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

Anis El Rabaa, Shady Elbassuoni, Jihad Hanna, Amer Mouawad, Ayham Olleik, Sihem Amer-Yahia

Noname manuscript No.
(will be inserted by the editor)

AUTHORS' COPY

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

Anis El Rabaa · Shady Elbassuoni · Amer Mouawad · Jihad Hanna · Ayham Olleik · Sihem Amer-Yahia

Received: date / Accepted: date

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 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 to reach a wider and more 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 online labor platforms, and 2) when jobs are being deployed by job providers on multiple online labor platforms. In our proposed framework, 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 proposed framework.

Keywords group fairness · online labor platforms · crowdsourcing · optimization · job seeker · job provider

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

Anis El Rabaa
Computer Science Department
American University of Beirut, Lebanon
E-mail: ase29@mail.aub.edu

Shady Elbassuoni
Computer Science Department
American University of Beirut, Lebanon
E-mail: se58@aub.edu.lb

Amer Mouawad
Computer Science Department
American University of Beirut, Lebanon
E-mail: aa368@aub.edu.lb

Jihad Hanna
Electrical and Computer Engineering Department
American University of Beirut, Lebanon
E-mail: jgh20@mail.aub.edu

Ayham Olleik
Electrical and Computer Engineering Department
American University of Beirut, Lebanon
E-mail: abo00@mail.aub.edu

Sihem Amer-Yahia
CNRS, University Grenoble Alpes, France
E-mail: sihem.amer-yahia@univ-grenoble-alpes.fr

1 Introduction

Online labor platforms such as TaskRabbit[1] and Upwork[2] are gaining popularity as platforms to hire workers to perform certain jobs. On these platforms, people can find temporary workers in the physical world (e.g., someone to clean an apartment in New York City), or remote workers such as "someone to develop a mobile app" or "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 by the platform. These platforms thus rely heavily on job-matching services. 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. On the other hand, 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

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

the majority of these platforms, such job-matching services are algorithmic and most of the time opaque.

The algorithmic and opaque nature of job-matching services in online labor platforms thus raises fairness concerns. For instance, consider a job provider looking for someone to move furniture in San Francisco on an online labor platform such as TaskRabbit. The job provider receives a ranked list of potential workers on the platform for this job. Such ranking will be considered unfair if it is biased towards certain groups of people, say where white males are consistently ranked above black males or white females. This commonly happens since such ranking usually depends on the ratings of workers on the platform and the number of their past jobs, both of which perpetuate bias against certain groups of workers [13,21,6].

In online labor platforms, job seekers and job providers face many limitations, such as inability to state own constraints when seeking a job or limited control on job deployment. As the number of such online labor platforms and the jobs available on them increase, it becomes crucial to provide both job seekers and job providers with means to assess and compare the fairness of different jobs on different platforms. This can then be used to inform job seekers about which jobs on which platforms are deemed the most fair with respect to their demographic groups, thus maximizing their chances of landing jobs. Similarly, this can be used by job providers to decide on which platforms to deploy which jobs so as to maximize worker fairness.

In this paper, we propose a theoretical framework that can be used to assess and compare worker fairness of multiple jobs on multiple online labor platforms. We focus on group fairness, which is defined as the fair treatment of all groups of people [3,27], where groups are defined using protected attributes such as gender, age, or ethnicity. For example, the worker groups could be males, females, asians, whites, blacks, black females, young white males, etc. Our framework encapsulates multiple group fairness definitions proposed in the literature, such as demographic parity, disparate impact, and disparate treatment. 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.

Our framework can be used by two types of end-users: 1) job seekers looking to find which jobs to apply to on which platforms, and 2) job providers looking to deploy multiple jobs on multiple platforms. To be able to serve these two types of users, we formulate a series of optimization problems that aim to maximize worker group fairness subject to various constraints such as payment constraints, number of jobs applied to, etc.

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 goal in the first optimization problem we propose is to find 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 the find the top-k fairest job-platform pairs that increase her chances of landing a job, while guarantying 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. We assume that each job is associated with a cost on a platform it is available on, and that this cost differs from one platform to the other. Given a set of jobs to be deployed on a set of platforms and a budget, our goal in the third optimization problem is 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. A slight variation of this optimization problem is our fourth and final optimization problem we define. 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 thus 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 reduction to well-known NP hard problems such as Knapsack [17] and General Assignment problems [18], and we propose algorithms to efficiently solve all four

of them. More precisely, for the first Job Seeker optimization problem, we propose an adaptation of Fagin's Top-k algorithm [8] to solve the problem. For the the second Job Seeker problem, we propose a new Dynamic Programing algorithm to solve the problem. Similarly, for the first Job Provider optimization problem, we also propose a Dynamic Programming algorithm to solve that problem and finally, for the second Job Provider problem, we explore various exact and approximation algorithms from the literature to solve our fourth optimization problem.

Empirical Validation. We also 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. More precisely, we use synthetic data to demonstrate the scalability of our proposed algorithms as the number of jobs, the number of platforms and the number of worker groups increase and to compare them to adequate baselines. Our experiments demonstrate that our proposed algorithms scale very well and that they consistently outperform the compared-to baselines. On the other hand, we use semi-synthetic data to conduct case studies that highlight the merits of the solutions generated by our proposed algorithms from a qualitative perspective. Our qualitative experiments 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 ones.

The rest of the paper is organized as follows. In Section 2, we review related work that addresses fairness in online labor platforms. In Section 3, we describe our proposed framework, which is composed of four optimization problems and algorithms to solve them efficiently. In Section 4, we describe the experiments that we used to evaluate our proposed framework and their results. Finally, we conclude and present future work in Section 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 [22], 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 [16] studied user behavior when presented with manipulated Google search results, and found that users exhibit "partial bias" towards 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 [2].

Many notable works focused on assessing fairness of a worker ranking in online labor platforms. For instance, the authors in [12] 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 [5], 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 [7,11,6,10]. In [7,11,6], the authors formulated an optimization problem to find the partitioning of workers (based on their protected attibutes) 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 in online labor platforms, various methods have been proposed to actively generate fair rankings. Many of them are *post-processing* methods (e.g., [24,2,4,26]), where given an existing ranking of workers, a new ordering of the workers 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 [25].

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 as well as real-world generated data.

3 Framework

Our framework assumes the presence of an unbounded number of platforms on which an unbounded number of jobs are available. A job can be available on multiple platforms, and each job is associated with a different fairness value for each worker group on each platform. The worker groups are defined using one or more protected attributes such as gender, ethnicity, age and so on. For example, the worker groups could be males, females, asians, whites, blacks, black females, young white males, etc.

More precisely, we assume that a job j for demographic 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. We do not make any assumptions on how these fairness values are computed and thus different methods for computing them that depend on different group fairness notions can be seamlessly plugged into our framework. Sihem: Say first that the different fairness definitions proposed before can be accommodated. Shady: We already mention that, do you suggest we rephrase? In our experiments, we make use of the framework in [1], which uses two different notions for computing group fairness.

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 group g is available for job j on platform p. 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 correspond to the different fairness values for the different groups that exist in the platform p for job j. Figure 1 shows an example of such bipartite graph.

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.

3.1 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:

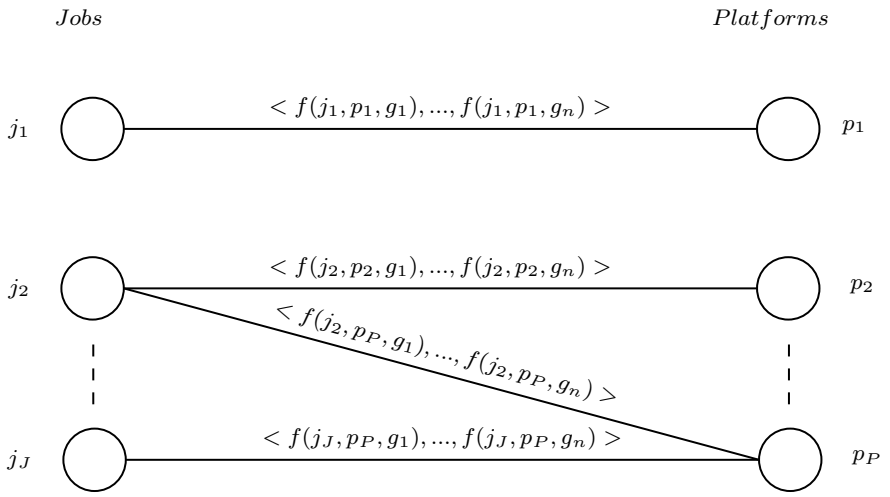


Fig. 1: An example bipartite graph with jobs on one side and platforms on the other side. Each edge between a job j and a platform p has a set of weights representing the fairness values of job j for the different groups g on platform p

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

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

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. 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. 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 $O(|J||P||G|)$.

A more efficient approach can make use of optimal aggregation algorithms such as Fagin's Algorithm [8] 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 index 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, 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*. The worst-case scenario for this algorithm is reading all entries from all lists, giving a worst-case time complexity of $O(|\mathcal{J}||\mathcal{P}||G|)$, where $|\mathcal{J}|$ is the total number of jobs

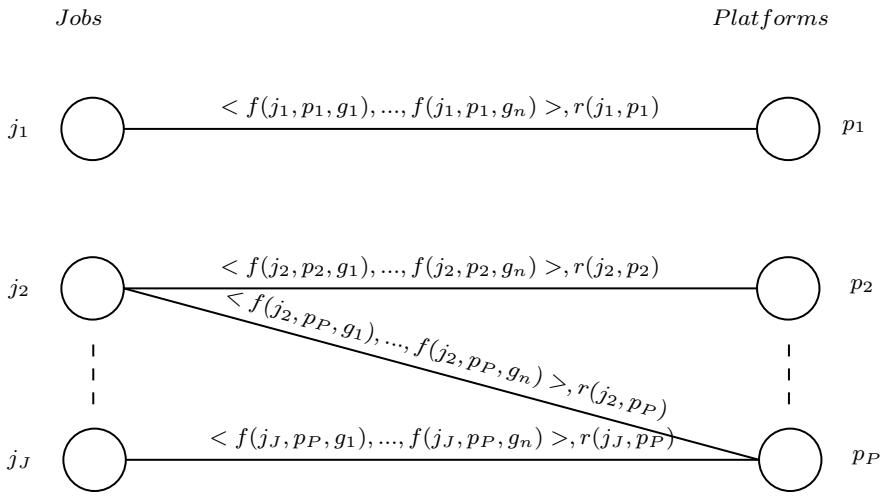


Fig. 2: An example bipartite graph for the Constrained Job Seeker problem. In addition to the fairness values per group, each edge between a job j and a platform p has a weight $r(j,p)$ representing the reward of job j on platform p

in the inverted lists (i.e., all possible jobs; which is usually different from $|J|$, the number of jobs *of interest* for the seeker), and $|\mathcal{P}|$ is the total number of platforms (again different from $|P|$, which is the number of platforms that is provided as input to the algorithm).

Shady: Why does Algorithm 1 have blue header and footer? Anis: Fixed, there was a comment (in blue) going through it, which is now moved.Shady: It still has blue headers and footers.

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 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 Figure 2. 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 formulated as the following optimization problem:

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

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

The same problem can be formulated as an Integer Linear Programming optimization problem as follows:

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

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

Theorem 1 *The Constrained Job Seeker problem is polynomial-time reducible to the optimization variant of the Knapsack problem and is therefore at least as hard.*

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 [17], this theorem gives us a lower bound on the hardness of the Constrained Job Seeker problem.

Sihem: The proof is long, Maybe we could have a sketch here and put the proof in the appendix or an

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 $topk$

extended report registered in arxiv (preferred option)? Shady: I agree with Sihem. Anis: Alright, moved the proof to the appendix.

Proof See Appendix A. Shady: Add a sketch of the proof here. Please seek Amer's help with this.

