as they rely on ranking or scoring of workers, i.e., fairness of exposure, which directly relates to the chances of workers landing jobs [31]. It does so by defining a single function $f(j, p, g)$, where j is a job, p is a platform, g is a demographic group, and $f(j, p, g)$ is a fairness value of job j on platform p for group g. To serve both job seekers and job providers, we formulate several optimization problems that aim to maximize worker group fairness subject to desired constraints such as payment and number of jobs.

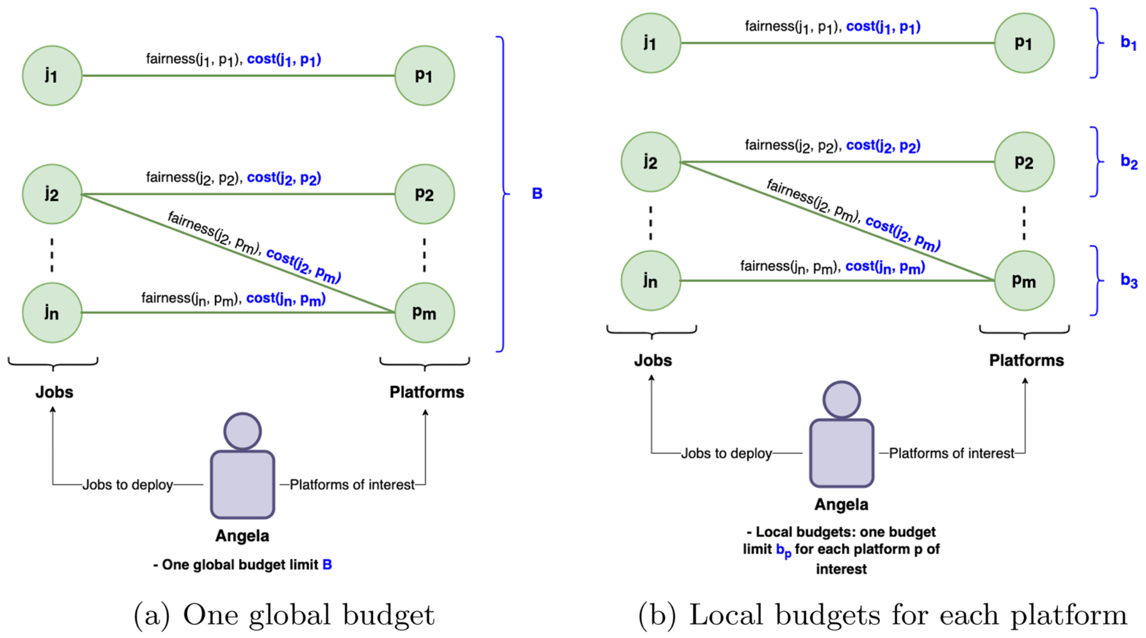

j1
j2
jn
P1
P2
Pm
fairness(j1, p1), cost(j1, p1)
fairness(j2, p2), cost(j2, p2)
fairness(j2, pm), cost(j2, pm)
fairness(jn, pm), cost(jn, pm)
B
b1
b2
b3
Jobs
Platforms
Jobs to deploy
Platforms of interest
Angela
- One global budget limit B
- Local budgets: one budget limit bp for each platform p of interest
(a) One global budget
(b) Local budgets for each platform

Fig. 2 Motivating example: Angela the job provider

Optimization Problems for Job Seekers

Our first and second optimization problems aim to maximize worker fairness for job seekers. Given a set of worker groups that the job seeker belongs to, a set of jobs of interest and a set of platforms on which these jobs might be available, our first optimization problem seeks the top-k fairest job-platform pairs. The worker can then use those k retrieved pairs to focus her efforts on when applying for jobs. We also consider the case where jobs are associated with rewards. That is, we assume that each job available on a platform is associated with a reward. This constitutes the basis for our second optimization problem, where the goal is to find the top-k fairest job-platform pairs such that their total reward is above a certain threshold. In this case, the worker's goal is to find the top-k fairest job-platform pairs that increase her chances of landing a job, while guaranteeing a minimum reward or payment.

Optimization Problems for Job Providers

Our third and fourth optimization problems aim to maximize worker fairness when a job provider is deploying a set of jobs on different platforms. Each job is associated with a cost, e.g., the worker compensation for completing that job, and this cost may differ from one platform to the other. Given a set of jobs to be deployed on a set of platforms and a budget, the third optimization problem seeks to assign each job to at most one platform such that the total cost of the jobs assigned does not exceed the budget and the total fairness of the assigned jobs is maximized. We impose that each job is deployed on at most one platform to reduce deployment cost and satisfy as many providers as possible. A variation of this optimization problem is: given a set of jobs to be deployed on a set of platforms and a budget *for each platform*, our goal is to assign each job to at most one platform such that the total cost of the jobs assigned to each platform does not exceed its budget, and the total worker fairness of the assigned jobs is maximized. The result of both optimization problems can be used by the job provider to decide on which platforms to deploy her jobs so as to maximize worker fairness subject to budget constraint(s) the job provider might have.

Computational Solutions

We prove that three of our four optimization problems are computationally hard by reductions from well-known NP-hard problems such as Knapsack [20] and general assignment problems [21], and we propose algorithms to efficiently solve them. More precisely, for the first job seeker optimization problem, we propose an adaptation of Fagin's top-k algorithm [10]. For the second job seeker problem, we propose a dynamic programing (DP) algorithm. Similarly, for the first job provider optimization problem, we also propose a dynamic programming algorithm and finally, for the second job provider problem, we explore various exact and approximation algorithms from the literature.

Empirical Validation

We design a series of experiments using synthetic and semi-synthetic data generated from TaskRabbit, a real-world online labor platform, to evaluate our proposed framework and algorithms. We use synthetic data to demonstrate the scalability of our algorithms as the number of jobs, the number of platforms and the number of worker groups increase and to compare them to adequate baselines. To compute fairness values, we use the two metrics defined in [1], namely Earth Mover Distance (EMD) and Exposure. Our experiments demonstrate that our algorithms scale very well and that they consistently outperform their baselines. On the other hand, we use semi-synthetic data to conduct case studies that highlight the merits of the solutions generated by our algorithms from a qualitative perspective. To create semi-synthetic data, we propose a data generation protocol based on interventions [30] to create multiple worlds, each of which simulates a platform. An intervention is a sampling of workers from a snapshot of TaskRabbit such that the sampled "world" matches a desired distribution of protected attributes (in our case either on gender or ethnicity). We run a series of qualitative experiments on different worlds. Our results confirm that our framework can indeed increase the chances of job seekers landing jobs and can result in maximizing worker fairness when job providers are deploying jobs, subject to various constraints such as reward or budget.

In summary, this paper makes the following contributions:

1. We formulate four novel optimization problems to maximize group fairness for workers when job seekers are looking for multiple jobs on multiple online labor platforms and when job providers are deploying multiple jobs on multiple online labor platforms.
2. We prove that three of our optimization problems are computationally hard, and propose algorithms to solve the four problems efficiently.
3. We establish a benchmark of synthetic and semi-synthetic data based on interventions to evaluate our algorithms both from a scalability perspective as well as from a usability one. Given that there exists no available benchmarks to perform such evaluations, our established benchmark and proposed experimental framework is thus a major contribution of this work.

The rest of the paper is organized as follows. In Sect. 2, we review related work that addresses fairness in online labor platforms. In Sect. 3, we describe our proposed framework, which is composed of four optimization problems and algorithms to solve them efficiently. In Sect. 4, we describe the experiments that we used to evaluate our proposed framework and their results. Finally, we conclude and present future work directions in Sect. 5.

2 Related Work

Fairness of ranking is an increasingly trending topic in research. Many works have already underlined the importance of fair rankings, and their impact on the actual selection of ranked items by users. As Singh and Joachims explained in [32], the probability of a ranked item being selected (e.g., a job candidate being hired) decreases significantly with lower ranking positions; a concept referred to as *exposure*. Along the same topic, the experiment in [19] studied user behavior when presented with manipulated Google search results, and found that users exhibit "partial bias" toward an item's rank, tending to select items at the top of search results. Fairness of ranking is thus especially important for online labor platforms, where unfair rankings of workers can lead to disparate distributions of work opportunities or income [3]. In this work, we focus only on group fairness, as opposed to individual fairness [3, 7, 26], which is the subject of our future work.

Many notable works focused on assessing fairness of a worker ranking in online labor platforms. For instance, the authors in [16] found evidence of bias in two prominent online labor platforms, TaskRabbit and Fiverr. In both platforms, they found that perceived gender and race have significant correlations with worker evaluations, and even with worker rankings in the case of TaskRabbit. In [6], the author examined gender bias in the resume search platforms: Indeed, Monster and CareerBuilder. Two notions of fairness issues were considered: a) *ranking bias*, which is the disparity of ranking distributions across genders (*group* unfairness), and b) *unfairness*, i.e., the gap in ranking between male and female applicants having the same qualifications (*individual* unfairness). The author found evidence of both issues on all three platforms.

Notable efforts have also been made to quantify unfairness [8, 9, 13, 14]. In [8, 9, 14], the authors formulated an optimization problem to find the partitioning of workers (based on their protected attributes) that exhibits the highest unfairness based on a given scoring function. They used Earth Mover's Distance (EMD) between score distributions as a measure of unfairness. In [1], the authors proposed a unified framework to study fairness in online jobs. They defined two generic fairness problems: *quantification*, which is finding the k worker groups, or jobs or locations, for which a job search site is most or least unfair, and *comparison*, which is finding the locations at which fairness between two groups differs from all locations, or finding the jobs for which fairness at two locations differ from all jobs for instance. They adapted Fagin top-k algorithms to address their fairness problems and case-studied two particular job search sites: Google job search and TaskRabbit.

To address fairness of ranking, various methods have been proposed to actively generate fair rankings. Many of them are *post-processing* methods (e.g., [3, 5, 23, 24, 35, 36]), where given an existing ranking of items, a new ordering of the items is generated so as to satisfy certain fairness constraints. On the other hand, *in-processing* methods address ranking bias of an algorithm at the training phase, such as the DELTR Learn-to-Rank framework in [34].

Our proposed work differs from all the reviewed related work above in that it is, to the best of our knowledge, the first to establish a generic framework that can be used to assess and compare worker fairness of multiple jobs on multiple online labor platforms. Our framework can accommodate all definitions of group fairness proposed before. It also has multiple use cases from the perspective of both job seekers and job providers. It can be deployed as a stand-alone service on top of existing online labor platforms to maximize fairness of job-matching services on these platforms when job seekers are being matched to jobs and when job providers are deploying jobs on these platforms. Our framework is theoretically founded and we propose an extensive and thorough experimental setup to evaluate it using both synthetic and real-world generated data.

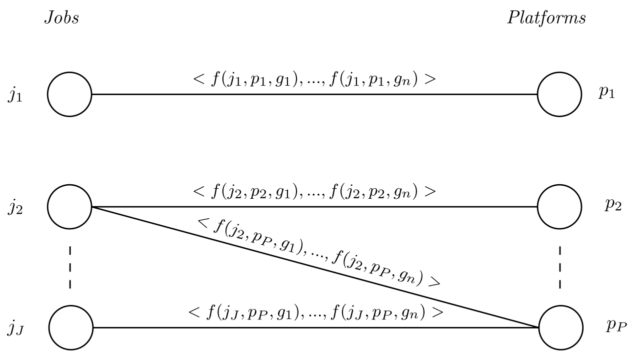
3 Framework

3.1 Setting

We assume that our framework has access to an unbounded number of platform signatures. A platform signature is a list of job descriptions available on the platform, and where each job is associated with a list of worker profiles that indicate the protected attributes of each worker and the score or rank of the worker for the job by the platform. We also assume that the same job can be available on multiple platforms, and for each job-platform pair, we define all worker groups using one or more protected attributes of the workers provided by the platform signature. For example, if the protected attributes are gender, ethnicity and age, then the worker groups would be males, females, asians, whites, blacks, black females, young white males and so on. Finally, we assume that each job-platform pair is associated with a fairness value for each worker group on each platform. The fairness value of a worker group depends on the ranking or scores of the workers that belong to that group and thus can be different for different platforms.

More precisely, a job j for worker group g on platform p is associated with a fairness value $f(j, p, g)$. Without loss of generality, we assume that $f(j, p, g)$ is a value between 0 and 1, and that the higher the value is, the more fair job j is considered for group g on platform p. To obtain such fairness values for each job-platform-group tuple, we assume the presence of a blackbox that takes as input a job j, a platform p and a group g and returns a fairness value $f(j, p, g)$ between 0 and 1. The fairness values are computed using the platform signature which indicates for each job, the list of worker profiles (i.e., their protected attributes) and their scores or ranks for the job by the platform. In our experiments, we make use of the framework in [1], which uses two different notions for computing group fairness. However, other notions of group fairness can also be used, as long as they can be computed using the scores or ranks of workers with respect to jobs [2, 28, 31].

Furthermore, we assume the presence of two predicates: $a(j, p)$ which is only true if job j is available on platform p and $e(j, p, g)$ which is only true if job j is available on platform p for group g. This is done to accommodate the fact that in practice in online labor platforms not all jobs are and not all worker groups are available on every platform. Our framework thus operates on an incomplete weighted bipartite graph where the first set of nodes represent jobs, the second set of nodes represent platforms and there is an edge between a job j and a platform p only if $a(j,p) = true$. Moreover, each edge in this bipartite graph is associated with a set of weights $\{f(j,p,g) | g \in G \wedge e(j,p,g) = true\}$ that



Jobs
Platforms
j_1
$< f(j_1,p_1,g_1),...,f(j_1,p_1,g_n) >$
p_1
j_2
$< f(j_2,p_2,g_1),...,f(j_2,p_2,g_n) >$
p_2
$< f(j_2,p_P,g_1),...,f(j_2,p_P,g_n) >$
j_J
$< f(j_J,p_P,g_1),...,f(j_J,p_P,g_n) >$
p_P

Fig. 3 An example bipartite graph with jobs on one side and platforms on the other side. Each edge between a job j and a platform p has a set of weights representing the fairness values of job j for the different groups g on platform p

Table 1 Summary of terminology

Variable	Meaning
j	Job
p	Platform
g	Group
$f(j, p, g)$	Fairness of job j for group g on platform p
$a(j, p)$	Predicate indicating the availability of job j on platform p
$e(j, p, g)$	Predicate indicating the availability of group g for job j on platform p

correspond to the different fairness values for the different groups that exist in the platform p for job j. Figure 3 shows an example of such a bipartite graph. Table 1 provides a summary of the main terminology in this paper.

The main goal of our framework is to assess and compare worker fairness of multiple jobs on multiple platforms, which can then be used to maximize fairness of job-matching services on online labor platforms when job seekers are being matched to jobs and when job providers are deploying jobs on these platforms. To achieve this goal, we define four different optimization problems, two for the job seeker case and two for the job provider case. We prove that three of our optimization problems are at least as hard as NP-hard problems and we propose a set of algorithms to solve the four of them efficiently.

3.2 Maximizing Fairness for Job Seekers

A job seeker is a person looking for the top-k fairest jobs available on different platforms that fits her interests or skills. A job seeker belongs to multiple demographic groups. For example, a job seeker can be female, white and middle-aged. We also consider combinations of these values to exhaust all the groups the job seeker belongs to. That is, in our example, the job seeker would be also a white female, a middle-aged white and a middle-aged white female. Our first optimization problem for maximizing fairness for job seekers is defined below.