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. Anis: Is this approach really "nearly immediate" to the Knapsack DP? The proposed solution is a recursive approach, where for a given ordering of the job-platform pairs in a list, we take each pair and recursively find a) the best fairness value attainable if we choose this pair to be in the top-k results, and b) the best fairness attainable if we do *not* choose this pair. Based on the results of both options, the decision is made to either take the pair or leave it aside. To avoid repetitive calculations, we make use of DP by storing intermediate results in matrices, for ease of reuse by subsequent recursive calls. Shady: Not quite sure the description of the algorithm is sufficient. Anis: Not sure how to describe further. Maybe Prof. Amer and Jihad can help here?

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 DP phase described above, which has a time complexity of $O(|J||P|kR)$. So overall, this algorithm has a time complexity of $O(max(|J||P||G|, |J||P|kR))$.

Anis: NOTE: The running time stated in Algorithm 2 was $O(|J||P|kR)$. However, if we take into account the preprocessing phase which has runtime $O(|J||P||G)$, then the overall runtime becomes $O(max(|J||P||G|, |J||P|kR))$, as discussed in the above paragraph. Shady: Run this by Amer. Also, not sure the explanation of this is clear enough.

3.2 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 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 the other. This extends our bipartite graph in Figure 1 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 Figure 3. 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):

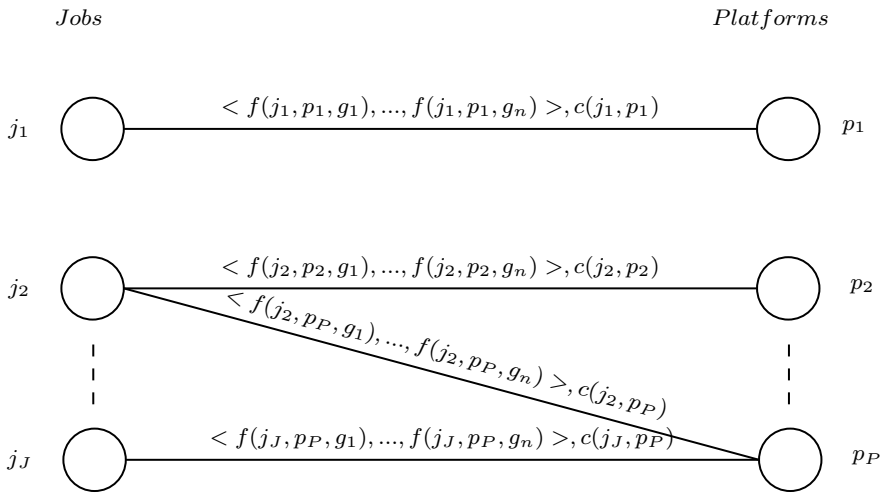


Fig. 3: An example bipartite graph for the Job Provider problem. In addition to the fairness values per group, each edge between a job j and a platform p has a weight $c(j,p)$ equal to the cost of deploying job j on platform p

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

$$\text{subject to: } x(j,p) \in \{0,1\} \ \forall j \in J, \forall p \in P$$

$$x(j,p) = 1 \rightarrow a(j,p) = true \ \forall j \in J, \forall p \in P$$

$$\sum_{j \in J} \sum_{p \in P} c(j,p) \times x(j,p) \leq B$$

$$\sum_{p \in P} x(j,p) \leq 1 \ \forall j \in J$$

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 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:

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

$$\text{subject to: } x(j,p) \in \{0,1\} \ \forall j \in J, \forall p \in P$$

$$x(j,p) = 1 \rightarrow a(j,p) = true \ \forall j \in J, \forall p \in P$$

$$\sum_{j \in J} c(j,p) \times x(j,p) \leq b_p \ \forall p \in P$$

$$\sum_{p \in P} x(j,p) \leq 1 \ \forall j \in J$$

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 both polynomial-time reducible to the optimization variant of the Knapsack problem and are therefore at least as hard.*

Proof Constraining both problems to one group and one platform gives 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.

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

Algorithm 2 Constrained Job Seeker Algorithm

Input: A set of jobs J, a set of platforms P, a set of groups G, and two integers k and R.
Output: The k (j,p) pairs with the highest minimum fairness over all groups G having reward at least R. Running time is $\mathcal{O}(max(JPG, JPkR))$.

▷ Step 1: Initialization + aggregation of fairness values

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

$L \leftarrow$ Empty list
for $j \in J$ and $p \in P$ **do**
  $(j,p,f,r) \leftarrow (j,p,minFair[j][p], r(j,p))$
  $L.append((j,p,f,r))$
end for

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

$DP[0...len(L)][0...k][0...R] \leftarrow$ new 3D array initialized to -1.
$choice[0...len(L)][0...k][0...R] \leftarrow$ new 3D array initialized to -1.
$maxFairness \leftarrow$ RECURSIVEMAXFAIRNESS(1, L, k, R, DP, $choice$)
if $maxFairness = -\infty$ **then return** ϕ

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

$i \leftarrow 0$, $result \leftarrow \phi$
while $i \neq len(L)$ **do**
  if $choice[i][k][R] = 0$ **then**
    $i \leftarrow i + 1$
    continue
  end if
  $result.add((j,p))$
  $k \leftarrow k - 1$
  $R \leftarrow \max(0,\ R - L[i].r)$
  $i \leftarrow i + 1$
end while

return $result$

Algorithm 3 Recursive Maximum Fairness Algorithm

procedure RECURSIVEMAXFAIRNESS(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** $DP[i][k][R]$

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

computation time, by avoiding repetitive calculations. Complexity-wise, this algorithm is composed of two main parts: a preprocessing phase similar to Algorithm 2's 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 $O(max(|J||P||G|, JPB))$.

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 *exaclty* one [18]. And since LEGAP is proven to be equivalent to the "standard" GAP [18,23], then Problem 4 (with pre-aggregated fairness values) is equivalent to GAP. This implies that Problem 4 is, like GAP, strongly NP-hard.

The advantage of this equivalence is that GAP algorithms from the literature can solve our problem [9, 18,20,15]. The only adjustment required to our 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 $c(j, p_{dummy}) = 0\ \forall j \in J$), and its budget limit to $|J|$. This creates an instance of GAP that is equivalent to our problem, and thus can be directly solved by available GAP algorithms. On the other hand, however, the strong NP-hardness of Problem 4 gives us a few limitations. By the property of strong NP-hardness, we have that: 1) no exact pseudo-polynomial time algorithm (such as DP-based methods) can exist for our problem, unless $\mathcal{P} = \mathcal{NP}$; and 2) no polynomial-time approximation scheme with a mathematically-guaranteed solution quality can exist either, unless $\mathcal{P} = \mathcal{NP}$ [18]. Therefore, when proposing an adequate algorithm to solve Problem 4, we are left with two possible choices: either non polynomial-time exact algorithms, or more efficient heuristics with no mathematical guarantee on solution accuracy.

With this in mind, we start first by exploring exact GAP algorithms from the literature. A common out-

Algorithm 4 Job Provider Problem with Global Budget Algorithm

Input: A set of jobs J, a set of platforms P, a set of groups G, and an integer B.
Output: The maximum size subset of (j, p) pairs with the highest minimum fairness over all groups G having cost at most B and where each job is assigned to at most one platform. Running time is $\mathcal{O}(max(JPG, JPB))$.

▷ Step 1: Initialization, aggregation of fairness values
$minFair[1...len(J)][1...len(P)] \leftarrow$ new 2D array initialized to $+\infty$.
for $j \in J$ and $p \in P$ and $g \in G$ **do**
  if $e(j, p, g) = true$ **then**
    $minFair[j][p] = \min(minFair[j][p], f(j, p, g))$
  end if
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.
$DP[0...len(J)][0...B] \leftarrow$ new 3D array initialized to 0.
for $i \in [0, len(J))$ and $t \in [0, B]$ **do**
  $dp[i+1][t] \leftarrow \max(dp[i+1][t], dp[i][t])$
  for $j \in [1...len(P)]$ **do**
    $(f, c) \leftarrow (minFair[J[i]][P[j]], c(J[i], P[j]))$
    if $c + t \leq B$ **then**
      $dp[i+1][c+t] \leftarrow \max(dp[i+1][c+t], f + dp[i][t])$
    end if
  end for
end for

▷ Step 3: Get total cost of the optimal assignment found
$maxFairness \leftarrow 0$
$b \leftarrow 0$
$N \leftarrow len(J)$
for $t \in [0...B]$ **do**
  if $dp[N][t] > maxFairness$ **then**
    $maxFairnes \leftarrow dp[N][t]$
    $b \leftarrow t$
  end if
end for

▷ Step 4: Read result (optimal assignment) from the DP matrix and return
$result \leftarrow$ Empty list
while $N \neq 0$ **do**
  $(j) \leftarrow J[N]$
  if $dp[N-1][b] \neq dp[N][b]$ **then**
    for $i \in [1...len(P)]$ **do**
      if $b \geq c(j, P[i])$ and $dp[N][b] = minFair[j][P[i]] + dp[N-1][b - c(j, P[i])]$ **then**
        $result.append((j, P[i]))$
        $b \leftarrow b - c(j, P[i])$
        break
      end if
    end for
  end if
  $N \leftarrow N - 1$
end while

return $result$

line 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. [9,18], 2) the BB with steepest descent MAM by Karabakal et al. [15], and 3) the BB with variable fixing by Posta et al. [20]. 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 [20]). A scalability comparison of these algorithms is included in Section 4.

Shady: Add some complexity analysis. I understand that we are using solutions from the literature, but it would still be good to provide bounds on these different approaches and algorithms. Anis: The papers of these approaches do not provide time complexity claims for them, except for MTHG which is $O(|J||P|log|P|+|J|^2)$. For the exact BB approaches, their worst case scenario includes visiting all possible solutions ($O((|P|+1)^{|J|})$), plus additional computations (like computing the initial lower bound, or extra computations inside a search node for the Posta et al.'s algorithm). So the BB algorithms have exponential runtime, but I cannot give an exact complexity claim for them.

Anis: For the heuristics, all I can say is that MTHG is polynomial as mentioned above. The other heuristics (TS and LS Descent) use the solution of MTHG as a starting point and improve on it, so their runtime is $O([the MTHG\ one\ above] + somevalue)$. For TS precisely, it is $O([MTHG\ runtime] + |J|^2|P|)$ (thus also polynomial) **because** of its iteration limit we set to $100 \times |J|$. For LS Descent we did not set an iteration limit so I am not sure.

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 [18]; 2) a Local Search Descent method by Osman [19]; and 3) a Tabu Search method by Osman [19]. A comparison of these algorithms, both in terms of performance and solution quality, can be found in Section 4.

4 Experiments

To evaluate our proposed framework, we design two sets of experiments. The first set aims to study the scalability of our proposed 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 the different problems and for that we use semi-synthetic data generated from a real-world online platform.

We divide this section as follows. First, we explain how the semi-synthetic dataset (used in qualitative experiments) is generated. 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 ones.

4.1 Data Generation

To simulate multiple, semi-synthetic platforms, we use the TaskRabbit dataset from [1], and generate eight different "worlds" from it using interventions. An intervention is a sampling of the initial dataset's workers 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, is saved to files for ease of reuse, and we refer to these nine platforms collectively as the *alternative worlds*.

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 from switching the percentages 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 1. In the remainder of this section, we will be interchangeably using the words *world* and *platform* to refer to platforms.

4.2 Job Seeker Experiments

4.2.1 Algorithms Implementation

For the Unconstrained Job Seeker problem, both the naive algorithm that loops over all jobs, all platforms and all groups, and the top-k algorithm were implemented in Python 3.8, as the function to compute fairness values defined in [1], and needed for the naive algorithm, was already implemented in Python. For the top-k algorithm, the index files were built as simple text files for sequential access, each accompanied with a positions table for random access.