Problem 1 *(Unconstrained) Job Seeker Problem* Given a set of demographic groups G that the job seeker belongs to, a set of jobs of interest J and a set of platforms P on which these jobs might be available, our goal is to find the top-k fairest (j, p) pairs, where $j \in J$ is a job, $p \in P$ is a platform, and the pair (j, p) means job j on platform P. Our job seeker problem can then be formulated as the following optimization problem:

$$\operatorname*{argmax}_{S} \sum_{(j,p)\in S} \min_{g\in G \wedge e(j,p,g)=true} f(j,p,g)$$

$$\begin{aligned}\text{subject to: } & S \subseteq J \times P \\ & a(j,p) = true \ \forall (j,p) \in S \\ & |S| = k\end{aligned}$$

Since each job seeker belongs to different worker groups, we need to aggregate the different fairness values for each group the job seeker belongs to in order to obtain a single fairness value for a job-platform pair. In the optimization problem above, we use minimum as an aggregation operator. Thus, we take a conservative worst-case approach here to quantify the fairness value of a job-platform pair for a given job seeker. This is motivated by the large body of literature on intersectional fairness [12, 15, 22]. Other aggregation methods such as taking the average or the maximum can be also applied without any fundamental changes.

The input in the job seeker problem is a set of jobs J, a set of platforms P and all the demographic groups G that the job seeker belongs to. A naive approach to solve the job seeker problem defined above is to loop over all jobs, all the platforms and all the groups, and for each job-platform pair (j, p) such that $a(j, p)$ is true, it computes the minimum fairness for that pair overall groups G the job seeker belongs to and for which $e(j, p, g)$ is true. It then returns the k job-platform pairs with the highest minimum fairness over all groups G. The complexity of this naive approach is thus $O(|J||P||G|)$.

Algorithm 1 Top-k Job Seeker Algorithm

```
Input: a set of jobs $J$, a set of platforms $P$, a set of groups $G$, $k$
output: the k $(j, p)$ pairs with the highest minimum fairness over all groups $G$
$topk \leftarrow minHeap()$                                        ▷ Initialization
$cursor \leftarrow 0$
while $topk.minValue() < \tau$ or $topk.size() < k$ do
    $\tau \leftarrow -\infty$
    for $g \in G$ do
        $((j,p), f(j,p,g)) \leftarrow I_g.getEntry(cursor)$      ▷ Read entry at current line (cursor)
        if $j \in J$ and $p \in P$ then
            if $\tau < f(j,p,g)$ then                           ▷ Update threshold value
                $\tau \leftarrow f(j,p,g)$
            end if
            $min \leftarrow f(j,p,g)$
            for $g' \in G$ and $g' \neq g$ do     ▷ Perform random access on all other lists
                if $e(j,p,g')$ is true then
                    $f(j,p,g') \leftarrow I_{g'}.getValue((j,p))$
                    if $f(j,p,g') < min$ then
                        $min \leftarrow f(j,p,g')$
                    end if
                end if
            end for
            if $topk.size() < k$ then                       ▷ Update top-k set (if needed)
                $topk.insert(((j,p), min)$
            else
                if $topk.minValue() < min$ then
                    $topk.pop()$
                    $topk.insert((j,p), min)$
                end if
            end if
        end if
    end for
    $cursor \leftarrow cursor + 1$
end while
return $topk$
```

A more efficient approach can make use of optimal aggregation algorithms such as Fagin's algorithm [10] provided we use a monotone aggregation function (such as the minimum in our formulation) to compute the fairness value of a job-platform pair over groups. To be able to do this, we assume the existence of a set of inverted lists, one for each worker group g. The inverted list I_g contains an entry for each job-platform pair (j, p) where $e(j, p, g)$ is true. The entries in I_g are sorted in descending order based on the fairness values $f(j, p, g)$.

Our optimal aggregation algorithm (Algorithm 1) is an adaptation of Fagin's Threshold algorithm to solve our job seeker problem. The algorithm operates on $|G|$ inverted lists, one for each group, and it uses a threshold value τ initially set to $-\infty$, a cursor (line counter) initially set to 0

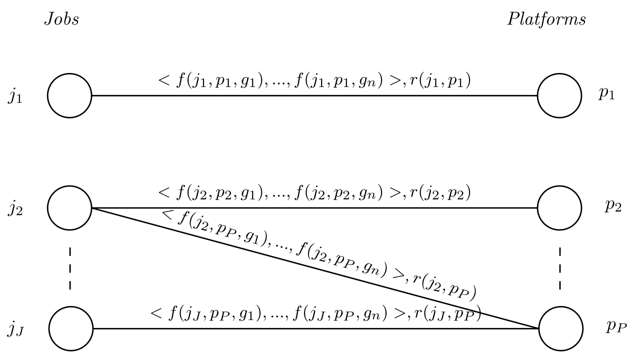

Jobs
Platforms
j_1
$< f(j_1,p_1,g_1),...,f(j_1,p_1,g_n) >, r(j_1,p_1)$
p_1
j_2
$< f(j_2,p_2,g_1),...,f(j_2,p_2,g_n) >, r(j_2,p_2)$
p_2
$< f(j_2,p_P,g_1),...,f(j_2,p_P,g_n) >, r(j_2,p_P)$
j_J
$< f(j_J,p_P,g_1),...,f(j_J,p_P,g_n) >, r(j_J,p_P)$
p_P

Fig. 4 An example bipartite graph for the constrained job seeker problem. In addition to the fairness values per group, each edge between a job j and a platform p has a weight $r(j, p)$ representing the reward of job j on platform p

and a min-heap *topk* that will store the top-k job-platform pairs seen so far. The algorithm then reads the inverted lists in parallel using sequential access. It starts by reading the first entry ($cursor = 0$, so first line) from each list. Each of the entries read corresponds to a job-platform pair, and its associated fairness value for the group corresponding to the inverted list that entry belongs to. τ is then set to the largest of these values, and for each of the pairs, we derive its aggregated fairness value by looking up its equivalent entries from the other inverted lists (using *random access*). The *topk* set is updated with the newly read pairs (and their aggregated fairness values) if necessary, and *cursor* is incremented by 1 for the next iteration (so as to read the next line of the lists). The algorithm keeps iterating until *topk* contains k elements *and* τ becomes smaller than the smallest fairness value in *topk*.

Note that a group g might not be available for a certain job-platform pair (j, p) (i.e., $e(j, p, g) = 0$). Hence, this job-platform pair will not be present in the inverted list I_g. If at each iteration, we update the threshold τ to be the minimum of the last read fairness values (as custom in the traditional Fagin's Threshold algorithm), then the algorithm might end up stopping too early and potentially missing some job-platform pairs with high minimum fairness that are present in some lists but not in others. This explains why in our adaptation of the algorithm, the threshold τ is updated to be the *maximum* of the last seen values, rather than their *minimum* (lines 10 to 12 in Algorithm 1). Also note that Fagin's Threshold algorithm is known to be instance optimal [10].

We also consider a scenario where the job seeker is interested in retrieving the top-k fairest job-platform pairs, subject to some user-defined constraints. For instance, one such constraint could be minimum reward integrated as follows. Assume that each job j available on platform p is associated with a reward $r(j, p)$, representing the earnings the job seeker can make by executing job j on platform p. Thus, each edge in our bipartite graph will include an additional weight as shown in Fig. 4. In this case, the goal of the job seeker can be formulated as the following optimization problem.

Problem 2 *Constrained Job Seeker Problem* Given a set of demographic groups G that the job seeker belongs to, a set of jobs of interest J, and a set of platforms P on which these jobs might be available, our goal is to find the top-k fairest (j, p) pairs, where $j \in J$ is a job, $p \in P$ is a platform, and the pair (j, p) means job j on platform P and such that the total reward for the selected job-platform pairs is above a certain threshold R. Our constrained job seeker problem can then be casted as the following optimization problem:

$$\underset{S}{\operatorname{argmax}} \sum_{(j,p)\in S} \min_{g\in G \wedge e(j,p,g)=true} f(j,p,g)$$

$$\begin{aligned}\text{subject to: } & S \subseteq J \times P\\ & a(j,p) = true\ \forall (j,p) \in S\\ & |S| = k\\ & \sum_{(j,p)\in S} r(j,p) \geq R\end{aligned}$$

The same problem can be formulated as an integer linear programming (ILP) optimization problem as follows:

$$\max \sum_{j\in J}\sum_{p\in P} \min_{g\in G\wedge e(j,p,g)=true} f(j,p,g) \times x(j,p)$$

$$\begin{aligned}\text{subject to: } & x(j,p) \in \{0,1\}\ \forall j \in J, \forall p \in P\\ & x(j,p) = 1 \rightarrow a(j,p) = true\ \forall j \in J, \forall p \in P\\ & \sum_{j\in J}\sum_{p\in P} x(j,p) = k\ \forall j \in J, \forall p \in P\\ & \sum_{j\in J}\sum_{p\in P} r(j,p) \times x(j,p) \geq R\end{aligned}$$

Theorem 1 *The optimization variant of the knapsack problem is polynomial-time reducible to the constrained job seeker problem, and therefore, the latter problem is at least as hard as the former.*

Proof Note that by having only one group and one platform, the constrained job seeker problem reduces to the following. Given a list M of pairs $m_i = (f_i, r_i)$, where f_i is the assigned fairness value and r_i the reward value, select k pairs such that fairness is maximized and the total reward is at least R. Using this version of the problem, we give a polynomial-time reduction from the optimization version of Knapsack. Given a list L of pairs $a_i = (v_i, w_i)$, where v_i represents the value of the pair and w_i its weight and an integer W, the Knapsack problem asks for a subset of L of maximum value such that the total weight is at most W.

Given an instance of the knapsack problem where $|L| = n$, create a list M of n pairs $m_i = (f_i, r_i)$ where $f_i = v_i$ and $r_i = W - w_i$. Moreover, add n additional pairs $(0, W)$ to M. Set $k = n$ and $R = (n - 1)W$. We prove equivalence of both instances in Appendix A. In other words, we prove that L contains a subset of total value X, satisfying the Knapsack constraints, if and only if M contains a subset of size n with total fairness X, satisfying the constrained job seeker problem constraints. □

Note that since the knapsack optimization problem is known to be at least as hard as its decision version, also known to be NP-Complete [20], Theorem 1 implies that polynomial-time algorithms for the constrained job seeker problem are unlikely to exist.

Next, we describe how to solve this problem efficiently in practice. The similarity with the knapsack problem gives a nearly immediate dynamic programming (DP) solution that we describe in Algorithms 2 and 3. This solution expects fairness values to be integers. For non-integer fairness values (like in our case), we provide a systematic method of converting them into integers in Sect. 4.3.4.

Algorithm 2 Constrained Job Seeker Algorithm

```
1: **Input:** A set of jobs $J$, a set of platforms $P$, a set of groups $G$, and two integers $k$ and $R$
2: **Output:** The $k$ $(j, p)$ pairs with the highest minimum fairness over all groups $G$ having reward at least $R$

    ▷ Step 1: Initialization + aggregation of fairness values

3: $minFair[1...len(J)][1...len(P)] \leftarrow$ new 2D array initialized to $+\infty$
4: for $j \in J$ and $p \in P$ and $g \in G$ do
5:     if $e(j, p, g) = true$ then
6:         $minFair[j][p] \leftarrow \min(minFair[j][p], f(j, p, g))$
7:     end if
8: end for

9: $L \leftarrow$ Empty list
10: for $j \in J$ and $p \in P$ do
11:     $(j, p, f, r) \leftarrow (j, p, minFair[j][p], r(j, p))$
12:     $L.append((j, p, f, r))$
13: end for

    ▷ Step 2: Call recursive DP procedure (see Algorithm 3)

14: $DP[0...len(L)][0...k][0...R] \leftarrow$ new 3D array initialized to $-1$
15: $choice[0...len(L)][0...k][0...R] \leftarrow$ new 3D array initialized to $-1$
16: $maxFairness \leftarrow$ MaxFairness$(1, L, k, R, DP, choice)$
17: if $maxFairness = -\infty$ then return $\phi$

    ▷ Step 3: Read result (optimal assignment) from the choice matrix and return

18: $i \leftarrow 0, \quad result \leftarrow \phi$
19: while $i \neq len(L)$ do
20:     if $choice[i][k][R] = 0$ then
21:         $i \leftarrow i + 1$
22:         continue
23:     end if
24:     $result.add((j, p))$
25:     $k \leftarrow k - 1$
26:     $R \leftarrow \max(0,\ R - L[i].r)$
27:     $i \leftarrow i + 1$
28: end while

29: return $result$
```

Algorithm 3 Recursive Maximum Fairness Algorithm

```
procedure MaxFairness($i, L, k, R, DP, choice$)
    if $k = 0$ then return $R = 0$ ? $0 : -\infty$
    if $i > N$ then return $-\infty$
    if $DP[i][k][R] \neq -1$ then return $DP[i][k][R]$

    $dontTakePair \leftarrow$ MaxFairness($i + 1, L, k, R, DP, choice$)
    $takePair \leftarrow$ MaxFairness($i + 1, L, k - 1, max(0, R - L[i].r), DP, choice$)

    if $dontTakePair = -\infty$ and $takePair = -\infty$ then return $DP[i][k][R] = -\infty$
    if $dontTakePair \neq -\infty$ and $takePair \neq -\infty$ then
        $choice[i][k][R] \leftarrow (dontTakePair < L[i].f + takePair)$
        return $DP[i][k][R] \leftarrow \max(dontTakePair, L[i].f + takePair)$
    end if
    if $dontTakePair \geq 0$ then
        $choice[i][k][R] \leftarrow 0$
        return $DP[i][k][R] \leftarrow dontTakePair$
    end if
    $choice[i][k][R] \leftarrow 1$
    return $DP[i][k][R] \leftarrow L[i].f +$ $takePair$
end procedure
```

Algorithm 2 starts by aggregating the fairness values for each job-platform of interest over all the groups the job seeker belongs to (lines 3 to 8). The algorithm then constructs a list L consisting of each job-platform pair of interest, its aggregated fairness values and its reward (lines 9 to 13). The algorithm then invokes Algorithm 3 (line 16), which takes as input a list L of all pairs in $J \times P$, an index i corresponding to the pair we are currently considering, k representing the number of pairs we need to select and R representing the minimum total reward required for the selected pairs. In addition, the algorithm takes two additional arguments: *DP* and *choice*, which are both 3D arrays. Algorithm 3 is used to find the maximum fairness value that can be induced by a list of exactly k pairs in $L[i...|L|]$ such that the sum of the reward of the k pairs is at least R.

To find the optimal fairness assignment, we need to make a decision about the i^{th} pair in $|L|$. Let r_i be the reward value of pair i, and f_i its fairness value. From there, we have two options: either take the pair into the top-k pairs set, or discard it. Discarding the i^{th} pair means solving the problem for pairs $(i + 1)$ till $|L|$ (instead of pairs i to $|L|$), which is handled by the recursive call in line 5 of Algorithm 3. On the other hand, including the pair into the top-k set means solving a new subproblem on pairs $(i + 1)$ till L, from which we need to select $(k - 1)$ pairs instead of k, with a reward threshold of $R - r_i$. This subproblem is solved using the recursive call in line 6 of Algorithm 3.

To choose the best action, the algorithm tries both options (i.e., both recursive calls), and compares the optimal fairness attainable if we take the i^{th} pair $(f_i + f(L, i + 1, k - 1, R - r_i))$ to the optimal value if we leave it $(f(L, i + 1, k, R))$. Since our goal is to maximize fairness, then the largest of these two values is chosen and returned, breaking ties arbitrarily if any. The choice made (taking or discarding the pair i) is recorded in the array *choice*,to facilitate reconstructing the final top-k set.

This algorithm is further optimized by the use of DP. The algorithm has three main parameters: i, k and R. Since the total number of pairs in the list L is at most $|J| \times |P|$, therefore, the algorithm has a total of $|J||P|kR$ states. To ensure that no state will compute its own value more than once, we include the DP array to save the result of a state after first computation, so that whenever another recursive call is called with the same parameters, all computation is skipped and we directly return the pre-computed value (i.e., memoization).

In terms of complexity, this algorithm is composed of two stages. First, there is an initialization or preprocessing phase, which loops over the $|J| \times |P|$ input job-platform pairs, computes their aggregated fairness value over the $|G|$ groups and then arranges the pairs into a list of (*job*, *platform*, *fairness*, *reward*) tuples. This phase's time complexity is then $O(|J||P||G|)$. Second, comes the recursive phase with DP described above, having a time complexity of $O(|J||P|kR)$ (as we have $|J||P|kR$ states in total). So overall, this algorithm has a time complexity of $O(|J||P||G| + |J||P|kR)$.

3.3 Maximizing Fairness for Job Providers

A job provider is a person looking to deploy a set of jobs on different platforms. In online labor platforms, typically

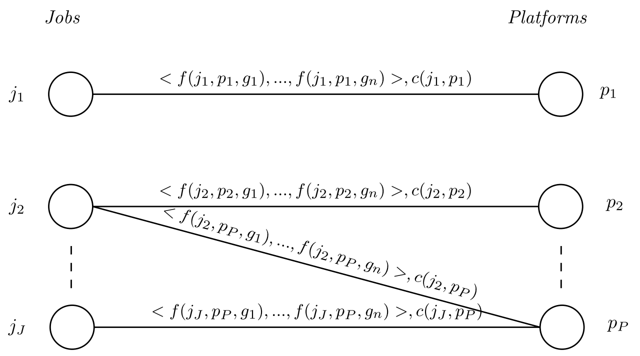

Jobs
Platforms
j_1
j_2
j_J
p_1
p_2
p_P
$< f(j_1,p_1,g_1),...,f(j_1,p_1,g_n) >, c(j_1,p_1)$
$< f(j_2,p_2,g_1),...,f(j_2,p_2,g_n) >, c(j_2,p_2)$
$< f(j_2,p_P,g_1),...,f(j_2,p_P,g_n) >, c(j_2,p_P)$
$< f(j_J,p_P,g_1),...,f(j_J,p_P,g_n) >, c(j_J,p_P)$

Fig. 5 An example bipartite graph for the job provider problem. In addition to the fairness values per group, each edge between a job j and a platform p has a weight $c(j, p)$ equal to the cost of deploying job j on platform p

each job j is associated with a cost $c(j, p)$ on every platform p it is available on, and this cost differs from one platform to another. This extends our bipartite graph in Fig. 3 so that each edge is now associated with an additional weight that represents the cost of deploying job j on platform p. An example of such graph is depicted in Fig. 5. The goal of the job provider is thus to deploy the jobs on the platforms such that the overall worker group fairness is maximized, while satisfying a budget constraint. To reduce deployment cost, we impose that each job is deployed on *at most one platform*. This goal can be formulated as the following optimization problem.

Problem 3 *Job Provider Problem with Global Budget* Given a set of jobs J to be deployed on a set of platforms P and a budget B, our goal is to assign each job $j \in J$ to at most one platform $p \in P$ such that the total cost of the jobs assigned does not exceed the budget B and the total fairness of the assigned jobs is maximized. Our job provider problem can be formulated as the following optimization problem (in integer linear programming form):

$$\max \sum_{j\in J} \sum_{p\in P} \min_{g|e(j,p,g)=true} f(j,p,g) \times x(j,p)$$

$$\begin{aligned} \text{subject to: } & x(j,p) \in \{0,1\} \; \forall j \in J, \forall p \in P \\ & x(j,p) = 1 \rightarrow a(j,p) = true \; \forall j \in J, \forall p \in P \\ & \sum_{j\in J} \sum_{p\in P} c(j,p) \times x(j,p) \le B \\ & \sum_{p\in P} x(j,p) \le 1 \; \forall j \in J \end{aligned}$$

In some cases, a job provider might have a separate budget for each platform on which the jobs are to be deployed, rather than a global budget over all platforms. This could be the case when platforms operate with different currencies or when a job provider already has some jobs running on a platform and wants to cap the number of jobs. This can be formulated as the following optimization problem.

Problem 4 *Job Provider Problem with Local Budget* Given a set of jobs J to be deployed on a set of platforms P and a budget b_p for each platform $p \in P$, our goal is to assign each job $j \in J$ to at most one platform $p \in P$ such that the total cost of the jobs assigned does not exceed the total budget for all platforms for which the jobs are assigned, and the total fairness of the assigned jobs is maximized. Our second version of the job provider problem can be formulated as the following optimization problem:

$$\max \sum_{j\in J} \sum_{p\in P} \min_{g|e(j,p,g)=true} f(j,p,g) \times x(j,p)$$

$$\begin{aligned} \text{subject to: } & x(j,p) \in \{0,1\} \; \forall j \in J, \forall p \in P \\ & x(j,p) = 1 \rightarrow a(j,p) = true \; \forall j \in J, \forall p \in P \\ & \sum_{j\in J} c(j,p) \times x(j,p) \le b_p \; \forall p \in P \\ & \sum_{p\in P} x(j,p) \le 1 \; \forall j \in J \end{aligned}$$

We next prove that both job provider problems are computationally hard.

Theorem 2 *The job provider with global budget and the job provider with local budget problems are at least as hard as the optimization variant of the knapsack problem.*

Proof Constraining both problems to one group and one platform is equivalent to solving the optimization version of the knapsack problem, known to be at least as hard as the decision version, which is known to be NP-Hard. In other words, we can reduce an instance of the knapsack problem to an instance of either problem having only one group and one platform. □

Algorithm 4 Job Provider Problem with Global Budget Algorithm

```
1: Input: A set of jobs $J$, a set of platforms $P$, a set of groups $G$, and an integer $B$
2: Output: The maximum size subset of $(j, p)$ pairs with the highest minimum fairness having cost at most $B$

   ▷ Step 1: Initialization, aggregation of fairness values
3: $minFair[1...len(J)][1...len(P)] \leftarrow$ new 2D array initialized to $+\infty$.
4: for $j \in J$ and $p \in P$ and $g \in G$ do
5:     if $e(j, p, g) = true$ then
6:         $minFair[j][p] = \min(minFair[j][p], f(j, p, g))$
7:     end if
8: end for

   ▷ Step 2: Iterative DP: For each subproblem containing the first $i$ jobs, $DP[i][t]$ will store the optimal fairness obtainable from these jobs at budget limit $t$
9: $DP[0...len(J)][0...B] \leftarrow$ new 3D array initialized to 0.
10: for $i \in [0, len(J))$ and $t \in [0, B]$ do
11:     $dp[i+1][t] \leftarrow \max(dp[i+1][t], dp[i][t])$
12:     for $j \in [1...len(P)]$ do
13:         $(f, c) \leftarrow (minFair[J[i]][P[j]], c(J[i], P[j]))$
14:         if $c + t \leq B$ then
15:             $dp[i+1][c+t] \leftarrow \max(dp[i+1][c+t], f + dp[i][t])$
16:         end if
17:     end for
18: end for

   ▷ Step 3: Get total cost of the optimal assignment found
19: $maxFairness \leftarrow 0$
20: $b \leftarrow 0$
21: $N \leftarrow len(J)$
22: for $t \in [0...B]$ do
23:     if $dp[N][t] > maxFairness$ then
24:         $maxFairnes \leftarrow dp[N][t]$
25:         $b \leftarrow t$
26:     end if
27: end for

   ▷ Step 4: Read result (optimal assignment) from the DP matrix and return
28: $result \leftarrow$ Empty list
29: while $N \neq 0$ do
30:     $(j) \leftarrow J[N]$
31:     if $dp[N-1][b] \neq dp[N][b]$ then
32:         for $i \in [1...len(P)]$ do
33:             if $b \geq c(j, P[i])$ and $dp[N][b] = minFair[j][P[i]] + dp[N-1][b - c(j, P[i])]$ then
34:                 $result.append((j, P[i]))$
35:                 $b \leftarrow b - c(j, P[i])$
36:                 break
37:             end if
38:         end for
39:     end if
40:     $N \leftarrow N - 1$
41: end while

42: return $result$
```

Table 2 Platform statistics for the alternative worlds (in percentages)

World	Male	Female	World	Black	White	Asian
(a) Gender statistics			(b) Ethnicity statistics			
TaskRabbit	0.75	0.25	TaskRabbit	0.24	0.69	0.07
World1	0.26	0.74	World1	0.27	0.66	0.07
World2	0.50	0.50	World2	0.25	0.68	0.07
World3	0.30	0.70	World3	0.26	0.67	0.07
World4	0.70	0.30	World4	0.24	0.69	0.07
World5	0.74	0.26	World5	0.33	0.33	0.34
World6	0.72	0.28	World6	0.69	0.24	0.07
World7	0.74	0.26	World7	0.24	0.07	0.69
World8	0.75	0.25	World8	0.07	0.69	0.24

World	Male asian	Male black	Male white	Female asian	Female black	Female white
(c) Group statistics						
TaskRabbit	0.05	0.17	0.52	0.02	0.07	0.17
World1	0.02	0.06	0.18	0.05	0.21	0.48
World2	0.04	0.11	0.35	0.03	0.14	0.33
World3	0.02	0.07	0.21	0.05	0.20	0.46
World4	0.05	0.16	0.49	0.02	0.08	0.20
World5	0.26	0.24	0.25	0.08	0.09	0.08
World6	0.05	0.49	0.18	0.02	0.20	0.06
World7	0.52	0.17	0.05	0.16	0.07	0.02
World8	0.18	0.05	0.52	0.06	0.02	0.17

Like Problem 2, the similarity between Problem 3 and the knapsack problem gives a near-immediate dynamic programming algorithm, described in Algorithm 4. This approach is essentially an iterative DP method, akin to the Knapsack one, where increasingly large subproblems of the original problem are solved. Solving these subproblems gradually populates a DP matrix called DP, where $DP[i][t]$ stores the optimal fairness obtainable when considering the first i job-platform pairs, at budget limit t. Using this DP matrix to store the subproblems' optimal values helps in reducing computation time, by avoiding repetitive calculations. Complexity-wise, this algorithm is composed of two main parts: a preprocessing phase similar to the one in Algorithm 2, with running time $O(|J||P||G|)$, and the DP phase described above, that iteratively populates a $(|J||P|) \times B$ matrix, and thus has a running time of $O(|J||P|B)$. Therefore, the overall time complexity for this algorithm is $O(|J||P||G| + |J||P|B))$.

As for Problem 4, if the aggregation of fairness values for each group is done a priori, then the problem becomes equivalent to LEGAP, a variant of the generalized assignment problem (GAP) where each job must be assigned to *at most* one platform instead of *exactly* one [21]. And since LEGAP is proven to be equivalent to the "standard" GAP [21, 33], then Problem 4 (with pre-aggregated fairness values) is equivalent to GAP. This implies that Problem 4 is, like GAP, strongly NP-hard and it is therefore unlikely to admit pseudo-polynomial-time algorithms.

On the other hand, this equivalence suggests that GAP algorithms from the literature can be used to solve Problem 4 [11, 18, 21, 27]. The only adjustment required to the problem is to add a dummy platform p_{dummy}, set its associated fairness values to zero (so $f(j, p_{dummy}, g) = 0\ \forall j \in J, g \in G$), cost values to 1 (so $c(j, p_{dummy}) = 1\ \forall j \in J$), and its budget limit to $|J|$. This way, the choice of not assigning a job to any platform in the initial problem is now represented as assigning the job to the platform p_{dummy}, essentially creating a "none of the above" option. This satisfies the GAP constraint that all jobs must be assigned to exactly one platform, while still giving the option of not *actually* selecting a job, by assigning it to p_{dummy}. This adjustment thus creates an instance of GAP that is equivalent to our problem, and hence can be directly solved by available GAP algorithms.

For the GAP problem (which is equivalent to our problem), it is known that: 1) exact pseudo-polynomial-time algorithms (such as the DP-based methods) are unlikely to exist unless $\mathcal{P} \Im \mathcal{NP}$; and 2) polynomial-time approximation schemes with a mathematically guaranteed solution quality are also unlikely to exist, unless $\mathcal{P} \Im \mathcal{NP}$ [21]. Therefore, when proposing an adequate algorithm to solve Problem 4, we are left with two possible choices: either non

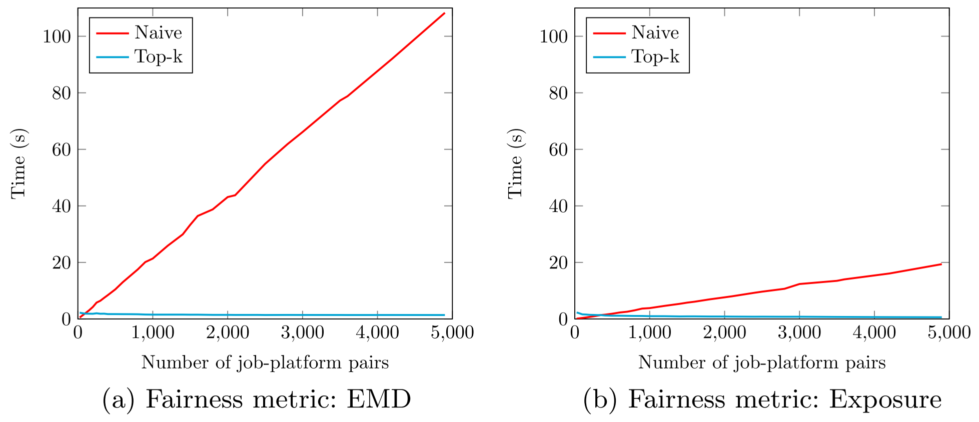


Fig. 6 Naive vs. Top-k runtimes for $k = 20$

polynomial-time exact algorithms, or more efficient heuristics with no mathematical guarantee on solution accuracy.

With this in mind, we start by first exploring exact GAP algorithms from the literature. A common outline for solving GAP is the branch-and-bound (BB) method. We examine three algorithms from this category: 1) the BB with multiplier adjustment method (MAM) by Fisher et al. [11, 21], 2) the BB with steepest descent MAM by Karabakal et al. [18] and 3) the BB with variable fixing by Posta et al. [27]. These three algorithms all use the BB technique, the main differences between them being the way lower bounds are computed, the branching strategies and extra computations involved (such as variable fixing in [27]). Assuming that fairness values are pre-computed, the worst-case scenario for the exact BB algorithms includes visiting all possible solutions, plus additional computations depending on the algorithm (e.g., computing the initial lower bound, or extra computations inside a search node). Therefore, all three exact algorithms have an exponential running time in the worst case.

For use cases where efficiency is more essential than solution accuracy, heuristic algorithms may also be worth considering. For this, we explore and test various heuristics from the literature that solve GAP, including: 1) MTHG, a polynomial-time greedy search with regret measure proposed by Martello and Toth [21]; 2) a Local Search Descent method by Osman [25]; and 3) a Tabu Search method by Osman [25]. In the next section, we compare all of these exact and heuristic algorithms, both in terms of performance and solution quality.

4 Experiments

To evaluate our framework, we design two sets of experiments. The first set aims to study the scalability of our algorithms to solve the different job seeker and job provider optimization problems as the number of jobs, the number of platforms and the number of worker groups increase. For such experiments, we rely on purely synthetic data. The second set of experiments aim to qualitatively analyze the solutions provided by our algorithms. For that, we use semi-synthetic data generated from a real-world online labor platform.

We divide this section as follows. First, we explain how the semi-synthetic dataset (used in qualitative experiments) is generated. Following that, we provide a summary of our findings. We then describe the different experiments (both scalability and qualitative) and their results for the job seeker problems. Finally, we describe the experiments and the results for the job provider problems.

4.1 Data Generation

To simulate multiple semi-synthetic platforms, we use the TaskRabbit dataset from [1], which is composed of 75% males and 25% females, and 24% blacks, 69% whites and 7% asians. We then generate eight different “worlds” from it using interventions [30]. An intervention is a sampling of workers from the initial dataset such that the sampled “world” matches a specific distribution of protected attributes (in our case either on gender or ethnicity). When generated, each of the obtained worlds is treated as a separate platform. The resulting dataset, consisting of the original TaskRabbit data and the eight new worlds, are saved, and we refer to these nine platforms collectively as the *alternative worlds*. In the remainder of this section, we will be interchangeably using *world* and *platform* to refer to platforms.

The worlds *world*1 to *world*4 are created based on gender interventions from the original world as follows: *world*1 has percentages of males and females switched compared to the original; *world*2 is composed of 50% males and 50% females; *world*3 is composed of 30% males and 70% females; and finally *world*4 is composed of 70% males and 30% females.