For the Constrained Job Seeker variant, we implemented the proposed algorithm in C++, since this routine relies on dynamic programming.

All scalability experiments were run on the same computer, an Apple MacBook Pro with a 2.3 GHz dual-core Intel Core i5 processor. Throughout this paper, all solving times are measured as CPU time, except for the Unconstrained Job Seeker experiments. For the latter, real (wall-clock) time was used, since the top-k algorithm relies on disk reads and memory accesses, which should be accounted for.

For all qualitative experiments, the fairness scoring function used is the EMD metric from [1].

4.2.2 Unconstrained Problem Scalability Experiments

To compare the performance of our two Unconstrained Job Seeker algorithms at various scales, we built a fully-synthetic dataset consisting of 5000 jobs and 70 platforms. Each job in each platform is represented as a file, containing a ranked list of its fictional workers. The number of these workers for each job-platform pair is a random value between 0 and 50. In addition, each worker is assigned values for two protected attributes, also at random. Then, the corresponding index files for the top-k algorithm are built from the generated data.

On this new dataset, we run both the naive and top-k algorithms we implemented, using increasing values of $|J|$, $|P|$, and k on each run[3]. To compute fairness values, we use the two metrics defined in [1], namely Earth Mover Distance (EMD) and Exposure. Therefore, this scalability experiment is run for both metrics.

The experiment then goes as follows. For each run, we generate ten fictional job seekers, and assign to each of them $|J|$ jobs and $|P|$ platforms of interest at random. Then, we find the top-k job-platform pairs for

[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.

World	Male	Female
Taskrabbit	0.75	0.25
World1	0.26	0.74
World2	0.50	0.50
World3	0.30	0.70
World4	0.70	0.30
World5	0.74	0.26
World6	0.72	0.28
World7	0.74	0.26
World8	0.75	0.25

(a) Gender statistics

World	Black	White	Asian
Taskrabbit	0.24	0.69	0.07
World1	0.27	0.66	0.07
World2	0.25	0.68	0.07
World3	0.26	0.67	0.07
World4	0.24	0.69	0.07
World5	0.33	0.33	0.34
World6	0.69	0.24	0.07
World7	0.24	0.07	0.69
World8	0.07	0.69	0.24

(b) Ethnicity statistics

World	Male Asian	Male Black	Male White	Female Asian	Female Black	Female White
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

(c) Group statistics

Table 1: Platform statistics for the alternative worlds (in percentages)

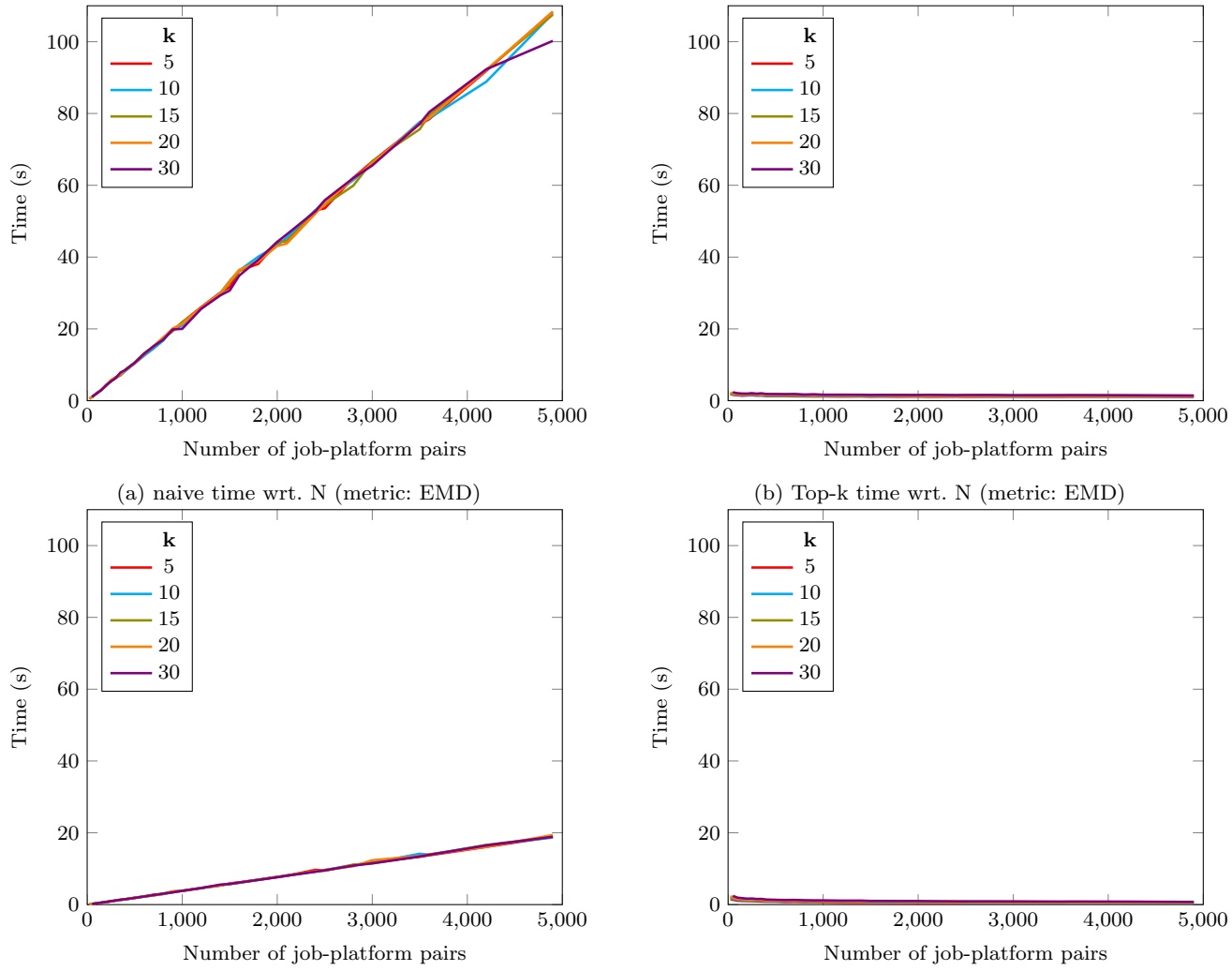
each seeker using both the naive and the top-k algorithms. Each possible $(|J|, |P|, k)$ combination is ran for all seekers, and the average running time of each algorithm per combination is recorded.

After performing all the runs, we first plot the execution time versus the number of job-platform pairs for all values of k in Figure 4. As the curves for the different values of k show very similar trends, we only focus on $k = 20$ for comparing the naive and the top-k algorithms. A plot comparing runtimes for both algorithms at $k = 20$ is shown in Figure 5. As the figure shows, a general trend is 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. Also, it seems that the naive algorithm performs better using the Exposure fairness metric rather than EMD, as EMD is more computationally expensive.

Next, we analyze how well the top-k algorithm scales as the number of protected attributes n increases. For this, we generate a new synthetic index, which also assumes 5000 jobs and 70 platforms. This index is essentially a large set of index files that map each new job-platform pair to a random fairness value, and where each index file represents one group.

At this stage, it is important to distinguish between a *protected attribute* and a *group*. While a protected attribute is only one attribute or characteristic, such as gender or age, a 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 then belongs to all groups that are combinations of one or more of their protected attributes' values. For example, a male asian worker 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 (e.g., a worker does not have two ages at the same time), then the total 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.

So, each synthetic index file corresponds to a *group*, and therefore, when we consider n protected attributes for each seeker, we need to read $2^n - 1$ index files concurrently for each seeker during the top-k algorithm run. This therefore hints at an exponential growth in runtime as we increase n, which is confirmed by the plot in Figure 6.



(a) naive time wrt. N (metric: EMD)

(b) Top-k time wrt. N (metric: EMD)

(c) Naive time wrt. N (metric: Exposure)

(d) Top-k time wrt. N (metric: Exposure)

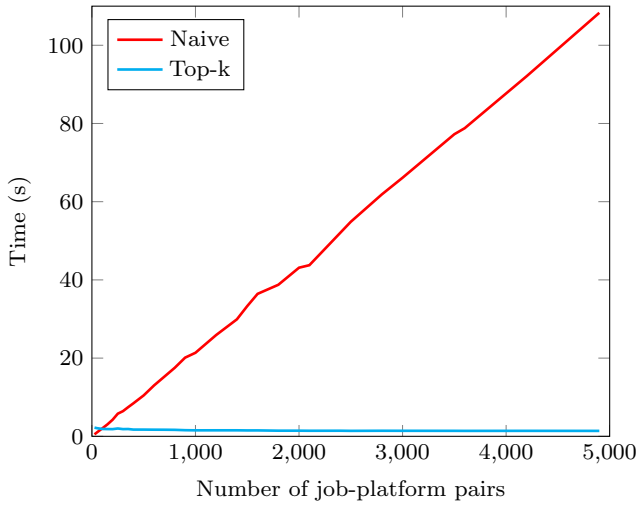
Fig. 4: Naive vs. top-k performance for different values of k

4.2.3 Unconstrained Problem Qualitative Experiments

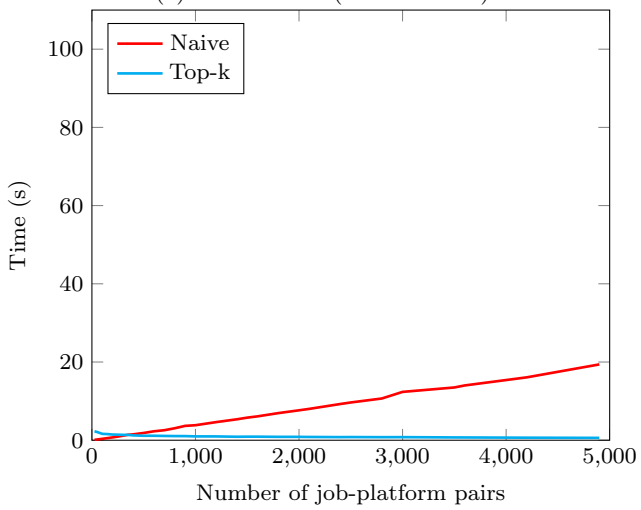
We design two experiments in this section. The first one focuses on the alternative worlds, and how their demographic group distributions affect the search results for seekers of different groups. This experiment goes as follows: generate six seekers (one per gender-ethnicity combination), assign the same $|J| = 20$ random jobs of interest to all seekers, set their platforms of interest to be the nine alternative worlds, and fetch the top five fairest (j, p) pairs for each seeker using the top-k algorithm. For each top-five result set, the number of occurrences of each platform is recorded in Figure 7, and the number of occurrences of each job in Figure 8.

Looking at the world frequency results, we see that platforms *world*2 and *world*4 are present in all of the seekers' top-five results, suggesting that these worlds are fair to every group for the twenty chosen jobs of interest. On the other hand, we see that *taskrabbit* and *world*7 do not occur in any of the seekers' top-five results, which suggests these platforms are the least fair of the bunch for the chosen jobs. For the job frequencies, we notice that the job "Cleaning in London, UK" appears across the board, implying that this job is fair to all demographic groups in our study. Other frequently appearing jobs are "Furniture Shopping and Assembly in Colombus, OH", which appears in the top-five for all groups except the Black ones, and "Pack for a Move in Raleigh, NC" which appears for all groups except the Asian ones.