The worlds *world*5 to *world*8 are created based on ethnicity interventions from the original world as follows: *world*5 contains 33% black, 33% white and 34% asian workers. Worlds 6 through 8 are created by switching the percentages

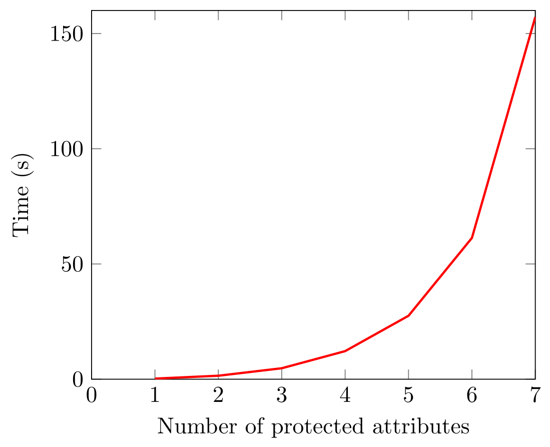


Fig. 7 Top-k algorithm runtime vs. number of protected attributes n ($k = 20$)

of two of the ethnicities from the original world. So, *world*6 is created by swapping the percentages of whites and blacks, *world*7 by swapping those of whites and asians and finally *world*8 by swapping those of blacks and asians. A summary of the resulting platforms and their worker distributions can be found in Table 2.

4.2 Summary of Results

4.2.1 Scalability Experiments

We first examine our solution to the unconstrained job seeker problem. We find that while the number of job-platform pairs retrieved (i.e., k) does not affect scalability, increasing the number of job-platform pairs and the number of protected attributes render the naive algorithm that loops over all jobs, platform and groups unusable. The top-k algorithm (Algorithm 1), on the other hand, scales very well. We then examine our solution to the constrained job seeker problem. We find that the DP algorithm (Algorithm 2) is much faster than an integer linear programming (ILP) baseline, for all values of k. Both algorithms' runtimes increase with the number of job-platform pairs but the observed increase for DP is less pronounced than for ILP.

We now turn to the job provider problem with global budget. The DP algorithm (Algorithm 4) is faster than an ILP baseline for all job-platform pairs, and its runtime is less steep than that of the ILP as the number of job-platform pairs increases. We also find that DP time increases linearly as the budget increases.

Finally, we examine our solution to the job provider problem with local budget. We compared three exact algorithms from the literature: 1) the branch-and-bound (BB) algorithm with multiplier adjustment method (MAM) by Fisher et al. [11, 21], 2) the BB with steepest descent MAM by Karabakal et al. [18] and 3) the BB with variable fixing by Posta et al. [27], to an ILP solver and found that they do not perform significantly faster. In fact, the method by Fisher et al. was far slower and the algorithm did not provide a solution after considerable time. We also considered three heuristic algorithms to solve this problem, namely: 1) MTHG by Martello and Toth [21]; 2) a Local Search Descent method by Osman [25]; and 3) a Tabu Search method by Osman [25]. Comparing these algorithms to an ILP solver, we found that all three heuristic algorithms perform much faster than the ILP solver for increasing values of job-platform pairs while returning solutions within 2% from optimality on average.

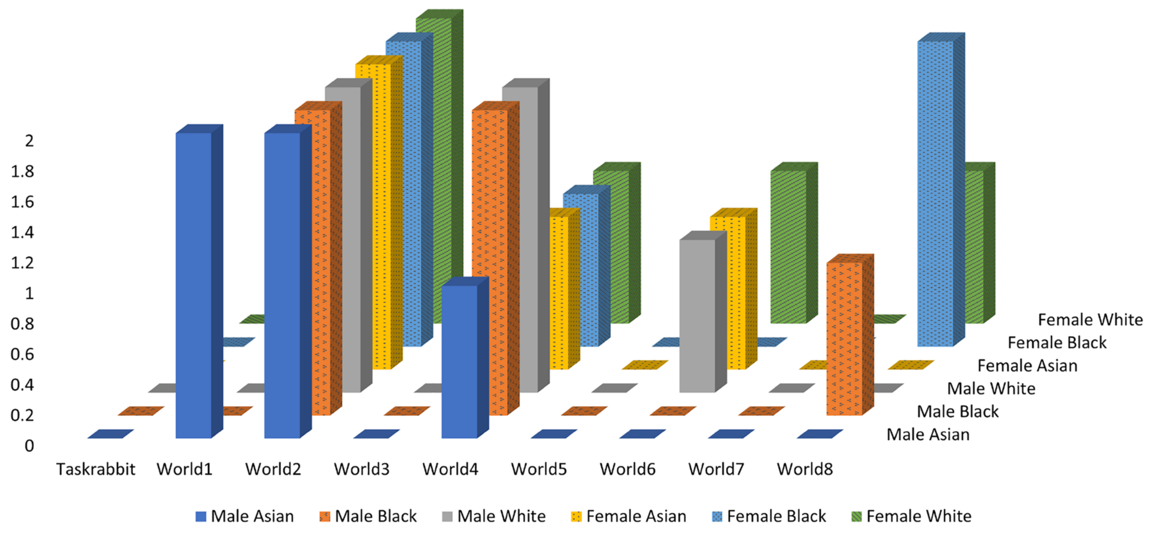


Fig. 8 Occurrences of each platform in the seekers' top-5 retrieved job-platform pairs

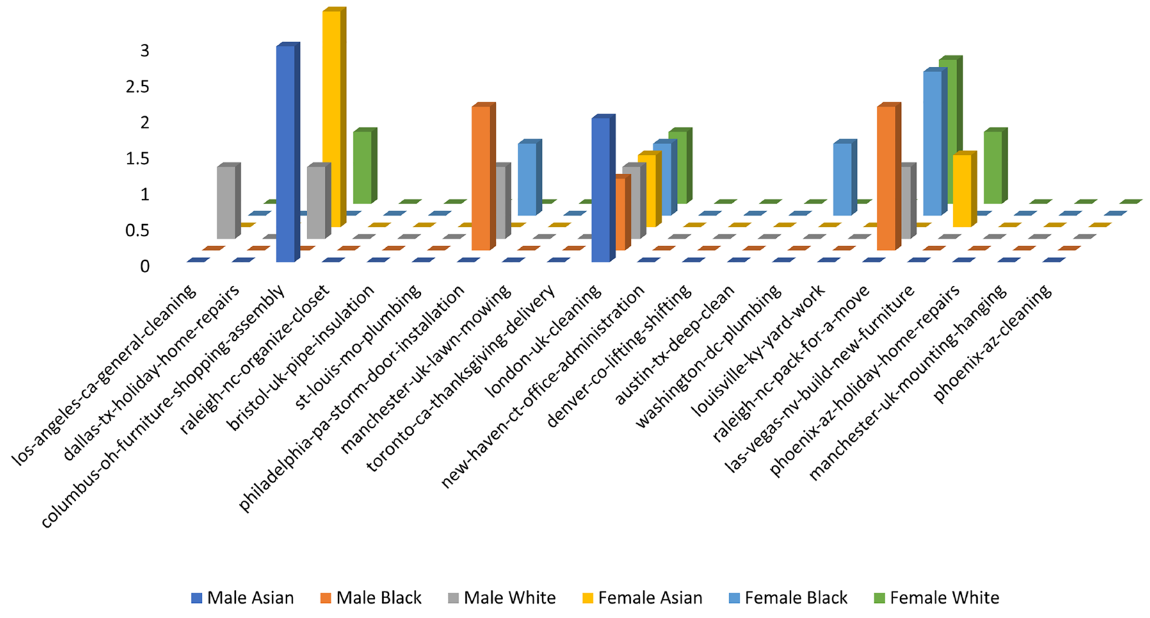


Fig. 9 Occurrences of each job in the seekers' top-5 retrieved job-platform pairs

4.2.2 Qualitative Experiments

In our qualitative experiments for the unconstrained job seeker problem, we observe that the top-5 fairest job-platform pairs vary across seekers. We also notice that some platforms and jobs occur more frequently than others in the top-5 retrieved pairs for different seekers. This implies that some platforms like *world2* and *world4* are fairer than others across worker groups. Similarly, some jobs like "London UK Cleaning," "Furniture Shopping and Assembly in Columbus, OH" and "Pack for a Move in Raleigh, NC" are fairer than others across worker groups. Finally, we also observe that seeking jobs on multiple platforms yields higher overall fairness value than seeking jobs on any of the platforms alone.

On the other hand, our qualitative experiments for the constrained job seeker problem allow us to observe that the top-5 job-platform pairs differ from the ones returned by the solution to the unconstrained one. This, in turn, indicates that the constraints are taking effect and validates the necessity of the constrained version of the problem.

In the qualitative experiments of the job provider problem with global budget, we find that the optimal fairness values returned are not the same for individual platforms, and that considering all platforms together yields a much better fairness value than any of the nine platforms separately. We also observe that a higher budget limit implies a better fairness value but only up until a certain point. When the algorithm reaches an optimal job-platform pairs selection, increasing the budget limit further does not improve the overall fairness value.

Finally, for the job provider problem with local budget, we first fixed a total overall budget and sought to determine what achieves more fairness: fewer platforms with higher budgets per platform, or more platforms with smaller budgets each. We find that choosing the right number of platforms is a trade-off. While increasing the number of platforms does give us more options in terms of job-platform pairs, it also divides the budget limit over more platforms, tightening the budget constraints. In such scenario, choosing the best "middle ground" number of platforms of interest should be handled on a case-by-case basis. Our last experiment shows that for the same input data and when selecting only one platform of interest, the fairness values obtained for the job provider problem with local budget are the same as those of the job provider problem with global budget, since for the case of one platform, the two problems are equivalent. However, when selecting multiple platforms of interest, the obtained fairness values are lower for the local budget problem variant. This is because for multiple platforms, the constraints of the job provider problem with local budget are tighter than just having a global budget constraint. This further justifies the problem version with local budget.

4.3 Job Seeker Experiments

4.3.1 Unconstrained Problem Scalability Experiments

We conducted two experiments to demonstrate the scalability of the top-k algorithm (Algorithm 1) we proposed to solve the unconstrained job seeker problem. Both experiments were run on an Apple MacBook Pro with a 2.3

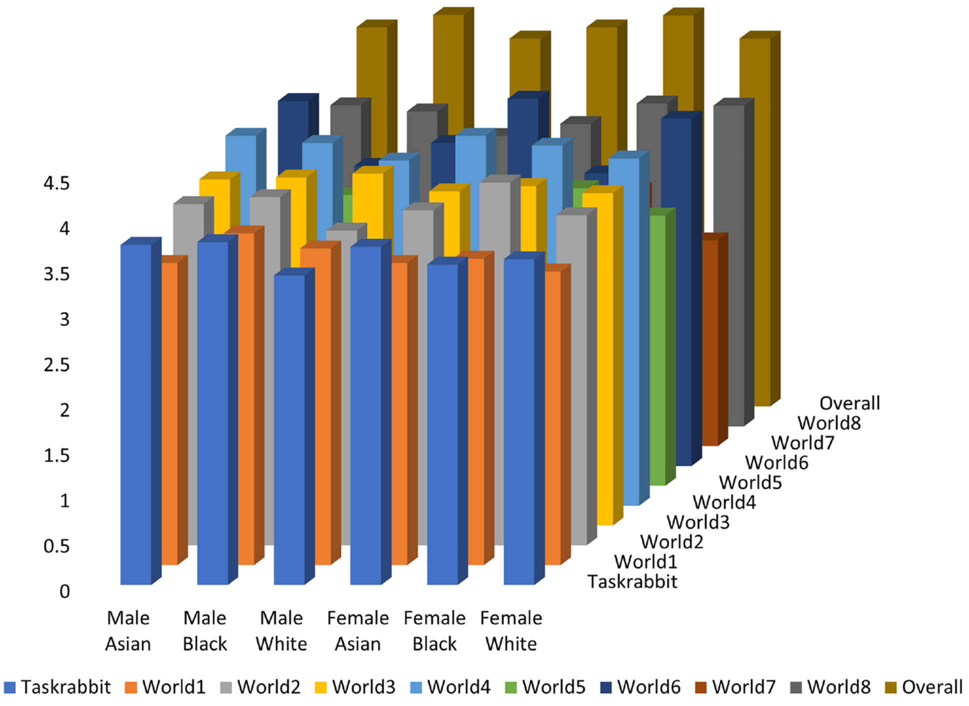


Fig. 10 Sum of fairness values for the top-5 job-platform pairs per seeker

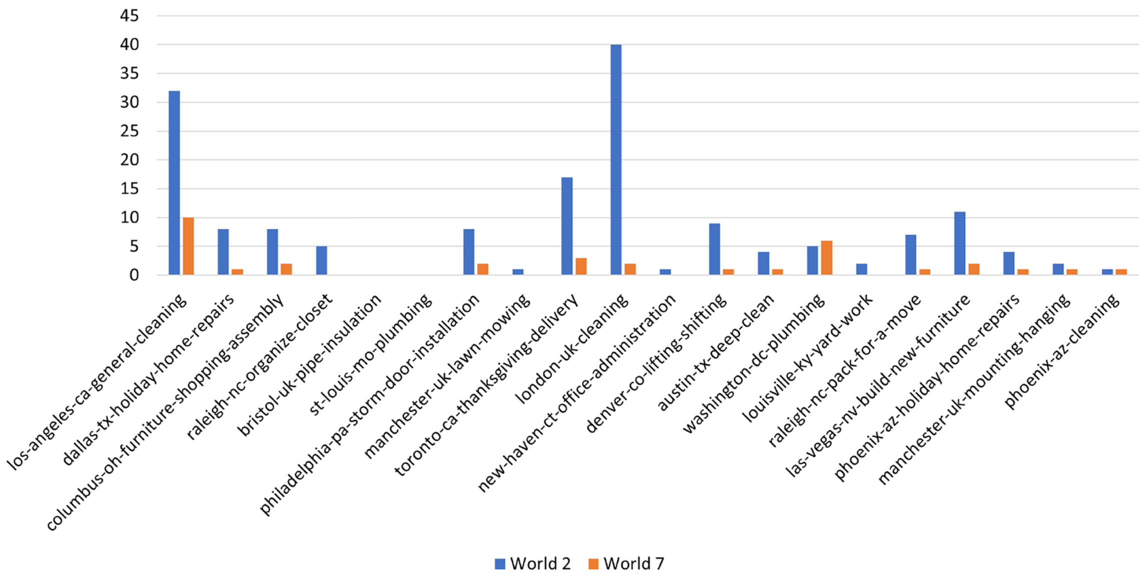


Fig. 11 Comparing worker counts for the 20 selected jobs in *world*2 vs. *world*7

GHz dual-core Intel Core i5 processor and 8 GB of RAM. When recording runtimes, real (wall-clock) time was used, since the top-k algorithm relies on disk reads and memory accesses, which should be accounted for.

In the first experiment, we built a *fully synthetic* dataset consisting of 5000 jobs and 70 platforms. Each job in each platform was represented as a file, consisting of a ranked list of workers. The number of these workers for each job-platform pair was set to a random value between 0 and 50. In addition, each worker was assigned random values for two protected attributes. Form this generated data, we then built the required inverted lists that the top-k algorithm uses. More precisely, we built an inverted list for every possible worker group, which consists of job-platform pairs along with their fairness values for the corresponding group sorted in descending order of fairness. To compute fairness values, we used the two metrics defined in [1], namely Earth Mover

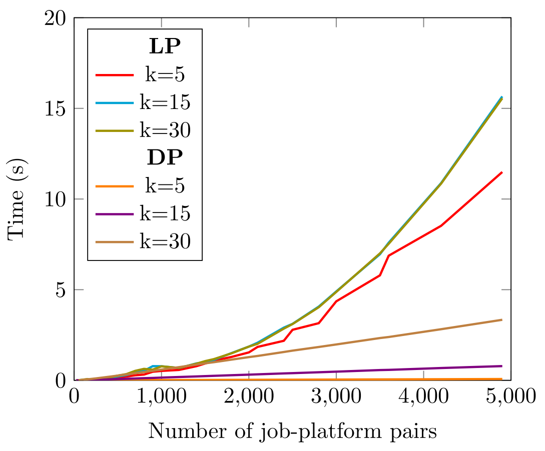


Fig. 12 Performance of the ORTools solver (LP) and the dynamic programming algorithm (DP) algorithm

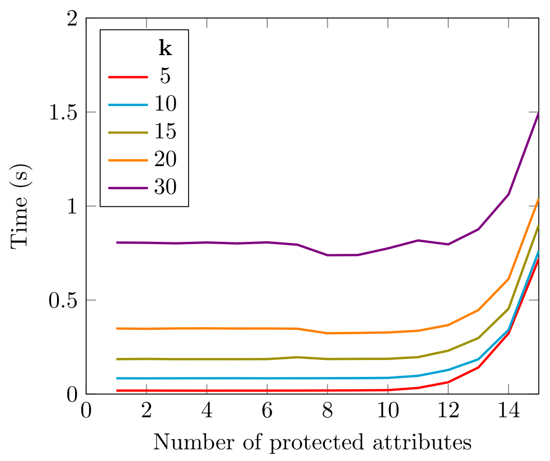


Fig. 13 DP algorithm runtimes wrt. the number of protected attributes n

Distance (EMD) and Exposure. That is, this scalability experiment was run using both metrics.

Given the above setup, we then ran both the top-k algorithm and the naive baseline algorithm that loops over all jobs, platform and groups, with increasing values of $|J|$, $|P|$ and k[3]. For each run, we generated 10 job seekers, where each seeker's protected attributes were selected at random. We assigned to each of these 10 job seekers $|J|$ jobs and $|P|$ platforms of interest at random. We then retrieved the top-k job-platform pairs for each seeker using both the naive and the top-k algorithms. Each possible $(|J|, |P|, k)$ combination was run for all seekers (one run per seeker, so a total of 10 runs per combination), and the average running time of each algorithm per combination was recorded.

After running the just-described experiment, we observed that the results for different values of k showed very similar trends. Therefore, we only focus on $k = 20$ to compare the naive and the top-k algorithms here, and the results for additional values of k are given in Appendix B. For $k = 20$, Fig. 6 shows that as the number of pairs $(N = |J| \times |P|)$ increases, the naive algorithm becomes much slower, while the top-k algorithm becomes slightly faster until its speed eventually plateaus, which indicates that the top-k algorithm scales much better than the naive one. Moreover, the figure shows that the naive algorithm performs better when using the Exposure fairness metric (Sub-figure b) compared to when using EMD (Sub-figure a), as EMD is more computationally expensive. In contrast, this behavior is not seen for the top-k algorithm, as it makes use of inverted lists to store pre-computed fairness values. This makes the top-k algorithm's runtime independent of the fairness "blackbox" used. Given this observation and for space limitation, *we only present the results using EMD as a fairness metric in the rest of this section*.

Our second scalability experiment aims to analyze how well the top-k algorithm scales as the number of protected attributes n increases. Before we describe the setup of this experiment, it is important to first distinguish between a *protected attribute* and a *group*. A worker group represents a combination of one or more protected attributes that are assigned a value, e.g., $\{gender: \}\}female''\}$. This means that, when n attributes are being considered, each worker belongs to all groups that are combinations of one or more of their protected attributes' values. For example, an asian male belongs not only to the group $\{gender: \}\}male'', ethnicity: \}\}asian''\}$, but also to $\{gender: \}\}male''\}$ and $\{ethnicity: \}\}asian''\}$. Assuming that a worker can only have one value for an attribute at a given point in time, the maximum number of groups that each worker belongs to is $2^n - 1$, which is the size of the powerset of the attributes set, minus the empty set.

To conduct our second scalability experiment, we generated another *fully synthetic* dataset, again with 5000 jobs and 70 platforms as in our first dataset. But unlike the former dataset, this one makes use of 256 synthetic groups numbered *group*1 to *group*256 in order to scale up the number of groups each seeker can belong to. This new dataset consists of a set of inverted lists, one for each worker group. Each such inverted list contains job-platform pairs with their fairness values for the group that the inverted list corresponds to, again sorted in descending order of fairness. The experiment then goes as follows for increasing values of $|J|$, $|P|$, and n, the number of protected attributes. For each $(|J|, |P|, n)$ combination, 10 job seekers are generated and assigned $|J|$ jobs and $|P|$ platforms of interest each, selected at random. The job seeker problem is then solved for each

[3] $|J|$ is the number of jobs, $|P|$ is the number of platforms, and k is the number of job-platform pairs with the maximum fairness to be returned by the algorithms.

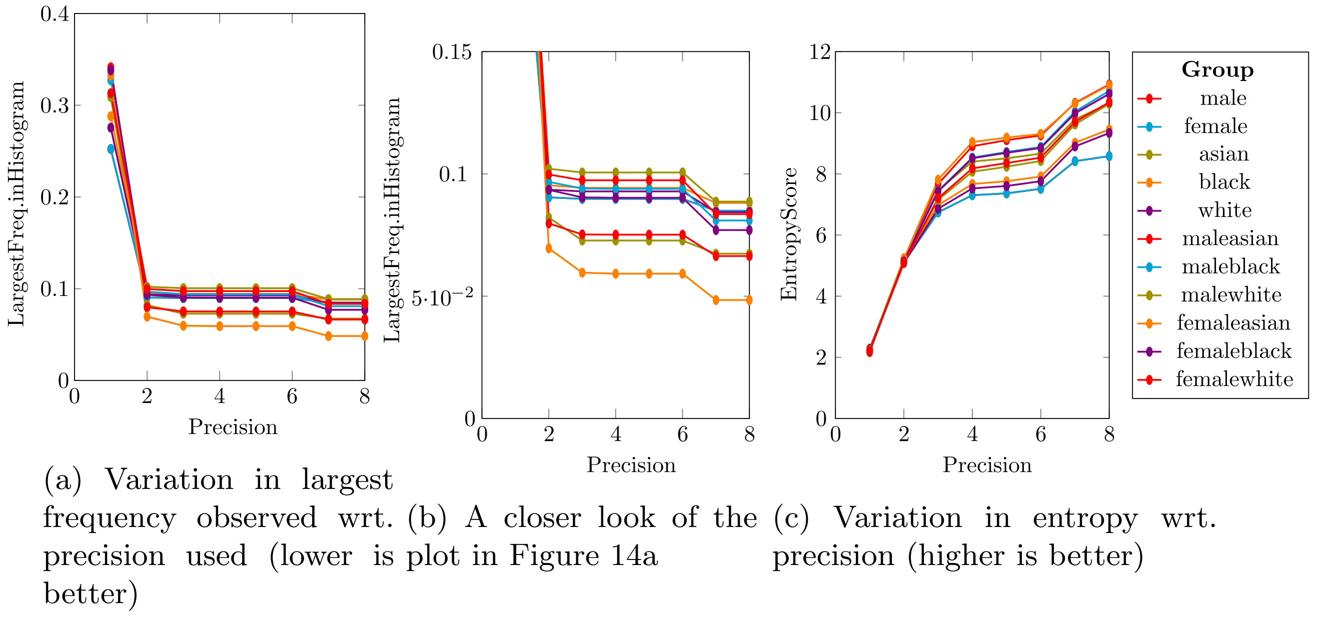


Fig. 14 Finding the optimal number of digits to map fairness values from floats to integers

seeker using the top-k algorithm, and the average solving time over the 10 runs was recorded.

The runtime of the top-k algorithm as we vary the number of protected attributes n is shown in Fig. 7. As can be seen from the figure, the runtime of the algorithm grows exponentially as the number of protected attributes increases. This is intuitive given that when we consider n protected attributes for each seeker, the top-k algorithm needs to operate on $2^n - 1$ inverted lists concurrently that correspond to $2^n - 1$ groups the job seeker belongs to, as explained just above.

4.3.2 Unconstrained Problem Qualitative Experiments

We design two experiments to qualitatively demonstrate the utility of solving our unconstrained job seeker problem. Both experiments utilize our *semi-synthetic* data that was obtained from the real-world online labor platform TaskRabbit and where nine additional worlds (i.e., platforms) were generated using interventions on protected attributes.

The first experiment focuses on the alternative worlds, and how their demographic group distributions affect the search results for seekers of different groups. To this end, we generated six seekers (one per gender/ethnicity combination), set the same $|J| = 20$ random jobs of interest to all of them, set their platforms to be the nine alternative worlds, and retrieved the top-5 fairest job-platform pairs for each seeker using our top-k algorithm. For each top-5 retrieved pairs, the number of occurrences of each platform is shown in Fig. 8, and the number of occurrences of each job in Fig. 9.

As can be seen in Fig. 8, platforms *world*2 and *world*4 are present in all of the seekers' top-5 retrieved job-platform pairs, suggesting that these worlds are fair to every group for the 20 chosen jobs of interest. On the other hand, we see that TaskRabbit and *world*7 do not occur in any of the seekers' top-5 retrieved pairs, which suggests that these platforms are the least fair for the chosen jobs. Note that while TaskRabbit and *world*4 have similar group statistics as can be seen in Table 2, these statistics for individual jobs can differ. For example, the job "Pack for a Move in Raleigh, NC" in TaskRabbit consists of 83% male and 17% female workers, while the same job in *world*4 consists of 75% male and 25% female workers. Also note that the percentage of workers from a particular group does not correlate with how fair the platform is with respect to that group. For example, *world*6 consists of 49% black males, the highest percentage of black males among all platforms (see Table 2). However, *world*6 is not considered one of the top-5 fairest platforms for the black male seeker since the ranks of black males in this platform are lower on average compared to other platforms for all considered jobs.

Finally, as can be seen in Fig. 9, the job "Cleaning in London, UK" appears in the top-5 retrieved job-platform pairs for all job seekers regardless of their groups, implying that this job is fair to all demographic groups in our study. Other frequently appearing jobs are "Furniture Shopping and Assembly in Columbus, OH," which appears in the top-5 for all groups except the black one, and "Pack for a Move in Raleigh, NC" which appears for all groups except the asian one.

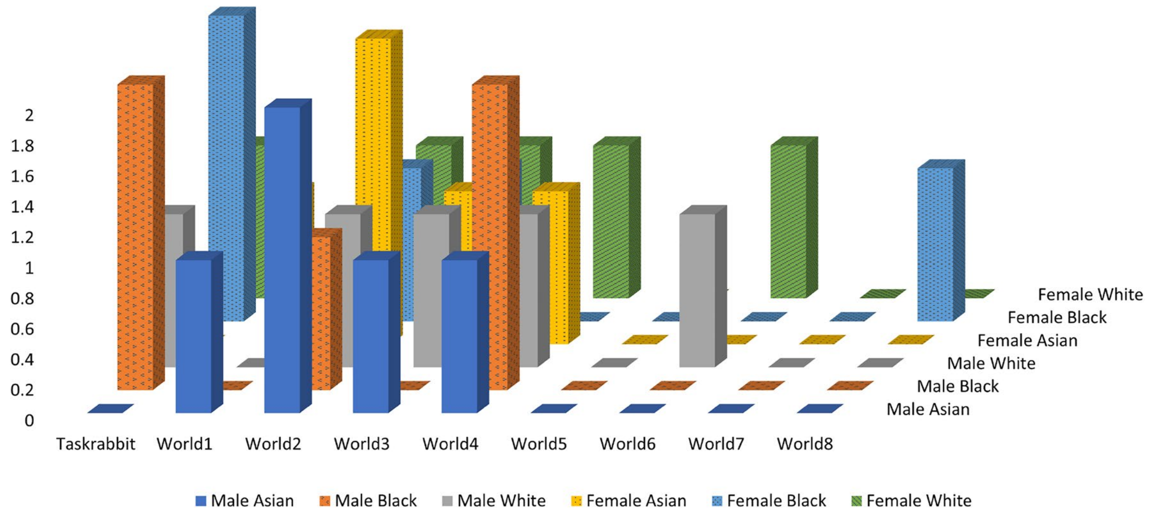


Fig. 15 Occurrences of each platform in the seekers' top-5 retrieved job-platform pairs

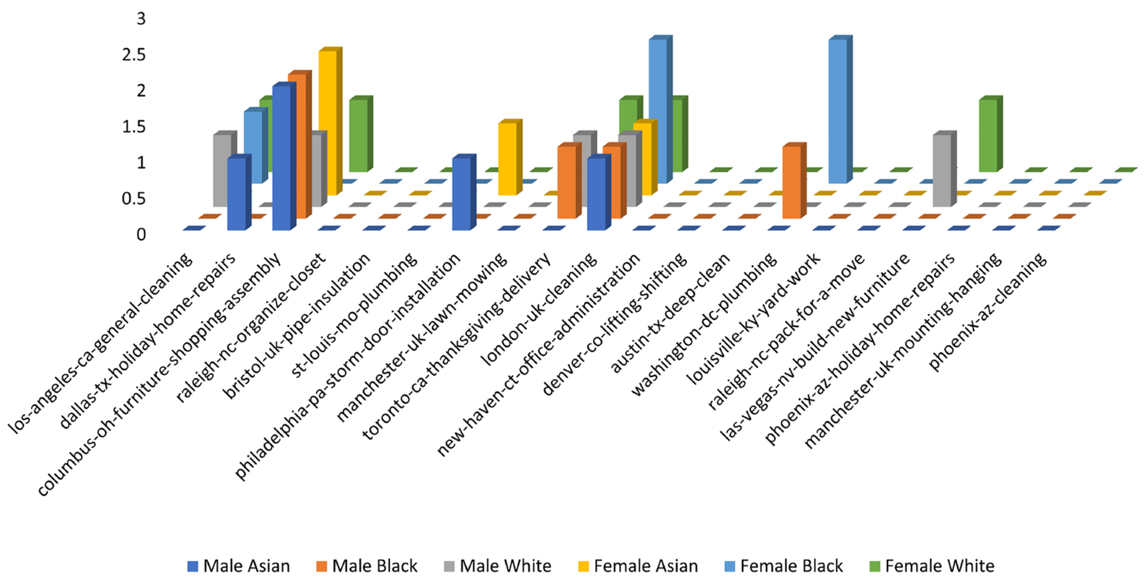


Fig. 16 Occurrences of each job in the seekers' top-5 retrieved job-platform pairs

Table 3 Sum of fairness values and rewards for the top-5 retrieved job-platform pairs

Seeker	Sum of fairness values	Sum of rewards
Male asian	43434	402
Male black	38456	400
Male white	39750	401
Female asian	43434	402
Female black	38607	402
Female white	38753	401

The second experiment investigates how the chosen worlds of preference affect a seeker's chances of finding fair jobs. For this, we fixed one random set of 20 jobs of interest, and assigned it to all six seekers. Then, for each seeker and alternative world p_i, we retrieved the seeker's top-5 fairest jobs in platform p_i. We also retrieved the seeker's overall top-5 fairest jobs considering all of the platforms (i.e., by solving our unconstrained job seeker problem). Finally, for each top-5 retrieved job-platform pairs, the sum of the pairs' fairness values was computed, which is shown in Fig. 10.