The second experiment investigates how the chosen worlds of preference affect a seeker's chances of finding fair jobs. For this, we fix one random set of jobs of interest, and assign it to all six seekers. Then, for each seeker



(a) Time wrt. N (metric: EMD)



(b) Time wrt. N (metric: Exposure)

Fig. 5: Naive vs. Top-k Times for $k = 20$

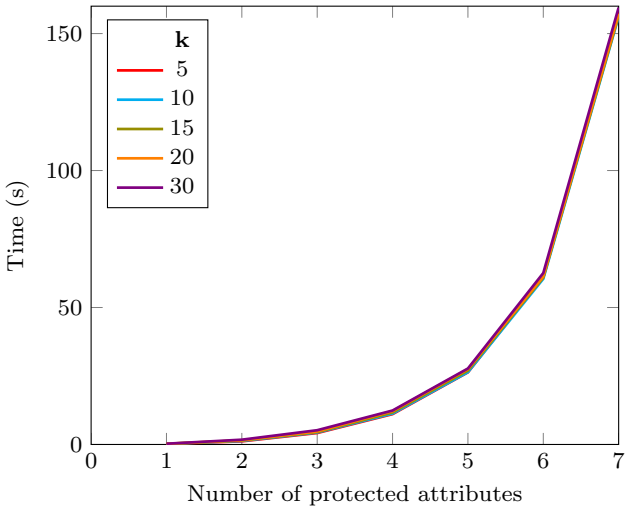


Fig. 6: Top-k algorithm runtime vs the number of protected attributes n

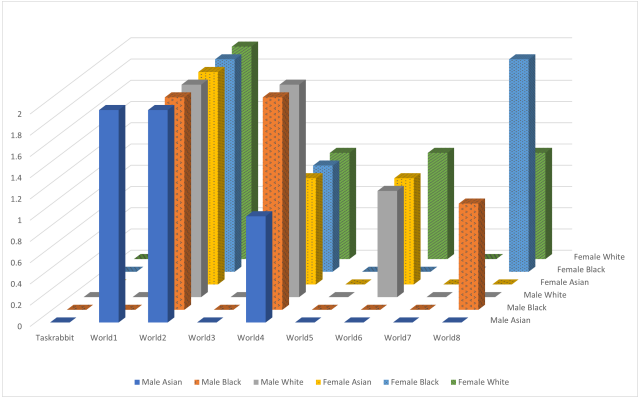


Fig. 7: Occurrences of each World in the seekers' top-5 results

and each alternative world p_i, we retrieve the seeker's top-5 fairest jobs in platform p_i. Then, we also retrieve the seeker's overall top-5 fairest jobs in all platforms combined. Finally, for each top-5 result set, the sum of the jobs' fairness values is computed and compared against the ones of the other sets. The obtained results are shown in Figure 9.

We notice that *world*7 has the lowest sum of fairness values across the board, which indicates that *world*7 is the least fair platform for the chosen set of jobs. Recall that *world*7 is the world sampled from *taskrabbit* such that the percentage of asians and whites is reversed. As asians form quite a minority in *taskrabbit* (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 poor fairness performance on the selected jobs, we compare statistics between this world and *world*2, one of the worlds that fared the best in our previous tests. We first compare the number of workers between *world*2 and *world*7 for the 20 jobs as shown in Figure 10. 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 containing less than five workers each. Also, we notice that many of these jobs only contain workers from a very few groups (especially the jobs that have very few workers). This leaves many demographic groups unrepresented in these jobs, hence we have no fairness data for the affected (job, world, group) combinations. As a result, these combinations cannot appear in any seeker's top results.

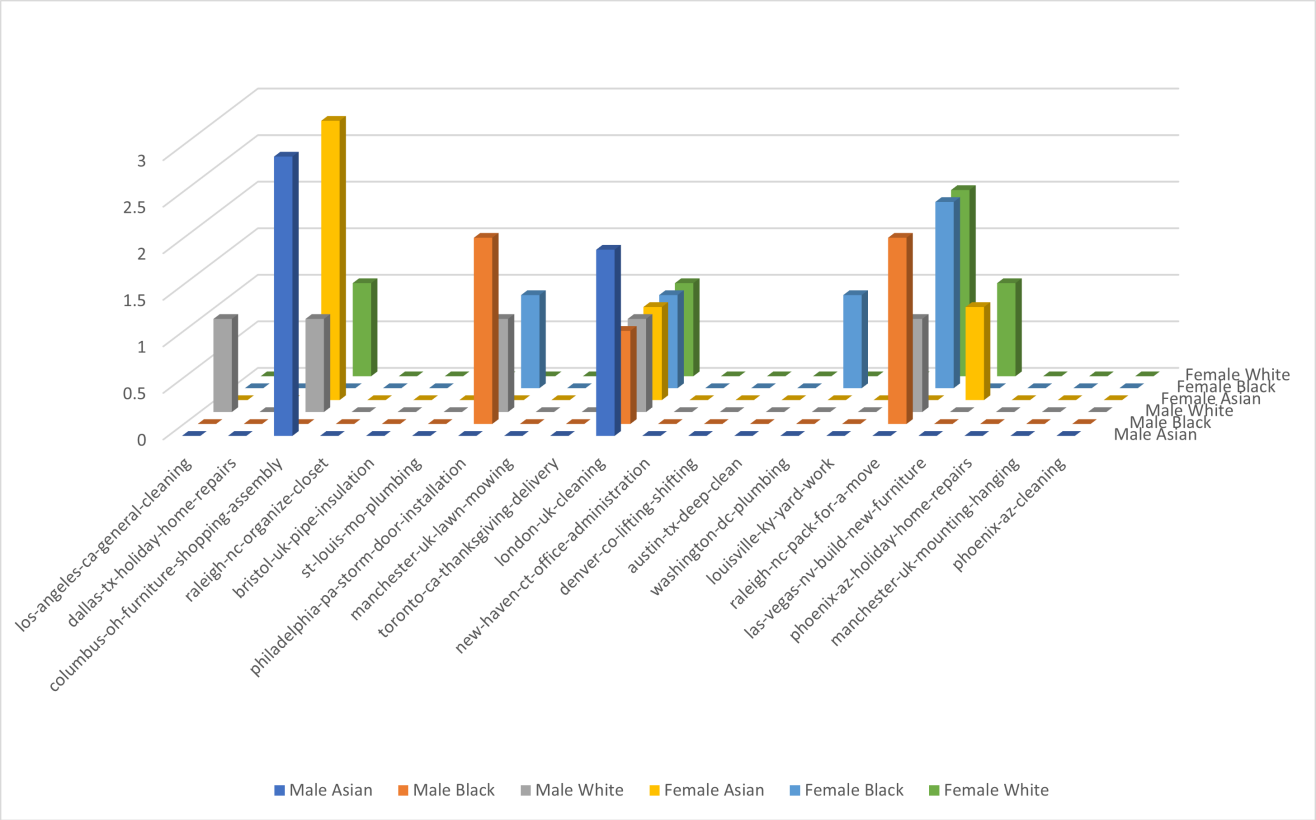


Fig. 8: Occurrences of each Job in the seekers' top-5 results

4.2.4 Constrained Problem Scalability Experiments

Shady: From here on, it seems we are only using one notion of fairness, is that correct? and which one is it? EMD? If yes, explicitly mention that and why. Anis: Done, see below.

Our previous experiments above established that our framework does indeed support multiple notions of fairness (EMD and Exposure in our case). So, from this point on, our experiments only focus on one fairness metric (EMD for qualitative expriements, random values for fully-synthetic scalability experiments) to avoid duplication of effort. Also, as explained later in Section 4.2.5, the DP-based algorithms of Section 3 expect fairness values as integers. For this, the fairness values we use in the following experiments are all integers rather than floats, and we do provide a method to convert fairness values from the $[0, 1]$ float range to an integer range in Section 4.2.5.

For the Constrained Job Seeker problem, we study the scalability of our Dynamic Programming algorithm (Algorithm 2) proposed in Section 3. To do this, we created a synthetic set of N job-platform pairs, where each pair is associated with a set of fairness values (one per group, selected at random between 1000 and 9999) and a reward value, selected at random between 10 and 99. From there, the task is to find, for various values of N and k, the top-k pairs that maximize fairness while satisfying a reward threshold of $80 \times k$. For now, the number of protected attributes n considered is fixed to $n = 2$ (and so, the number of groups considered is $2^2 - 1 = 3$). Shady: Explain why and how the fairness values were turned into integers. Anis: Done, see above paragraph.

So, for each run, we pick different fairness and reward values at random, and then find the desired optimum result in two ways: 1) using our Dynamic Programming (DP) algorithm from Section 3, and 2) an off-the-shelf Integer Linear Programming (ILP) solver (Google's ORTools[4]). We execute 10 such runs for every (N, k) combination, and record the average runtime of each algorithm over the 10 runs. The results are summarized and compared in the plots of Figure 11.

As the plots show, the proposed DP algorithm finds the desired results much faster than the general-purpose ILP solver, for all values of k considered. Both algorithm's runtimes seem to increase as N gets larger, but this observed increase for DP is less pronounced and

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

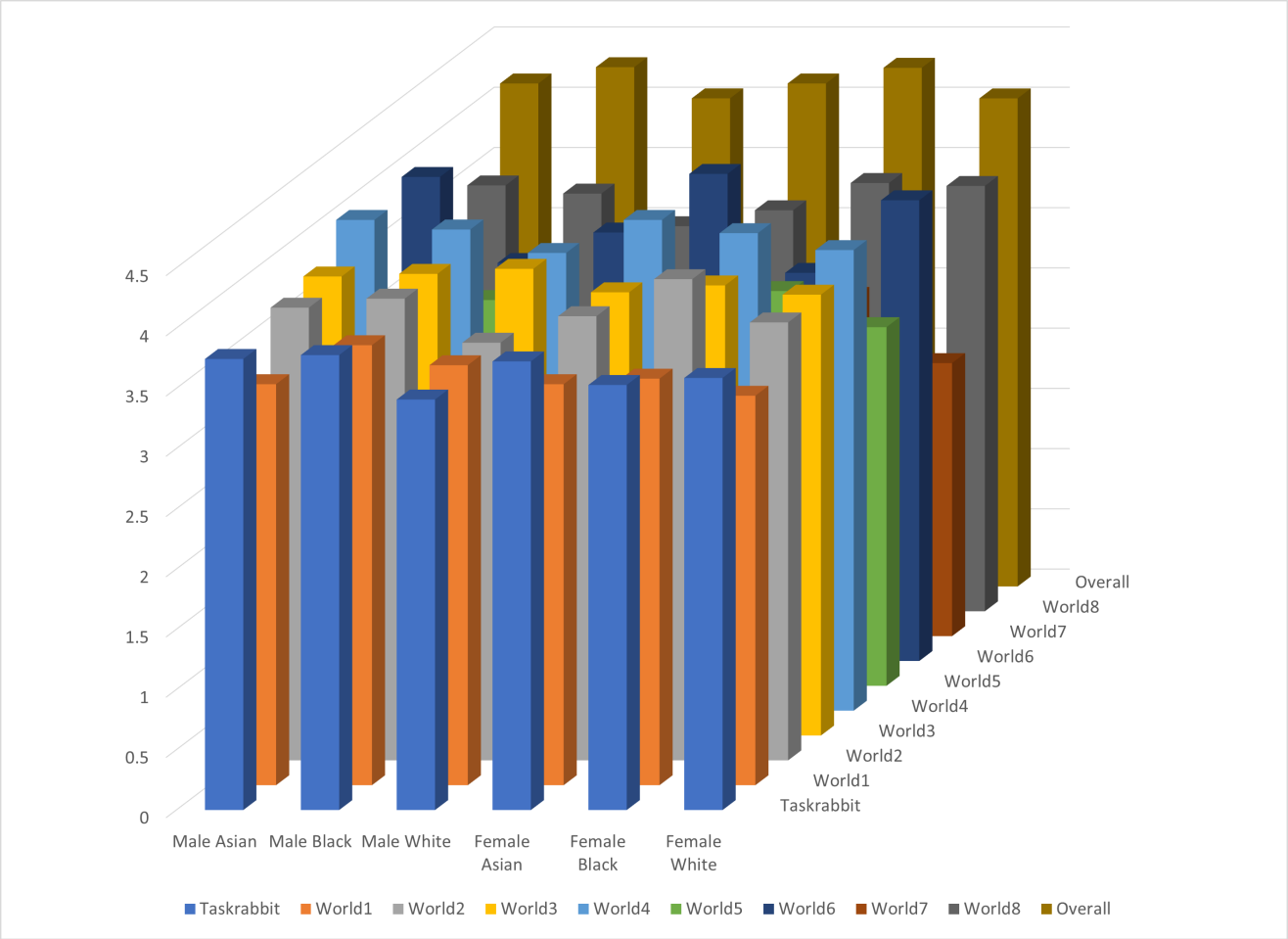


Fig. 9: Sum of fairness values of the top-5 (j, p) pairs per seeker

much more linear than for ILP. This suggests that the proposed DP scales much better than ILP in terms of N. With respect to k, we see the DP algorithm's running time 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.