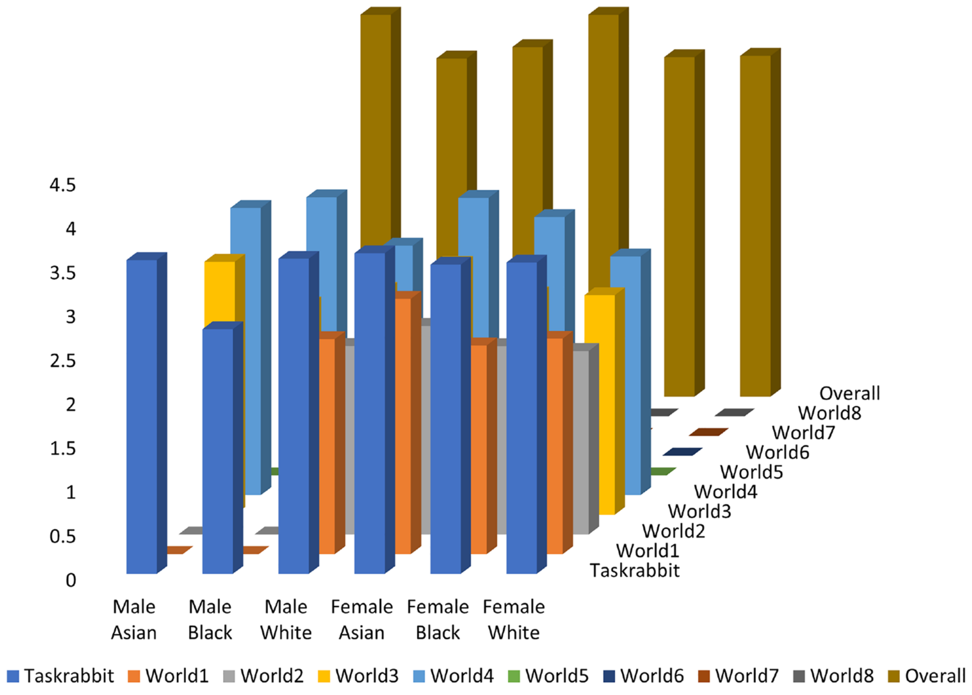


Fig. 17 Sums of fairness values for the top-5 job-platform pairs per seeker

As can be seen from the figure, solving our unconstrained job seeker problem results in the highest total fairness as compared to retrieving the top-5 fairest jobs from any of the platforms alone. This demonstrates the utility of considering multiple platforms at the same time when a job seeker is looking for a job. Moreover, the figure shows that *world*7 has the lowest sum of fairness values across all worker groups, which indicates that *world*7 is the least fair platform for the chosen set of jobs. Recall that *world*7 is sampled from the TaskRabbit platform by reversing the percentage of asian and white workers. As asians form a minority in the original TaskRabbit platform (7% of all workers), this world has by far the fewest number of workers in it, which can negatively affect fairness values.

To further understand the reason behind *world*7's relatively low fairness for the selected jobs, we compare statistics between this world and *world*2, one of the worlds that is considered the most fair across all worker groups as can be seen in Fig. 10. We first compare the number of workers in *world*2 and *world*7 for the 20 jobs shown in Fig. 11. The plot shows that the 20 jobs in *world*7 have in general very few workers compared to *world*2, with most of these jobs having fewer than 5 workers each. We also notice that many of these jobs only have workers from very few groups (especially the jobs that have very few workers). This leaves many worker groups unrepresented for these jobs, and hence there are no fairness values for those (job, world, group) combinations. As a result, these combinations cannot appear in any seeker's top retrieved job-platform pairs.

4.3.3 Constrained Problem Scalability Experiments

Similar to the case of the unconstrained job seeker problem, we again conducted two experiments to demonstrate the scalability of the DP algorithm we proposed to solve the constrained job seeker one. Again, all experiments were run on an Apple MacBook Pro with a 2.3 GHz dual-core Intel Core i5 processor and 8 GB of RAM. However, unlike the experiments for the unconstrained problem, we used CPU time here to record runtimes.

Our first experiment here aims to study the scalability of the DP algorithm (Algorithm 2) we proposed to solve the constrained job seeker problem. For this experiment, the number of protected attributes considered is fixed to 2 (so $n = 2$), meaning each seeker belongs to $|G| = 2^2 - 1 = 3$ groups. For increasing values of N and k, we simulate problem instances as follows. First, we generate N synthetic job-platform pairs, each associated with a set of $|G|$ fairness values, one for each of the seeker's groups (so three integers selected at random between 1000 and 9999; integers since the DP algorithm operates on integer fairness values) and an integer reward value, randomly selected between 10 and 99. From there, the problem's objective is to find the top-k job-platform pairs that maximize fairness while satisfying a reward threshold of $80 \times k$. The threshold was set as a function of k as it is intuitive that the higher the number of the job-platform retrieved is, the higher the reward threshold of the job seeker is expected to be.

For each (N, k) combination considered, we randomly generated 10 problem instances as described above, and then solved each instance in two ways: 1) using our DP

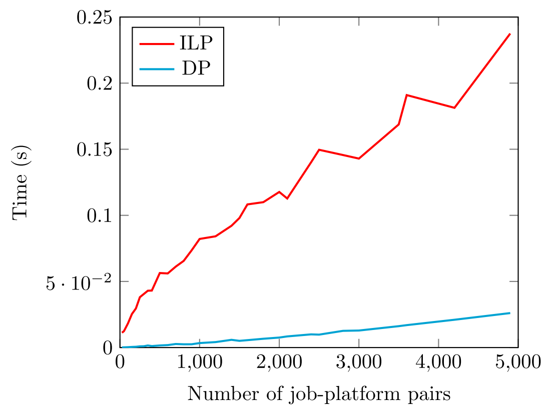


Fig. 18 Runtimes of the DP algorithm vs. the ILP solver wrt. number of job-platform pairs N

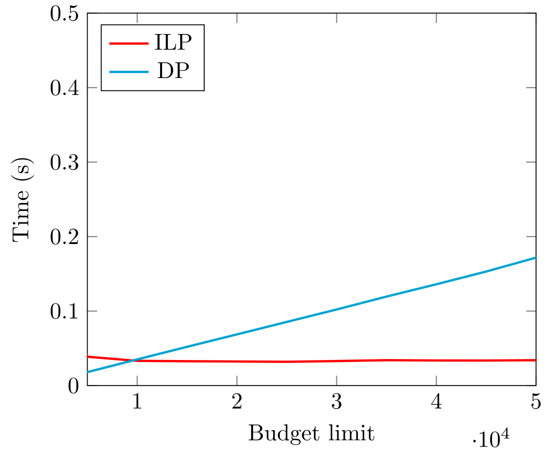


Fig. 19 Runtimes of the DP algorithm vs. ILP solver wrt. budget limit B

algorithm, and 2) an off-the-shelf integer linear programming (ILP) solver (Google's ORTools[4]). We then recorded the average runtimes of each method over the 10 runs as N increases, which are shown in Fig. 12. As can be seen in the figure, the proposed DP algorithm solves the constrained job seeker problem much faster than the general-purpose ILP solver, for all values of k. Both algorithm's runtimes seem to increase as N increases, but this observed increase for DP is less pronounced and much more linear than for ILP. This suggests that the proposed DP algorithm scales much better than ILP in terms of N. With respect to k, we observe that the DP algorithm's runtime also increases with k, but the ILP's seems to remain mostly unchanged as k varies, suggesting that the ILP's running time does not depend much on k.

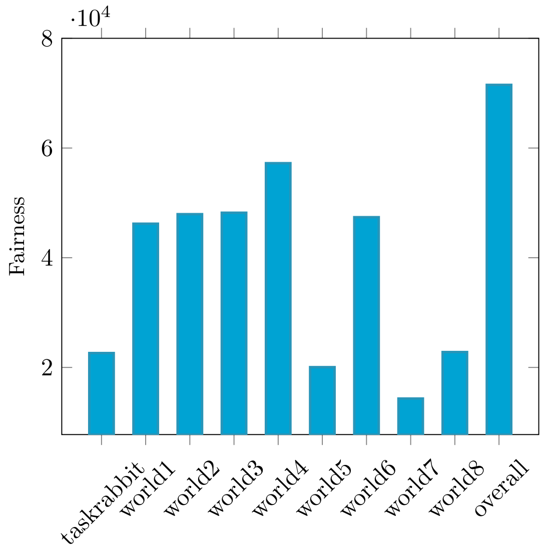


Fig. 20 Sum of fairness values for the selected job-platform pairs

Next, we examine how the DP algorithm performs as the number of protected attributes n increases. For this, we repeat the experiment above, but instead of setting $n = 2$ protected attributes, we run the experiment for increasing values of n. The results are shown in Fig. 13. We can see that up until $n = 11$, the DP algorithm's runtime does not change much, but then grows exponentially after that point. Remember that the DP algorithm consists of two main stages: a "preprocessing" stage where the minimum fairness of each job-platform pair is computed, with time complexity $O(|J||P||G|)$, followed by a solving phase using dynamic programming, with complexity $O(|J||P|kR)$. As can be observed from Fig. 13, the point where the runtime of the algorithm starts to increase exponentially is the point where the value of $|J||P||G|$ becomes as significant (same order of magnitude) as $|J||P|kR$. From there, we conclude that as long as the number of groups $|G| = 2^n - 1$ is of smaller order of magnitude than kR, the DP algorithm's runtime will not depend much on n.

4.3.4 Constrained Problem Qualitative Experiments

We conducted two qualitative experiments to demonstrate the utility of solving our constrained job seeker problem. However, as the DP algorithm expects fairness values to be integers, and our fairness blackbox provides fairness values as floats between 0 and 1, we first had to convert the fairness values to integers before invoking the algorithm. Note that we did not need to do that for the scalability experiments, since these experiments were conducted on fully synthetic data and thus fairness values were generated as integers rather than floats. To convert fairness values from floats to integers for the semi-synthetic data the qualitative

[4] https://developers.google.com/optimization.

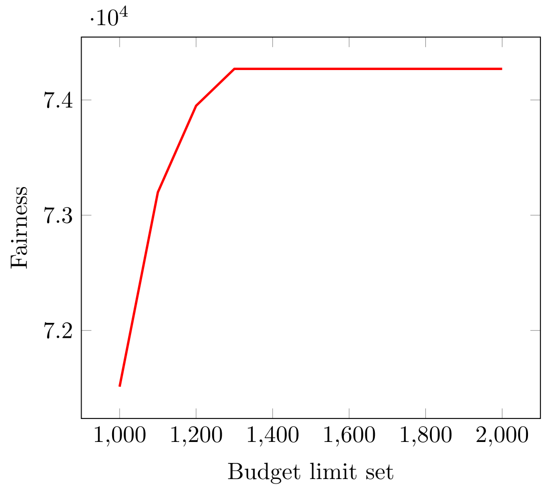


Fig. 21 Sum of fairness values of the selected job-platform pairs vs. budget limit

experiments are conducted on, we resorted to truncating each fairness value to d significant digits, and then multiplying the result by 10^d. For example, if $d = 2$, then a fairness value of 0.831 will be mapped to the integer 83, and the range of possible integer values will be between 0 and 99.

However, we need to ensure that d is large enough to avoid mapping too many fairness values to the same integer, yet small enough that the fairness integers are not too large or too granular. An optimal value of d would thus be the smallest value that gives us enough precision when truncating the fairness values, so as to avoid too many collisions when mapping to integers. To find the optimal d, we considered integers from 1 to 8 as candidate values. For each candidate value of d, we took all fairness values in our semi-synthetic dataset, and mapped them to d-digit integers. We then binned the resulting values in a histogram, where the bins are $\{0, 1, 2, ..., 10^d - 1\}$, so that we get for each possible integer value, the frequency of fairness values that were actually mapped to it.

Next, we recorded 1) the largest frequency observed (in percentage), which gives us the size of the largest collision in the histogram; and 2) the entropy of the obtained fairness values, which we used as an indicator of how well-distributed (and not biased toward certain values) the mappings are. Comparing these metrics between candidate values of d shows us how much "improvement" (fewer collisions), there is going from one precision d to the next. The observed values are shown in Fig. 14a and c.

As can be seen in Fig. 14a (with a zoomed in view in Fig. 14b), the size of the largest collision declines significantly from $d = 1$ to $d = 2$, followed by a slower decline at $d = 3$, before stagnating mostly between $d = 4$ and $d = 6$. We then see another marginal decline at $d = 7$. This means that the biggest precision gains lie between $d = 1$ and $d = 3$, with a relative gain starting from $d = 7$ onwards.

In addition, Fig. 14c reveals that the increase in entropy is most noticeable from $d = 1$ till $d = 4$, with much slower increases from there till $d = 6$, followed by a further increase at $d = 7$. As entropy is a good indicator of the spread and variety of the obtained fairness values, we can then conclude that the most impactful decreases in collisions occur between $d = 1$ and $d = 4$, with other relative improvements seen from $d = 7$ and on. Therefore, we conclude from the three figures that $d = 4$ is a reasonable precision to use.

We now describe our two qualitative experiments for the constrained job seeker problem. Both experiments use the exact same setting as the one conducted for the unconstrained job seeker problem. That is, we used the same six job seekers (one per gender/ethnicity combination) and assigned to them the same $|J| = 20$ random jobs of interest, and set the nine alternative worlds as platforms of interest P. As the constrained job seeker problem also assumes that each job-platform pair is associated with a reward value, we assigned each job-platform pair in $J \times P$ with a random reward value between 1 and 100. We then retrieved the top-5 fairest job-platform pairs with a total reward of at least 400 for each seeker using our DP algorithm. The goal of the two experiments is to confirm that our reward constraint is actually affecting the obtained top-k retrieved job-platform pairs, which in turn demonstrates the need for the constrained job seeker problem formulation, compared to the unconstrained one.

In the first experiment, we recorded the number of times each world (i.e., platform) and job occurs in every seeker's top-5 job-platform pairs retrieved by the DP algorithm. Figure 15 shows the number of occurrences of the different worlds in each of the top-5 retrieved job-platform pairs per job seeker . Similarly, Fig. 16 shows the number of occurrences of the different jobs in each of the top-5 retrieved job-platform pairs per job seeker. In addition, Table 3 shows the sum of (four-digit) fairness values and the sum of rewards for each top-5 retrieved job-platform pairs.

As can be seen in Figs. 15 and 16, the results are different from those of the unconstrained job seeker problem experiment (Sect. 4.3.2), even though both experiments share the exact same setting apart from the reward threshold. This indicates that the reward threshold is actively affecting the choice of which job-platform pairs are being retrieved by the DP algorithm. Table 3, on the other hand, shows that the reward constraint is indeed met for every retrieved top-5 job-platform pairs as they all have a reward of at least 400. In addition, each of the top-5 retrieved pairs have a satisfactory total fairness as can be seen from the first column of Table 3. That is, in our experiment, the maximum total fairness achievable is 50,000 (i.e., 5.0 without the integer conversion, since we aim to select five job- platform pairs

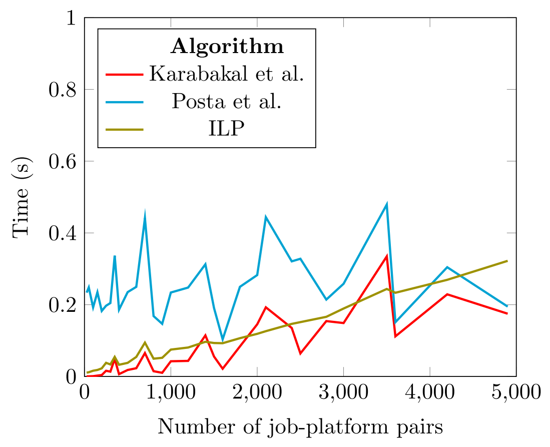


Fig. 22 Exact algorithms' runtimes wrt. N (number of job-platform pairs)

and the maximum fairness for each pair is 1). Looking at the obtained total fairness for each seeker in Table 3, they are all around 40,000, so around the higher end of the [0, 50000] scale.

For the second experiment, we again used the same sets of seekers, jobs and platforms as we did in the first experiment. However, in this experiment, we retrieved for each job seeker the top-5 jobs in platform $p_i \in P$ that maximize fairness, while satisfying the reward constraint (of at least 400). We also retrieved each seeker's top-5 job-platform pairs using all platform P (i.e., by solving our constrained job seeker problem). Figure 17 shows the sum of fairness values for each top-5 retrieved pairs, when each platform is considered separately and when all of them are considered together. As can be seen from the figure, solving our constrained job seeker problem results in the highest total fairness as compared to retrieving the top-5 fairest jobs from any of the platforms alone. This demonstrates the utility of considering multiple platforms at the same time when a job seeker is looking for a job, even with presence of a reward constraint. Comparing the results to the ones of the corresponding unconstrained run, we note here again that the results of the two experiments differ, which further confirms that the reward constraint is taking effect as expected.

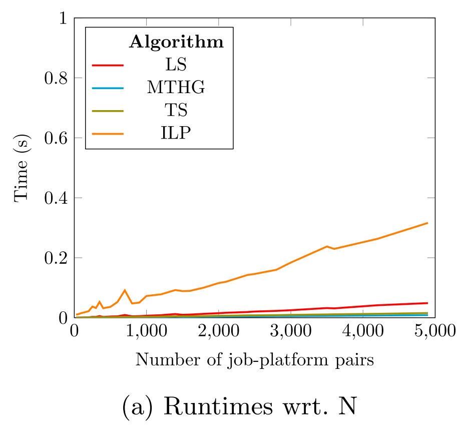
4.4 Job Provider Experiments

4.4.1 Global Budget Problem Scalability Experiments

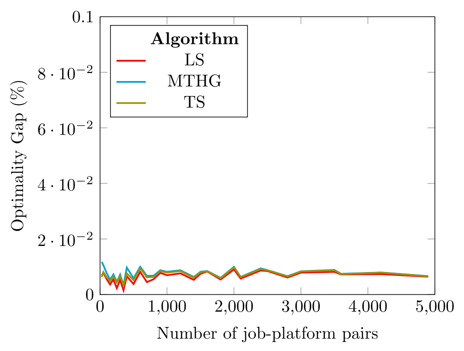
We conducted two experiments to assess the scalability of the DP algorithm (Algorithm 4) we proposed to solve the job provider problem with global budget. Both experiments were run on the same computer, an Apple MacBook Pro with a 2.3 GHz dual-core Intel Core i5 processor. Runtimes were measured in CPU time.

In the first experiment, we generated 100 *fully synthetic* instances of the problem, assuming a fixed number of protected attributes $n = 2$ for workers in all of them. Each problem instance consisted of different $N = |P| \times |J|$ job-platform pairs. Each such pair was associated with $2^n - 1 = 3$ fairness values, corresponding to the different worker groups. These values were all set at random between 1000 and 9999. Moreover, as the job provider problems assume that each job is associated with a cost for each platform it is available on, all costs for all job-platform pairs in all problem instances were set at random between 50 and 150. The goal of the experiment was then to assign each job to at most one platform so that their total fairness is maximized, while respecting a budget limit of $50 \times |J|$. We set the budget limit as a function of the jobs as it is intuitive to assume that the more jobs the job provider aims to deploy, the more budget she will be willing to spend to deploy these jobs.

Each problem instance was then solved using our DP algorithm and the Google ORTools ILP solver. Figure 18 shows the average runtime of each method over the 100 instances as the number of job-platform pairs N increases. As can be seen in the figure, the DP is significantly faster than the ILP solver for all values of N, and the DP



(a) Runtimes wrt. N



(b) Optimality gaps wrt. N (where 1 = 100%)

Fig. 23 Heuristic algorithms' runtimes and optimality gap wrt. N (number of pairs)

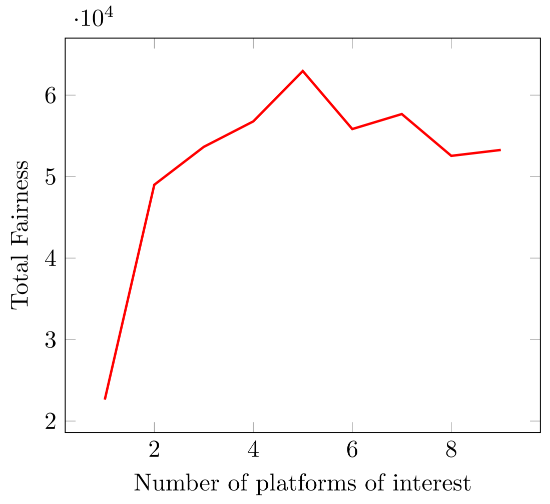


Fig. 24 Total fairness obtained vs. number of platforms of interest

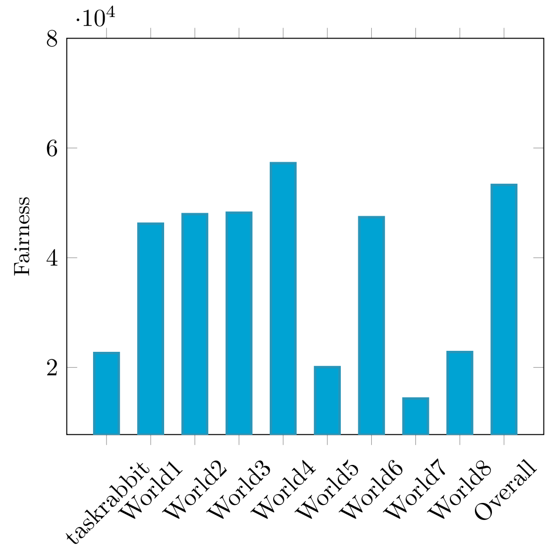


Fig. 25 Total fairness obtained per platform(s) of interest

algorithm's runtime increases much slower than the ILP one as N increases.

Next, as the budget limit B is part of the DP algorithm's time complexity as explained in Sect. 3, we designed a second scalability experiment to see how the runtime of the algorithm is affected by the value of B. To this end, we fixed the number of jobs and of platforms to $|J| = |P| = 50$, and generated 100 instances of the problem with random fairness and cost values for each job-platform pair. Each problem instance was then solved with different values of B, by both the DP algorithm and the ILP solver. Figure 19 shows the average runtimes of the two methods as B increases. As can be seen from the figure, the runtime of the DP algorithm increases linearly as B increases, whereas the runtime of the ILP solver is hardly affected by the budget limit B.

4.4.2 Global Budget Problem Qualitative Experiments

We conducted two experiments to demonstrate the utility of solving our job provider problem with global budget using the same semi-synthetic dataset we used in the qualitative experiments of the job seeker problems. The first experiment aimed to study to what extent the platforms of interest of a job provider affect the fairness of the assigned jobs on these platforms. To do this, we created one job provider, and fixed her jobs of interest to 20 random jobs. Each of the 20 jobs were assigned a random cost between 50 and 150 for each platform out of the nine we have in our dataset, and on which the job is available. We then solved the job provider with global budget problem assuming a budget limit of 1000 using our proposed DP algorithm. In addition, for each platform p_i, we selected the jobs that maximize sum of their fairness and where their total cost on the platform p_i does not exceed the budget limit of 1000.

Figure 20 shows the sum of the fairness values of the selected jobs per platform as well as the sum of the fairness values of the job-platform pairs returned by solving our overall optimization problem. Note that unlike the job seeker problems, the plot here is two-dimensional, since the "group" dimension is not relevant for the job provider problems. From this figure, we can make two observations. First, and as is the case in the job seeker qualitative experiments, not all platforms achieve comparable fairness values. That is, for the same jobs and same budget limit, some worlds like *world*4 yield noticeably better fairness values than other worlds, while others like *world*7 achieve lower fairness values. Second, solving our job provider problem with global budget using all platforms (indicated by "overall" in Fig. 20) results in the maximum total fairness for the selected job-platform pairs than when selecting the jobs that maximize fairness from each platform individually. This highlights the importance of solving our job provider problem with global budget.

Our second qualitative experiment aimed to answer the question: *does a higher budget limit necessarily imply selecting more fair job-platform pairs?* To do this, we again created one job provider, with the same 20 jobs of interest as the previous experiment, and we assumed the provider's platforms of interest to be all nine alternative worlds. We then solved our job provider problem with global budget, varying the budget limit between 1000 and 2000. Figure 21 shows the sum of fairness values of the selected job-platform pairs. As shown in the figure, the obtained sum of fairness values increases slightly at first as the budget limit becomes more permissive, before eventually plateauing when the budget limit reaches 1300. This happens because, in this particular problem instance, the jobs-to-platforms assignment with the maximum fairness possible has a cost of 1204. Thus, increasing the budget limit beyond 1204 cannot produce any better results fairness-wise, and will return this same optimum solution.

4.4.3 Local Budget Problem Scalability Experiments

Our scalability experiment for the job provider problem with local budget aimed to compare a selection of both exact and heuristic algorithms to solve the generalized assignment problem (GAP), which we have shown in Sect. 3 to be equivalent to the job provider problem with local budget, and hence can be used to solve it as well. To this end, we generated 100 instances of the problem using fully synthetic data, assuming a fixed number of protected attributes $n = 2$ for workers in all of them. Each problem instance consisted of different $N = |P| \times |J|$ job-platform pairs. Each such pair was associated with $2^n - 1 = 3$ fairness values, corresponding to the different worker groups. These values were all set at random between 1000 and 9999. Moreover, as the job provider problems assume that each job is associated with a cost for each platform it is available on, all costs for all job-platform pairs in all problem instances were set at random between 50 and 150. Finally, since the local budget problem assumes the presence of a local budget b_p for each platform of interest $p \in P$, this local budget limit was set as follows:

$$b_{\mathrm{p}} = \left\lceil \frac{100 \times |J|}{|P|} \right\rceil + \epsilon$$

and where ϵ is a random integer between 0 and 49.

The point of the above formula is to roughly even out the budget limits across platforms, while still having some fluctuation in the b_p values. That is, we start with a total budget limit of $100 \times |J|$ (so double that of the global budget variant), which will now be divided over the platforms. Hence, each platform will have $\frac{100\times|J|}{|P|}$ budget, plus a random value ϵ to add some variations in the budget limits. From there, the goal of the experiment is then to solve these problem instances using each of the algorithms considered, as well as a generic ILP solver (ORTools). This experiment was also run on an Apple MacBook Pro with a 2.3 GHz dual-core Intel Core i5 processor. Runtimes were measured in CPU time.

We explored the following three exact algorithms for solving our job provider problem with local budget:

- The BB algorithm by Fisher et al. [11], following the pseudo-code in [21];
- The BB algorithm by Karabakal et al. After parameter tuning, we set the root subgradient iteration limit ("ROOTSUBITLIM") to 200, the subgradient limit at other nodes to 100 and the maximum branching limit to 200,000, with all other parameters being kept at their default values.
- The BB algorithm by Posta et al. After parameter tuning, we set the subgradient iteration limit to 30, the root bundle iteration count to 25000 and kept all other parameters at their default values.

For the three algorithms, the experiment was run on the same 100 problem instances we described above. However, the Fisher et al. algorithm was extremely slow and did not terminate for many instances despite our best efforts at optimizing its code. Thus, this algorithm was not included in the rest of the experiment. The runtimes of the rest of the exact algorithms as well as an ILP baseline solver (ORTools) are shown in Fig. 22. As can be seen from the figure, both Karabakal et al.'s and Posta et al.'s algorithms scale fairly well as the number of job-platform N increases, with Karabakal et al.'s method having a slight edge. However, neither of the two algorithms was significantly faster than the ILP solver, as can be seen from Fig. 22.

We also explored the following three heuristic algorithms to solve our problem:

- MTHG
- Osman's LS Descent method (LS) without long-term procedure
- Osman's Tabu Search method (TS) without long-term procedure

For the Tabu Search method, the *MAXI* and tabu list size parameters were kept as in [25]; and all other parameters for all the algorithms were kept at their default values. The three algorithms, as well as an ILP baseline solver (ORTools), were then run on the same 100 problem instances we generated. Since the three algorithms are heuristic (i.e., approximation) algorithms, they might not return the optimal solution to our optimization problem. Thus, we also evaluated the solution quality of each algorithm, which is computed as the gap between the sum of fairness values for the job-platform pairs selected by a heuristic algorithm z_h, and the one for the pairs selected by ORTools, z_{opt} (which is an exact algorithm, hence providing an optimal solution). More precisely, this solution quality was computed as follows:

$$\mathrm{alignedoptimality\_gap_h} = \begin{cases} \frac{z_{\mathrm{opt}} - z_{\mathrm{h}}}{z_{\mathrm{opt}}} & amp; \text{if } z_{opt} \neq 0 \\ 0 & amp; \text{otherwise.} \end{cases}$$

Figure 23a shows the runtimes of the different heuristic algorithms vs the ILP solver, while Fig. 23b shows the optimality gap for each algorithm as the number of job-platform pairs N increases. As can be seen from the figure, all three heuristic algorithms are significantly faster than the ILP solver, while returning relatively good solutions within 2% from optimality on average. Therefore, if an exact solution is not a must, then heuristics can be a good more efficient alternative to exact algorithms for solving the job provider problem with local budget.

4.4.4 Local Budget Qualitative Experiments

We designed two experiments to demonstrate the utility of solving our job provider problem with local budget. The first one aimed to study the trade-off between the number of platforms of interest for a job provider versus the local budget limits for these platforms with respect to the total fairness obtained by solving our problem. To this end, we again relied on our semi-synthetic data as we did in all previous qualitative experiments. That is, we created a job provider and then used the same setting as in the first qualitative experiment for the job provider with global budget, which is described in Sect. 4.4.2. More precisely, we assumed all nine alternative worlds to be potential platforms of interests, and we assigned the job provider the same 20 random jobs of interest with their fairness values on different platforms also generated at random as described in that experiment. We then assumed that the job provider has a limited *total* budget she cannot exceed when deploying the jobs on the platforms, which we set to 1000 like in the first experiment in Sect. 4.4.2. We then varied the number of platforms $|P|$, and divided the budget limit evenly across them (if the total limit is not divisible by $|P|$, the remainder amount is added to the last platform). We ran the just-described experiment 9 times. In the first run, we set $P = \{taskrabbit\}$ as the platform of interest, in the second run $P = \{taskrabbit, world1\}$, in the third, $P = \{taskrabbit, world1, world2\}$, etc. In each run, the job provider with local budget problem was solved using the Karabakal et al. algorithm [18], and the total fairness of the optimal jop-platform assignment was recorded.

Figure 24 shows the total fairness obtained when solving our problem as the number of platforms increases. As can be seen in the figure, there is an apparent trade-off between total fairness and number of platforms. At first, total fairness generally increases, as we get more options (job-platform pairs) to select from. However, as we increase the number of platforms further, the budget limits keep getting tighter on each platform, and so we start seeing a decrease in the total fairness obtained.

Our second qualitative experiment aimed to study how the choice of platforms of interest affects the obtained total fairness. To do this, we used the same experiment setup as the first one, but this time for each run, only one platform is considered as a platform of interest rather than using subsets of them as in the first experiment. That is, for the first run, $P = \{taskrabbit\}$, for the second run, $P = \{world1\}$, the third, $P = \{world2\}$, etc., plus one final run where all the platforms are considered platforms of interest (i.e., thus solving our job provider problem with local budget). We assumed the same total budget of 1000 for the job provider in all of the runs. In the case of a single platform of interest, that total budget was then used to select the jobs that maximize the sum of their fairness values on that platform, while their total cost is within the budget limit. For the run that utilizes all platforms, the total budget was distributed across the different platforms using the same strategy as in the first experiment.

Figure 25 shows the sum of fairness values for the selected job-platform pairs, when using each platform individually and when using all of them (indicated by "overall" in the figure). As can be seen from the figure, the total fairness obtained vary between platforms, and using all platforms together does not actually achieve the maximum fairness possible, but very close to it. This is intuitive given that the job provider problem with local budget imposes additional constraints on the job-to-platform assignment, based on the available budget for each platform. This again shows a clear trade-off between local budget limits and fairness of the jobs on these platforms.

Moreover, when comparing these results to those of the equivalent experiment for the global budget problem (Sect. 4.4.2, which is shown in Fig. 20), we see that they are all identical, except for the last run (where all platforms are considered as platforms of interest). This is again very intuitive as the job provider problems with local budget and global budget are identical when considering only one platform at a time. Finally, when considering all platforms, the total fairness obtained in the local budget experiment is lower than that obtained in the global budget experiment. Again, this is attributed to the fact that the problem with local budget imposes tighter constraints on the selection of jobs in each platform, compared to the global budget one. That is, while the *total* budget of 1000 is the same in both experiments, the local budget variant has additional constraints on how costs should be distributed over all platforms.