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 of this experiment are shown in Figure 12.

From the plot, we can see that up until $n = 11$, the solving time 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 $O(JPG)$, followed by a solving phase using dynamic programming, of complexity $O(JPkR)$. Back to the plot, the point where the time starts increasing exponentially is the point where the value of JPG becomes significant (same order of magnitude) compared to $JPkR$. From there, we conclude that as long as the number of groups $G = 2^n - 1$ is of smaller order of magnitude than KR, then the DP algorithm's time will not depend much on n.

4.2.5 Constrained Problem Qualitative Experiments

As the Constrained Job Seeker algorithm requires fairness values to be input as integers, and our current values are floats between 0 and 1, we need to convert our values to integers before running the algorithm. The idea is then to truncate each fairness value to d significant digits, and then multiply 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

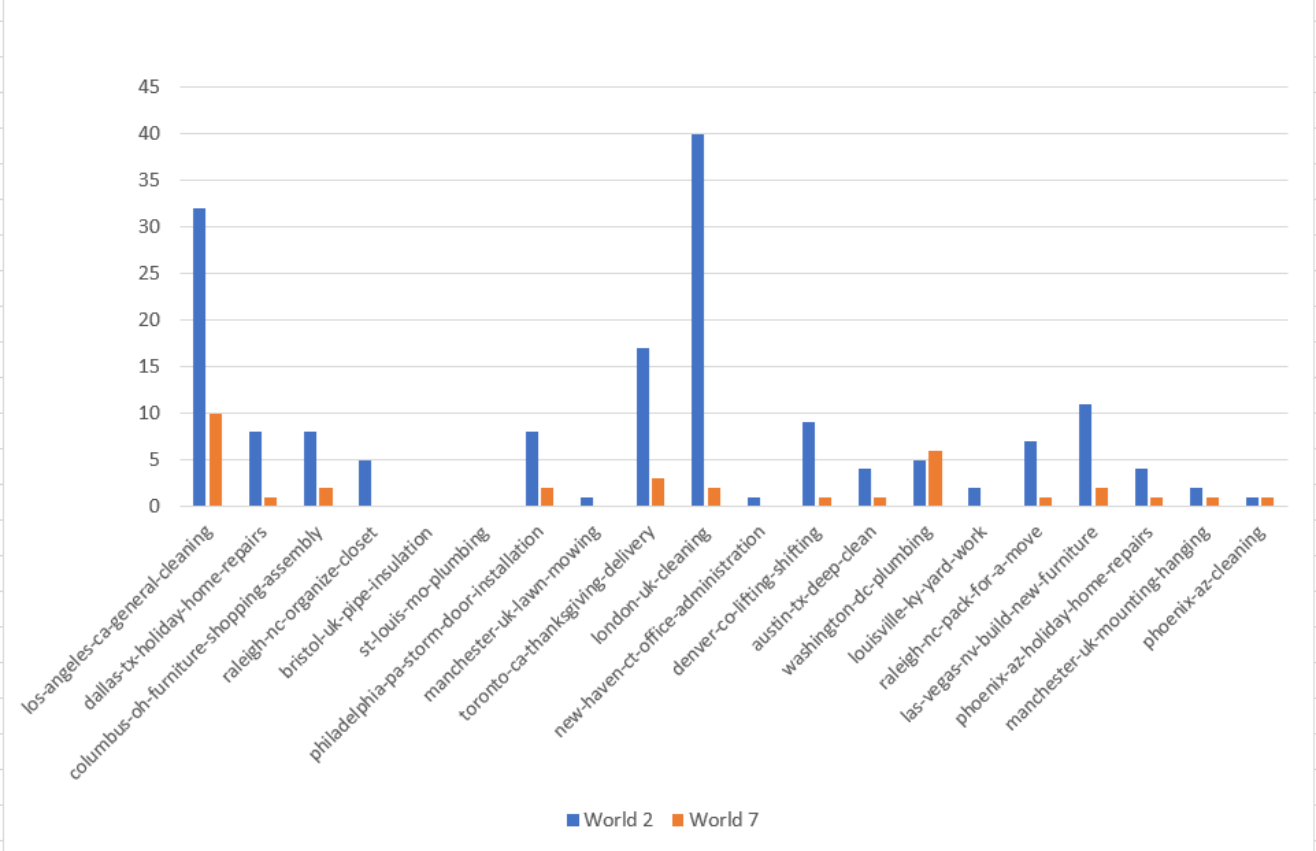


Fig. 10: Comparing worker counts for the 20 selected jobs in *world*2 vs. *world*7

integer, yet small enough that the fairness integers are not too large or too granular.

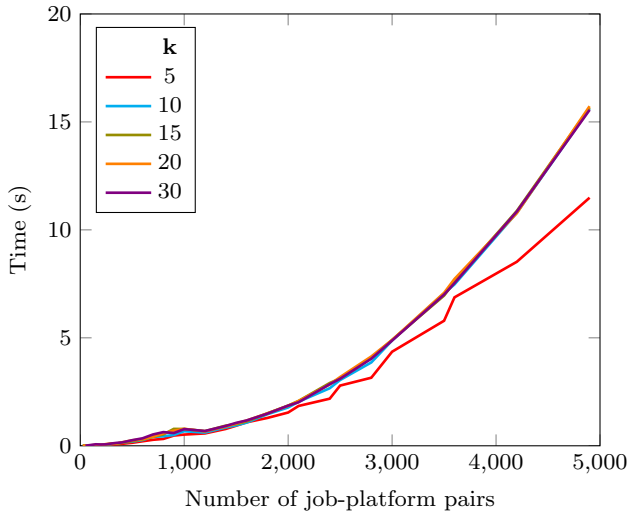
An optimal value of d would be the smallest value that gives us enough precision when truncating the fairness values, so as to avoid too many collisions when mapping to integers. To find the optimal d, we considered integers from 1 to 8 as candidate values. For each candidate value of d, we took all fairness values of our semi-synthetic data's index, 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.

From there, we record 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 use as an indicator of how well-distributed (and not biased towards 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 Figure 13 and Figure 15.

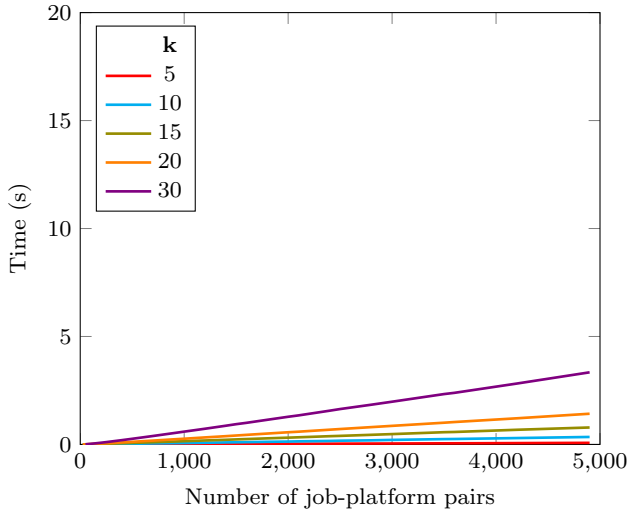
Looking at Figure 13 (with a clearer view in Figure 14), the size of the largest collision decreases significantly from $d = 1$ to $d = 2$, followed by a slower decrease at $d = 3$, before stagnating mostly between $d = 4$ and 6. Then we see another marginal decrease at $d = 7$. This means that the biggest precision gains we see lie between $d = 1$ and $d = 3$, with a relative gain starting from $d = 7$ onwards.

Also, Figure 15 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 conduct two experiments on the Constrained Job Seeker problem, using the exact same setting as the previous Unconstrained Job Seeker problem qualitative experiments: same seekers, same jobs J and platforms P of interest. The experiments themselves are very similar to their unconstrained



(a) ILP time wrt. N



(b) DP wrt. N

Fig. 11: Comparing performance of the ORTools solver (ILP) to the proposed Dynamic Programming algorithm (DP) algorithm

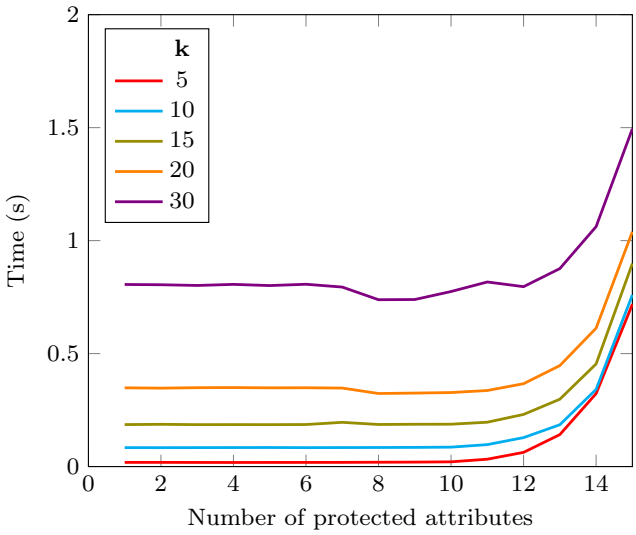


Fig. 12: DP algorithm runtimes wrt. the number of protected attributes n

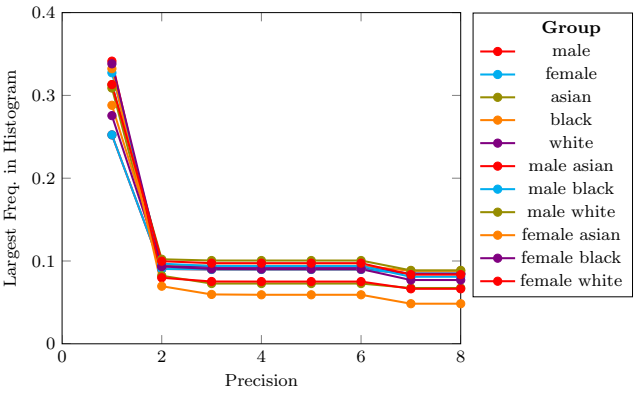


Fig. 13: Variation in largest frequency observed wrt. precision used (lower is better)

counterparts, with the only difference being that here, each job-platform pair is associated with a reward value between 1 and 100, and now each seeker aims to select the top-5 pairs that maximize the fairness values they get, while having a total reward of at least 400.

The goal of the two experiments is to confirm whether our newly-added reward constraint is actually affecting the obtained top-k results, which would demonstrate the effectiveness of our proposed algorithm.

For the first experiment, we use the same seekers as the first Unconstrained Job Seeker problem qualitative experiment run, and find the top-5 job-platform pairs for each seeker while satisfying the new reward threshold of 400. Then, we record the number of times each world (i.e., platform) and job occurs in every seeker's resultset, as shown in the plots of Figures 16 and 17. Also shown are the sum of (four-digit) fairness values and the sum of rewards for each resultset, which can be seen in Table 2.

Looking at Figures 16 and 17, we notice that for both plots, the results shown are different from those of the corresponding Unconstrained Job Seeker problem run, even though both experiments share the exact same setting aside from the reward threshold. This indicates that the latter is actively affecting results. Also, the values in Table 2 confirm that the reward constraint is indeed met, while still providing satisfactory fairness values.