5 Conclusion and Future Work

In this paper, we proposed a framework to assess and compare worker group fairness for multiple jobs on multiple online labor platforms. We based our framework on realistic use cases for both job seekers and job providers, which we formulated as four optimization problems. We also proved that three of these problems are computationally hard. As shown by our experiments, the algorithms we proposed for all four problems are efficient, and answer useful fairness-related inquiries.

In our experiments, we used two different notions of group fairness, but our framework is able to accommodate other notions of fairness as long as they rely on ranking or scoring of workers with respect to jobs, including individual fairness, which is the subject of our future work. Other possible future directions include using our framework to conduct real-world case studies, where real jobs and platforms are examined from a fairness standpoint. Also, it would be interesting to adapt our framework to handle fairness issues other than ranking, such as bias in worker ratings and evaluations and to deploy our framework as a stand-alone service on top of existing online labor platforms.

Appendix A Proof of Theorem 1

Theorem 3 1 *The optimization variant of the knapsack problem is polynomial-time reducible to the constrained job seeker problem, and therefore, the latter problem is at least as hard as the former.*

Proof Note that by having only one group and one platform, the constrained job seeker problem reduces to the following: Given a list M of pairs $m_i = (f_i, r_i)$, where f_i is the assigned fairness value and r_i the reward value, select k pairs such that fairness is maximized and the total reward is at least R. Using this version of the problem, we give a polynomial-time reduction from the optimization version of Knapsack. Given a list L of pairs $a_i = (v_i, w_i)$, where v_i represents the value of the pair and w_i its weight, and an integer W, the Knapsack problem asks for a subset of L of maximum value such that the total weight is at most W.

Given an instance of the knapsack problem where $|L| = n$, create a list M of n pairs $m_i = (f_i, r_i)$ where $f_i = v_i$ and $r_i = W - w_i$. Moreover, add n additional pairs $(0, W)$ to M. Set $k = n$ and $R = (n-1)W$. We now prove equivalence of both instances. In other words, we prove that L contains a subset of total value X, satisfying the Knapsack constraints, if and only if M contains a subset of size n with total fairness X, satisfying the constrained job seeker problem constraints.

Assume L contains a subset A of size s ($s \le n$) of total value X and total weight $W_A \le W$. Construct a subset B of size $n = k$ of M by taking $\forall p_i \in A$ its equivalent $m_i \in M$, and finally add $n - s \le n$ pairs of the form $(0, W)$. Let F_B denote the total fairness of B and R_B its total reward.

$$F_B = \sum_{m_i \in B} f_i = \sum_{p_i \in A} v_i + (n-s) \times 0 = X$$

$$R_B = \sum_{m_i \in B} r_i = sW - \sum_{p_i \in A} w_i + (n-s)W$$

Therefore,

$$R_B = nW - W_A \ge nW - W = (n-1)W = R$$

Assume now that M has a subset B of size $k = n$ of total fairness X and total reward $R_B \ge R$. Let s denote the number of pairs $(0, W)$ in B. By removing those s elements from B, we get a new set B' consisting of elements originating from pairs in L, of total fairness X (since all removed pairs had $f = 0$) and total reward $R_{B'} = R_B - sW \ge (n-s-1)W$. Construct the set $A = \{p_i : m_i \in B'\} \subseteq L$. Let V_A denote the total value of A and W_A its total weight.

$$V_A = \sum_{p_i \in A} v_i = \sum_{m_i \in B'} f_i = X$$

$$R_{B'} = \sum_{m_i \in B'} r_i = (n-s)W - \sum_{p_i \in A} w_i \ge (n-s)W - W$$

Therefore, $\sum_{p_i \in A} w_i = W_A \le W$. □

Appendix B Additional Results

Figure 26 shows the runtimes of the naive algorithm vs. the top-k algorithm (Algorithm 1) when solving the unconstrained job seeker problem using both fairness metrics EMD and Exposure. Figure 27 shows the runtimes of the ORTools solver vs. the DP algorithm (Algorithm 2) when solving the constrained job seeker problem using the EMD fairness metric.

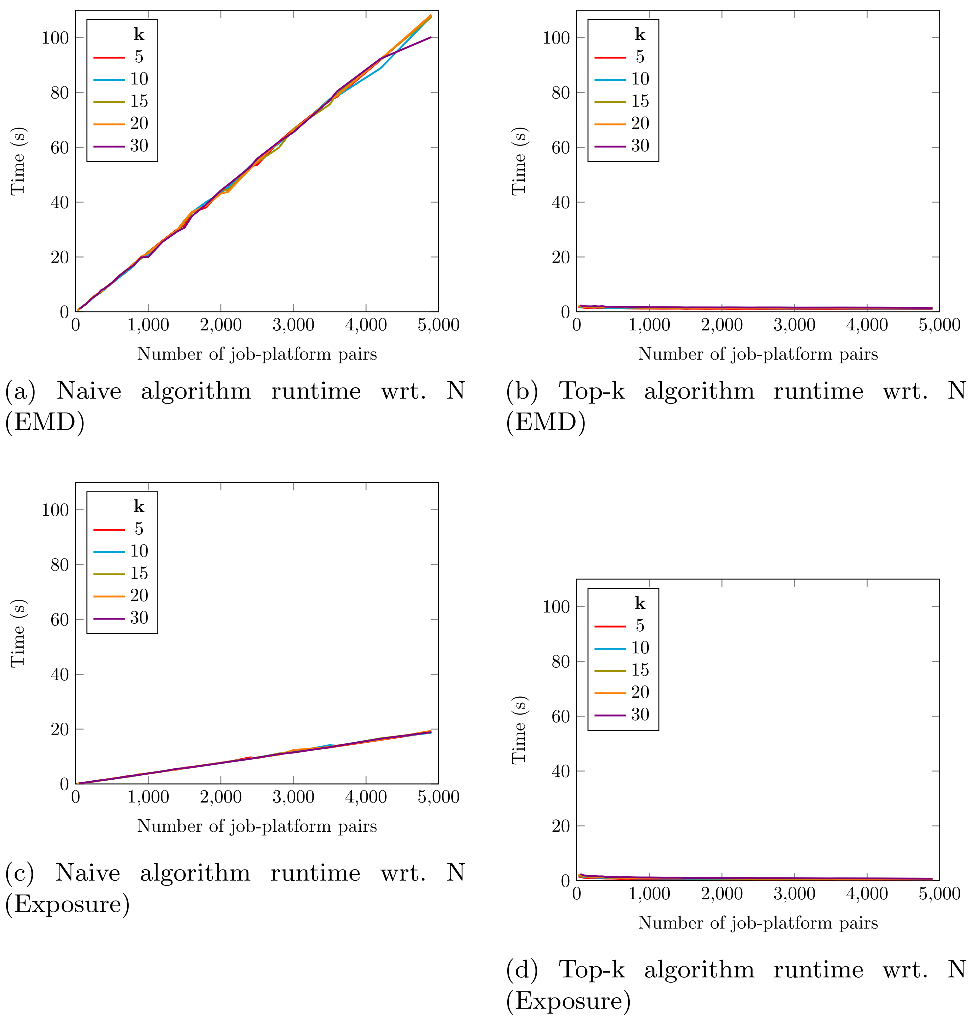


Fig. 26 Unconstrained job seeker problem: naive algorithm vs. top-k algorithm runtimes for different values of *k*

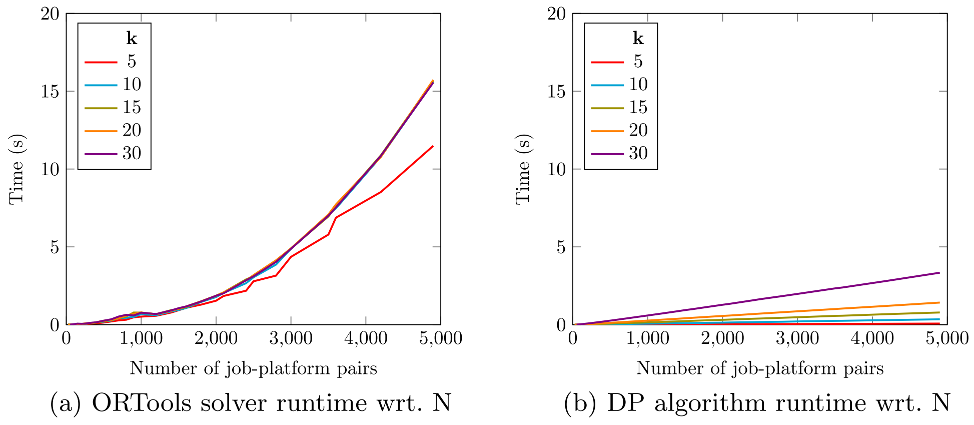


Fig. 27 Constrained job seeker problem: ORTools solver vs. DP algorithm runtimes for different values of *k*

Acknowledgements This work is supported by the Ford Foundation and the American University of Beirut Research Board (URB).

Author contributions Authors 1, 3 and 5 were students who helped with developing the algorithms and the code, preparing the data and running experiments. Authors 1 and 3 also helped with the drafting of the manuscript. Authors 2, 4 and 6 are senior authors who drove the work and the writing of the manuscript.

Funding This work is supported by the Ford Foundation and the American University of Beirut Research Board (URB).

Data availability Data will be made available upon request for research purposes.

Code availability Code will be made available upon request for research purposes.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Consent to participate N/A.

Consent for publication N/A.

Ethics approval N/A.

References

1. Amer-Yahia S, Elbassuoni S, Ghizzawi A, et al (2020) Fairness in online jobs:{A} case study on taskrabbit and google. In: Int Conf Ext Database Technol (EDBT)
2. Beutel A, Chen J, Doshi T, et al (2019) Fairness in recommendation ranking through pairwise comparisons. In: Proc 25th ACM SIGKDD Int Conf Knowl Discov & Data Min, pp 2212–2220
3. Biega AJ, Gummadi KP, Weikum G (2018) Equity of attention: amortizing individual fairness in rankings. In: The 41st Int acm sigir Conf Res & Develop Inf Retr, pp 405–414
4. Calders T, Verwer S (2010) Three naive bayes approaches for discrimination-free classification. Data Min Knowl Disc 21(2):277–292. https://doi.org/10.1007/s10618-010-0190-x
5. Celis LE, Straszak D, Vishnoi NK (2017) Ranking with fairness constraints. arXiv preprint arXiv:1704.06840
6. Chen L, Ma R, Hannák A, et al (2018) Investigating the impact of gender on rank in resume search engines. In: Proc 2018 chi Conf Hum Fact Comput Syst, pp 1–14
7. Dong Y, Kang J, Tong H, et al (2021) Individual fairness for graph neural networks: a ranking based approach. In: Proc 27th ACM SIGKDD Conf Knowl Discov & Data Min, pp 300–310
8. Elbassuoni S, Amer-Yahia S, Ghizzawi A, et al (2019) Exploring fairness of ranking in online job marketplaces. In: 22nd Int Conf Ext Database Technol (EDBT)
9. Elbassuoni S, Amer-Yahia S, Ghizzawi A (2020) Fairness of scoring in online job marketplaces. ACM Trans Data Sci 1(4):1–30
10. Fagin R, Lotem A, Naor M (2003) Optimal aggregation algorithms for middleware. J Comput Syst Sci 66(4):614–656
11. Fisher ML, Jaikumar R, Van Wassenhove LN (1986) A multiplier adjustment method for the generalized assignment problem. Manage Sci 32(9):1095–1103
12. Foulds JR, Islam R, Keya KN, et al (2020) An intersectional definition of fairness. In: 2020 IEEE 36th Int Conf Data Eng (ICDE), IEEE, pp 1918–1921
13. Geyik SC, Ambler S, Kenthapadi K (2019) Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In: Proce 25th acm sigkdd Int Conf Knowl Discov & Data Min, pp 2221–2231
14. Ghizzawi A, Marinescu J, Elbassuoni S, et al (2019) Fairank: an interactive system to explore fairness of ranking in online job marketplaces. In: 22nd Int Conf Ext Database Technol (EDBT)
15. Ghosh A, Genuit L, Reagan M (2021) Characterizing intersectional group fairness with worst-case comparisons. In: Artif Intell Divers, Belong, Equity, and Incl, PMLR, pp 22–34
16. Hannak A, Wagner C, Garcia D, et al (2017) Bias in online freelance marketplaces: evidence from taskrabbit and fiverr. In: Proc 2017 ACM Conf Comput Support Coop Work Soc Comput, CSCW 2017, Portland, OR, USA, February 25 - March 1, 2017, pp 1914–1933
17. Jahanbakhsh F, Cranshaw J, Counts S, et al (2020) An experimental study of bias in platform worker ratings: the role of performance quality and gender. In: Proc 2020 CHI Conf Hum Fact Comput Syst, pp 1–13
18. Karabakal N, Bean JC, Lohmann JR (1993) A steepest decent [sic] multiplier adjustment method for the generalized assignment problem. Tech rep
19. Keane MT, O'Brien M, Smyth B (2008) Are people biased in their use of search engines? Commun ACM 51(2):49–52
20. Lagoudakis MG (1996) The 0-1 knapsack problem–an introductory survey
21. Martello S, Toth P (1990) Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons Inc, USA
22. Morina G, Oliinyk V, Waton J, et al (2019) Auditing and achieving intersectional fairness in classification problems. arXiv preprint arXiv:1911.01468
23. Naghiaei M, Rahmani HA, Deldjoo Y (2022) Cpfair: personalized consumer and producer fairness re-ranking for recommender systems. In: Proc 45th Int ACM SIGIR Conf Res Develop Inf Retr, pp 770–779
24. Oosterhuis H (2021) Computationally efficient optimization of plackett-luce ranking models for relevance and fairness. In: Proc 44th Int ACM SIGIR Conf Res Develop Inf Retr, pp 1023–1032
25. Osman IH (1995) Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. Oper Res Spektrum 17(4):211–225
26. Patro GK, Biswas A, Ganguly N et al (2020) Fairrec: two-sided fairness for personalized recommendations in two-sided platforms. Proc Web Conf 2020:1194–1204
27. Posta M, Ferland JA, Michelon P (2012) An exact method with variable fixing for solving the generalized assignment problem. Comput Optim Appl 52(3):629–644
28. Raj A, Ekstrand MD (2022) Measuring fairness in ranked results: an analytical and empirical comparison. In: Proc 45th Int ACM SIGIR Conf Res Develop Inf Retr, pp 726–736

29. Rosenblat A, Levy KE, Barocas S et al (2017) Discriminating tastes: uber's customer ratings as vehicles for workplace discrimination. Policy Internet 9(3):256–279
30. Salimi B, Rodriguez L, Howe B, et al (2019) Interventional fairness: causal database repair for algorithmic fairness. In: Proc 2019 Int Conf Manag Data, pp 793–810
31. Singh A, Joachims T (2018a) Fairness of exposure in rankings. In: Proc 24th ACM SIGKDD Int Conf Knowl Discov & Data Min, KDD 2018, London, UK, August 19-23, 2018, pp 2219–2228
32. Singh A, Joachims T (2018b) Fairness of exposure in rankings. In: Proc 24th ACM SIGKDD Int Conf Knowl Discov & Data Min, pp 2219–2228
33. Yagiura M, Ibaraki T (2007) Generalized assignment problem. In: Gonzalez TF (ed) Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series). Chapman & Hall/CRC
34. Zehlike M, Castillo C (2020) Reducing disparate exposure in ranking: a learning to rank approach. Proc Web Conf 2020:2849–2855
35. Zehlike M, Bonchi F, Castillo C, et al (2017) Fa* ir: A fair top-k ranking algorithm. In: Proc 2017 ACM Conf Inf Knowl Manag, pp 1569–1578
36. Zehlike M, Sühr T, Baeza-Yates R et al (2022) Fair top-k ranking with multiple protected groups. Inf Process Manag 59(1):102,707
37. Zliobaite I (2015) A survey on measuring indirect discrimination in machine learning. CoRR abs/1511.00148. http://arxiv.org/abs/1511.00148