Next, for the second experiment, we use again the same sets of seekers, jobs and platforms as we did previ-

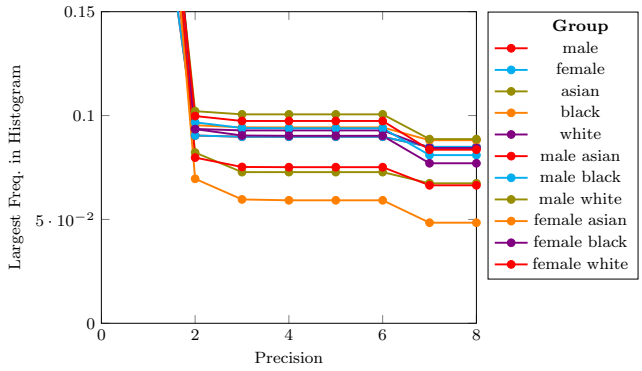


Fig. 14: A closer look of the plot in Figure 13

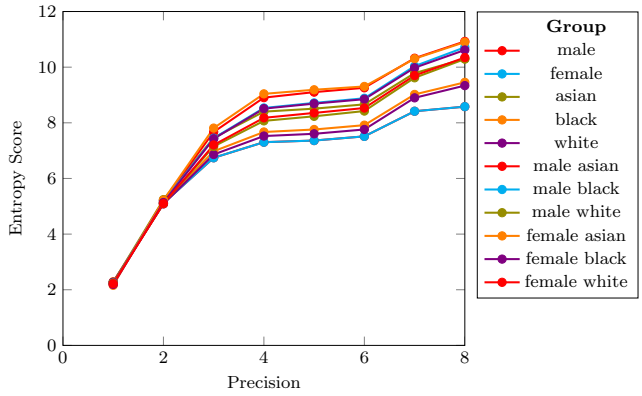


Fig. 15: Variation in entropy wrt. precision (higher is better)

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

Table 2: Sums of fairness values and rewards of the top-k pairs chosen. As expected, the sum of rewards for each seeker is indeed over 400

ously. From there, we find for each seeker and each platform p_i in P, the top-five jobs in platform p_i that maximize fairness, while satisfying the reward constraint. Then, we similarly find the seeker's (constrained) top-five pairs for all platforms in P combined. Finally, we record the sum of fairness values for each obtained result set as shown in Figure 18, and compare 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.

4.3 Job Provider Experiments

4.3.1 Algorithms Implementation

For the Job Provider with Global Budget problem variant, we implemented the proposed Dynamic Programming algorithm in C++. For the Local Budget variant, we implemented the algorithms in C++ as well, except for the Posta et al.'s algorithm, whose C code was taken from the authors' GitHub repository[5], and Karabakal et al.'s algorithm, for which we used the C code from the technical report in [14].

All scalability experiments were run on the same computer, an Apple MacBook Pro with a 2.3 GHz dual-core Intel Core i5 processor. Solving times are again measured in CPU time.

4.3.2 Job Provider with Global Budget Scalability Experiments

To assess the scalability of our proposed algorithm, we first create problem instances as follows. For given values of $|J|$ and $|P|$, and for a fixed number of protected attributes $n = 2$, we generate $N = |P| \times |J|$ job-platform pairs. Each pair is associated with $2^n - 1 = 3$ fairness values, selected at random between 1000 and 9999, and one cost value selected at random between 50 and 150. The task is then to find the subset of job-platform pairs with the highest fairness, while respecting a budget limit of $50 \times |J|$.

So, for increasing values of $|J|$ and $|P|$, we create 100 such problem instances per (J, P) combination. This time we went for a 100 instances instead of ten, because the solving time of this algorithm is very short and prone to slight fluctuations, so averaging time over more instances is needed to have a stable reading. Next, each of the instances is solved using two methods: 1) our proposed DP algorithm; and 2) the Google ORTools ILP sovler. The average solving time of each method over the 100 instances is then recorded. The results are shown in Figure 19, which reveals that the DP solving times are faster than the ILP times for all values of N, and that the DP times increase more slowly than the ILP times as N increases.

Next, as the budget limit B is part of the DP algorithm's time complexity, we design a second scalability experiment to see how solving times are affected by the value of B. For this, we fix the number of jobs and of platforms to $|J| = |P| = 50$, and generate 100 instances worth of fairness values and cost values in the same way as the experiment above. Then, each instance (i.e., set

[5] https://github.com/postamar/gap-solver

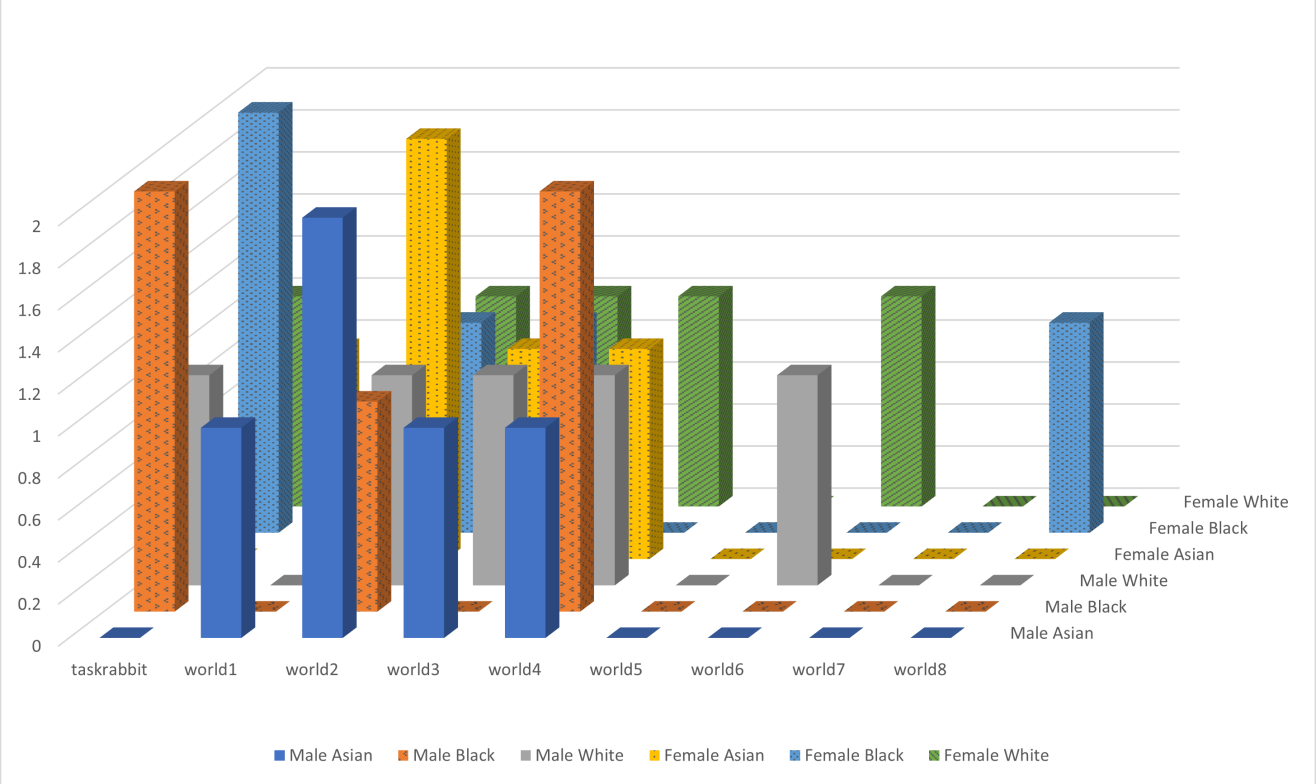


Fig. 16: Distribution of the top-5 pairs among worlds for each seeker

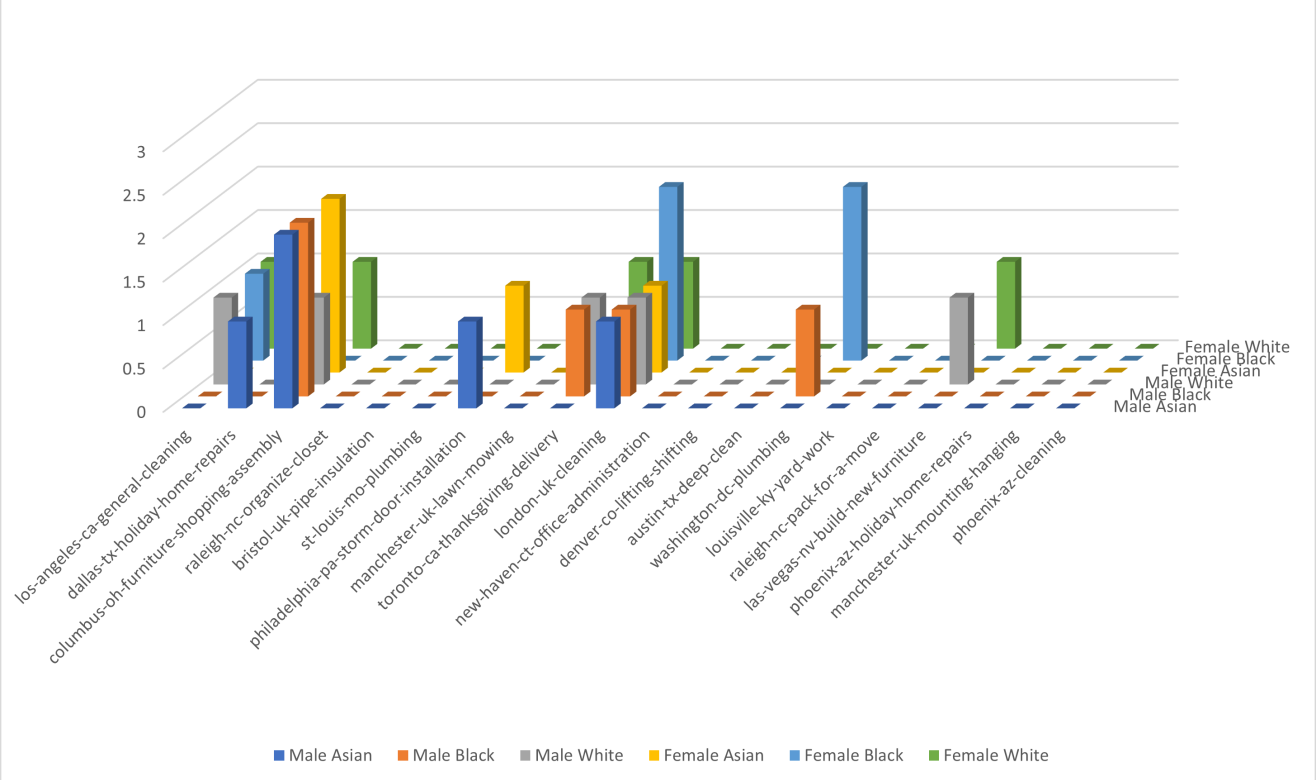


Fig. 17: Distribution of the top-5 pairs among jobs of interest for each seeker

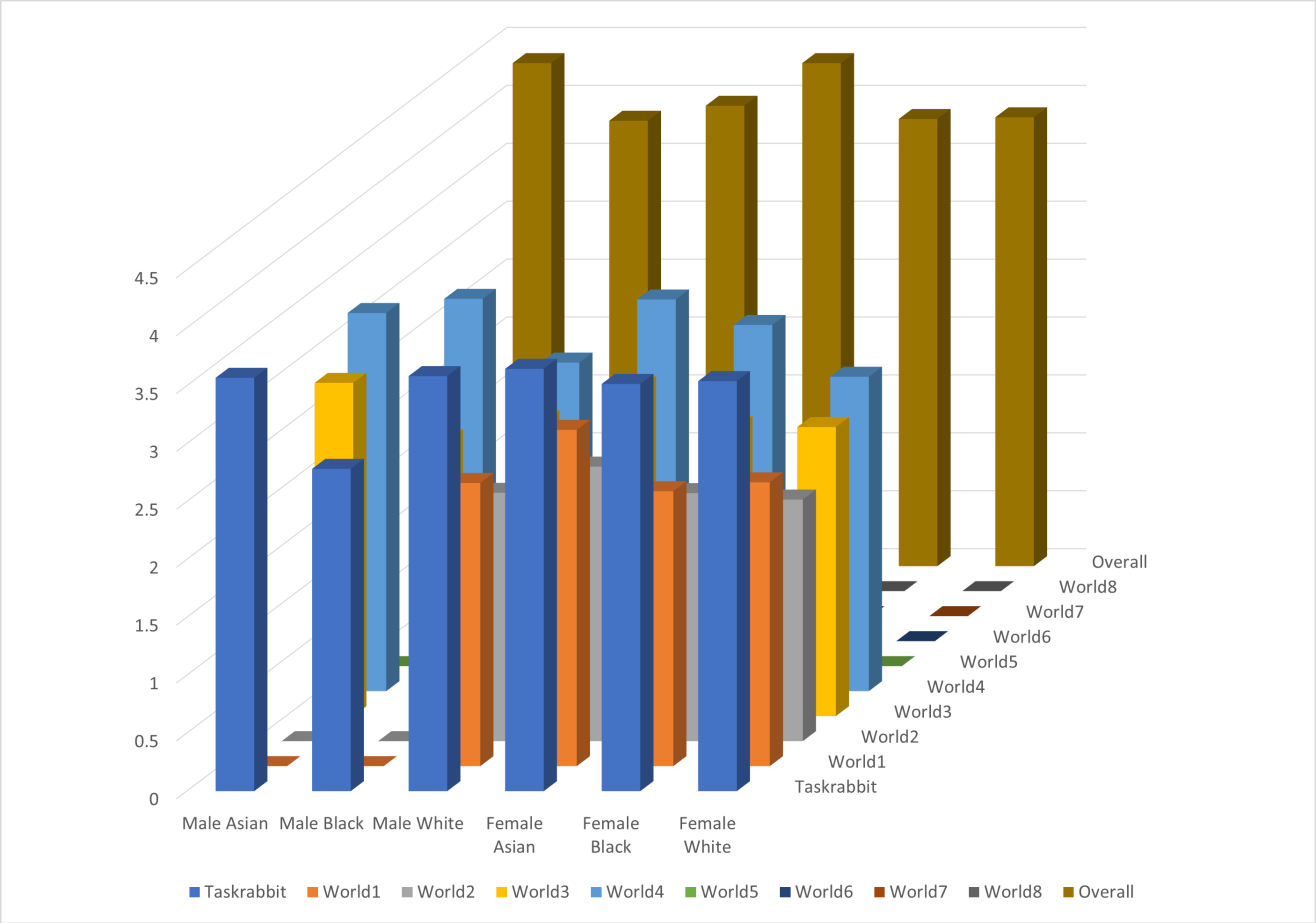


Fig. 18: Sums of fairness values of the top five (j, p) pairs per seeker and world

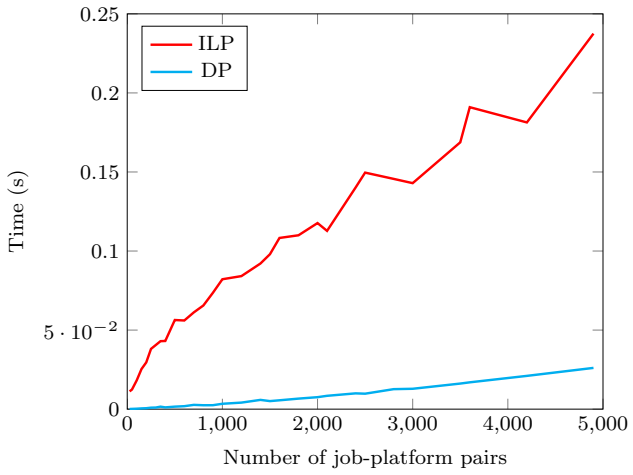


Fig. 19: DP algorithm vs. ILP solver scalability wrt. number of pairs N

of fairness and cost values) is solved with increasing values of B, by both the DP algorithm and the ORTools solver as a reference. Average running times are shown in Figure 20, hinting at a linear increase in the DP's solving time as B increases.

4.3.3 Job Provider with Global Budget Qualitative Experiments

We design two experiments in this section. The first aims to find to what extent the platforms chosen affect fairness results, for the same jobs of interest. For this, we take one job provider, and fix their jobs of interest to 20 jobs selected at random. Each of the 20 jobs is assigned a cost, selected as a random integer between 50 and 150. From there, for each alternative world p_i, we solve the Job Provider with Global Budget problem for the platform p_i, the selected 20 jobs, and a budget limit of 1000. We then do the same but with all alternative worlds combined. The optimal fairness value found for each instance is recorded, and displayed in the plot of Figure 21. (Note that unlike the Job Seeker problems, the plot here is two-dimensional, since the "seeker" dimension is not relevant for the Job Provider problems). The plot shows that the optimal fairness

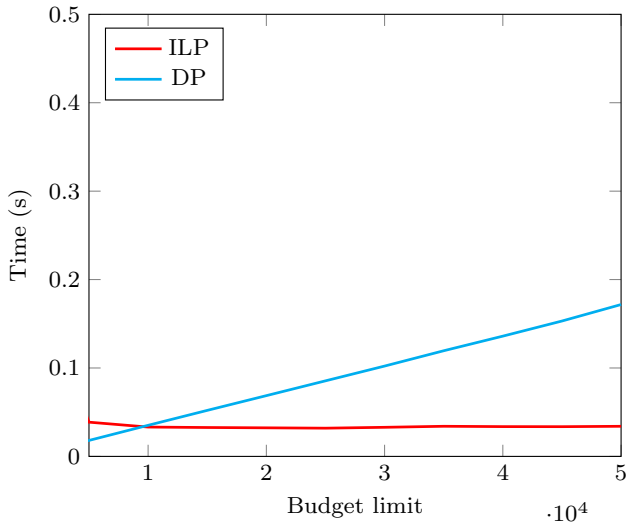


Fig. 20: DP algorithm vs. ILP solver scalability wrt. budget limit B

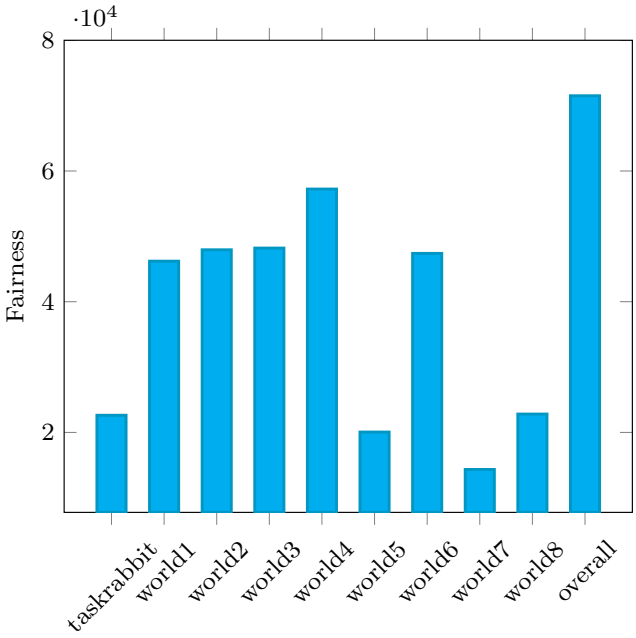


Fig. 21: Optimal fairness value obtained per platform(s) of interest

value found is not the same for each platform, and that choosing all platforms together (the "Overall" entry in the plot) yields a much better fairness value than any of the nine platforms separately. Thus, our conclusion here is two-fold: first, the best fairness achievable varies from platform to platform, so choosing a platform of interest wisely is important; and second, higher fairness values are achievable when choosing multiple platforms of interest instead of just one.

The second experiment aims to answer the question: *does a higher budget limit necessarily imply better fairness results?* For this, we take again one job provider, with the same 20 jobs of interest as the previous experiment, and we fix the provider's platforms of interest to be all nine alternative worlds. From there, we solve the Job Provider with Global Budget problem for this provider, with the same jobs and the same platforms of interest, but with budget limits varying between 1000 and 2000. For each budget limit considered, the optimal fairness value found is recorded and displayed in Figure 22. As shown by the plot, the obtained fairness value 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 job-to-platform assignment with the highest fairness possible has a cost of 1204. Thus, any input budget limit greater than 1204 will return this optimal assignment, with no further improvement possible on the fairness value obtained. Therefore, the answer to our question above is yes, a higher budget limit can imply better fairness results, but only up to a certain point.

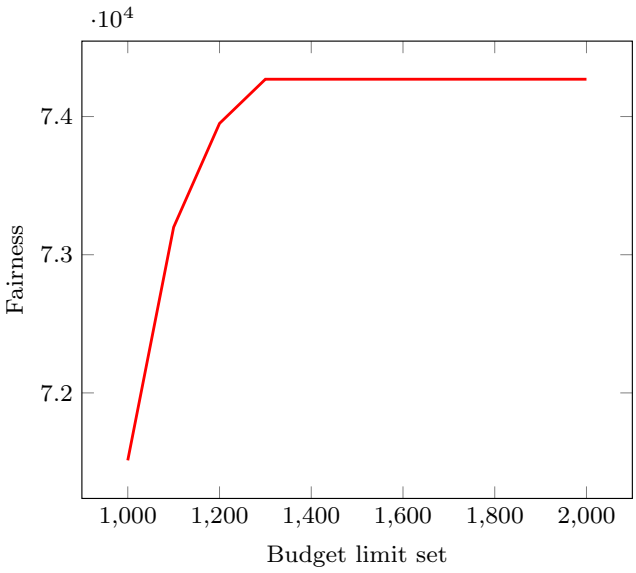


Fig. 22: Optimal fairness value obtained for the same problem instance, but with varying budget limits

4.3.4 Job Provider with Local Budgets Scalability Experiments

Recall that for this problem, we are comapring a selection of both exact and heuristic algorithms from the literature. To compare these algorithms for solving our problem, we develop a scalability experiment as follows. For increasing values of $|J|$ and $|P|$, and for a fixed number of attributes $n = 2$, we generate 100 problem

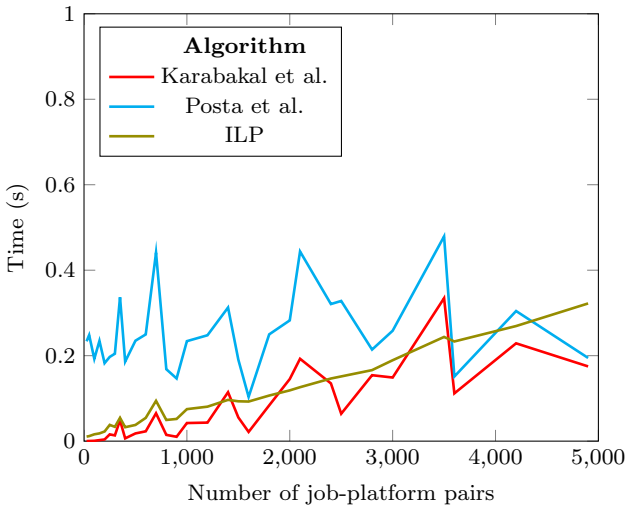


Fig. 23: Exact algorithms' runtimes wrt. N (number of pairs)

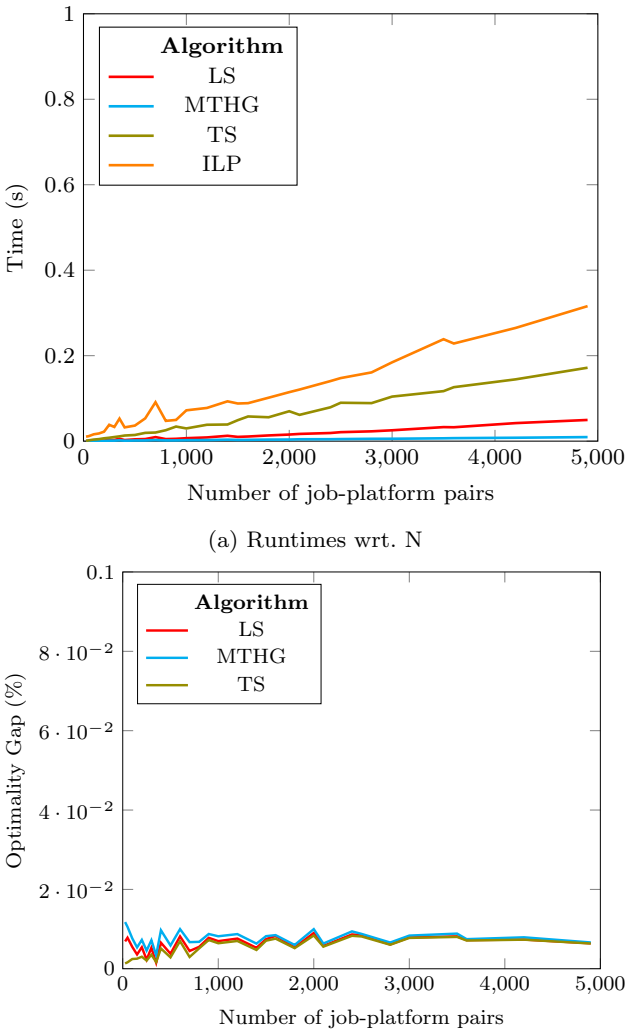


Fig. 24: Heuristic algorithms' runtimes and performance wrt. N (number of pairs)

instances as follows. First, we assign to each job in J a set of $2^n - 1 = 3$ fairness values, selected at random between 1000 and 9999, and a cost value selected at random between 50 and 149. Next, each platform p in P is assigned a budget limit b_p, where:

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

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. The goal is then to solve these problem instances using each of the algorithms considered, as well as a generic ILP solver (ORTools), while recording each method's solving times and optimality gaps.

Starting with exact algorithms, the three methods we are comparing are:

- The BB algorithm by Fisher et al. [9], following the pseudo-code in [18];
- 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 defaults.
- 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 leave other parameters at their default values.

For the three algorithms, the experiment is run on the same problem instances. Early on in the tests, we notice the Fisher et al. algorithm's solving times to be substantially slower than for the other two algorithms, despite our best efforts at optimizing our code. Therefore, this algorithm was dropped from the rest of our benchmark. The time results of the benchmark for the remaining algorithms versus ORTools can be found in Figure 23. In the absolute, both Karabakal et al.'s and Posta et al.'s algorithms scale fairly well with our problems' sizes, with Karabakal et al.'s method having a slight edge, however neither of them being much faster than the ILP solver.

Next, we move on to heuristic algorithms. The heuristic algorithms we compared are:

- MTHG
- Osman's LS Descent method (LS)
- Osman's Tabu Search method (TS)

For the Tabu Search method, the iteration limit was set to $100 \times |J|$, the tabu list size to $20 \times |J|$; all other parameters for all the algorithms were kept to their defaults. The three algorithms are then compared to each other and to ORTools based on solving time, and also based on solution quality this time. Here, solution quality is computed as the gap between the optimal value found by a heuristic, z_h, and the one found by ORTools, z_{opt} (which is assumed to be optimal), via the following formula:

$$optimality\_gap_h = \begin{cases} \frac{z_{opt} - z_h}{z_{opt}} & \text{if } z_{opt} \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The results are shown in Figure 24, revealing that all three heuristic approaches perform much faster than the ILP solver at scale, while returning decently accurate solutions within 2% from optimality on average. Therefore, if exact solution is not a must, then heuristics can be a solid, more efficient alternative to exact algorithms.

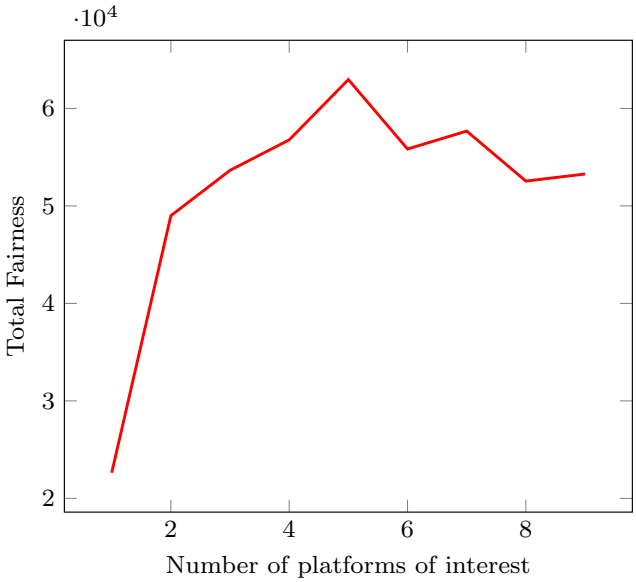


Fig. 25: Optimal fairness value obtained with different number of platforms of interest

4.3.5 Job Provider with Local Budgets Qualitative Experiments

We design two experiments in this section. The first one aims to find, for a given (total) budget limit, whether higher fairness values are achievable with fewer platforms (but with higher budget limits each), or with more platforms (and lower budget limits each). For this, we take the same nine worlds and twenty jobs as in Section 4.3.3. The twenty jobs are fixed as jobs of interest, and the fairness values for each job-platform pair are kept the same as in Section 4.3.3. We also fix a (total) budget limit of 1000, again like in Section 4.3.3's first experiment. The idea is then to vary the number of platforms $|P|$ of interest, and divide the budget limit evenly across these platforms (if the total limit is not divisible by $|P|$, then the remainder amount after division is added to the last platform). We run nine runs for this experiment: in the first run, we have $P = \{taskrabbit\}$ as 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 is solved using the Karabakal et al. algorithm [15], and the total fairness value of the optimal assignment is recorded.

The results of this experiment are displayed in Figure 25. The plot shows an apparent trade-off: at first, fairness values generally increase, as we get more options (job-platform pairs) to choose 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 value. Therefore, using our framework, the answer to the question above is that choosing the right number of platforms poses a trade-off, that should be handled on a case-by-case scenario.

Our second experiment aims to find the extent to which a platform of interest can affect the obtained fairness values. For this, we reuse the same setup as the first experiment, but this time at each run, only one platform is selected individually. 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 platforms combined are selected. Results are shown in Figure 26. As we can see, the fairness value chosen does vary from platform to platform, with all things remaining constant, which implies that choosing a platform of interest must be done wisely.

Also, when comparing these results with those of the equivalent experiment for the Global Budget variant (Section 4.3.3 as shown in Figure 21.), we see that they are all identical, except for the last run (where all platforms are combined). This is because for one platform, both the Global and the Local Budget Job Provider problems are equivalent, and thus their algorithms return the same results. For the last run, the fairness values obtained in the local budget experiment are lower, since the constraints are tighter compared to the global budget one. While the *total* budget of 1000 is the same, the local budget variant has additional constraints on how costs should be distributed over all platforms.

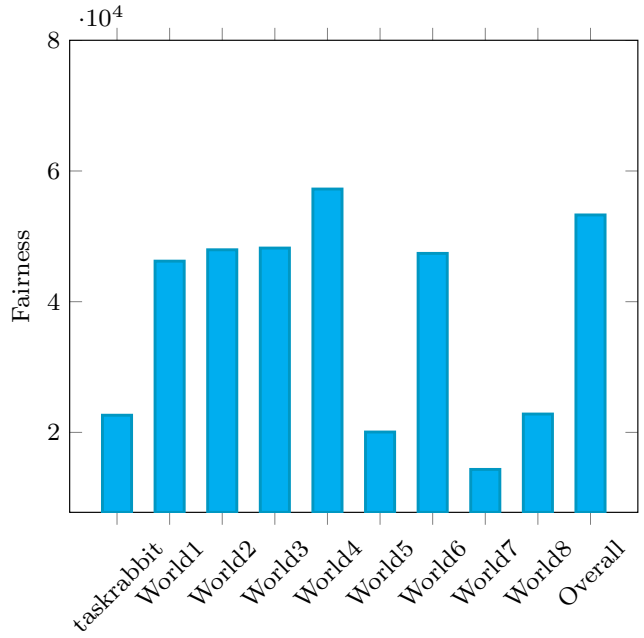


Fig. 26: Optimal fairness value obtained per platform(s) of interest

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 at least NP-hard. As shown by our experiments, the algorithms we proposed for all four problems are efficient, and answer useful fairness-related inquiries. Our framework does not assume any particular notion of fairness, and can thus be used with any group fairness notion or quantification method.

Possible future work includes 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 standalone service on top of existing online labor platforms.

References

1. Amer-Yahia, S., Elbassuoni, S., Ghizzawi, A., Borromeo, R., Hoareau, E., Mulhem, P.: Fairness in online jobs:{A} case study on taskrabbit and google. In: International Conference on Extending Database Technologies (EDBT) (2020)
2. Biega, A.J., Gummadi, K.P., Weikum, G.: Equity of attention: Amortizing individual fairness in rankings. In: The 41st international acm sigir conference on research & development in information retrieval, pp. 405–414 (2018)
3. Calders, T., Verwer, S.: Three naive bayes approaches for discrimination-free classification. Data Mining and Knowledge Discovery **21**(2), 277–292 (2010). DOI 10.1007/s10618-010-0190-x. URL `https://doi.org/10.1007/s10618-010-0190-x`
4. Celis, L.E., Straszak, D., Vishnoi, N.K.: Ranking with fairness constraints. arXiv preprint arXiv:1704.06840 (2017)
5. Chen, L.: Measuring algorithms in online marketplaces. Ph.D. thesis, Northeastern University (2017)
6. Elbassuoni, S., Amer-Yahia, S., Ghizzawi, A.: Fairness of scoring in online job marketplaces. ACM Transactions on Data Science **1**(4), 1–30 (2020)
7. Elbassuoni, S., Amer-Yahia, S., Ghizzawi, A., Atie, C.: Exploring fairness of ranking in online job marketplaces. In: 22nd International Conference on Extending Database Technology (EDBT) (2019)
8. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of computer and system sciences **66**(4), 614–656 (2003)
9. Fisher, M.L., Jaikumar, R., Van Wassenhove, L.N.: A multiplier adjustment method for the generalized assignment problem. Management science **32**(9), 1095–1103 (1986)
10. Geyik, S.C., Ambler, S., Kenthapadi, K.: Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In: Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining, pp. 2221–2231 (2019)
11. Ghizzawi, A., Marinescu, J., Elbassuoni, S., Amer-Yahia, S., Bisson, G.: Fairank: An interactive system to explore fairness of ranking in online job marketplaces. In: 22nd International Conference on Extending Database Technology (EDBT) (2019)
12. Hannák, A., Wagner, C., Garcia, D., Mislove, A., Strohmaier, M., Wilson, C.: Bias in online freelance marketplaces: Evidence from taskrabbit and fiverr. In: Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing, pp. 1914–1933 (2017)
13. Jahanbakhsh, F., Cranshaw, J., Counts, S., Lasecki, W.S., Inkpen, K.: An experimental study of bias in platform worker ratings: The role of performance quality and gender. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1–13 (2020)
14. Karabakal, N.: A c code for solving the generalized assignment problem. Tech. rep. (1992)
15. Karabakal, N., Bean, J.C., Lohmann, J.R.: A steepest decent [sic] multiplier adjustment method for the generalized assignment problem. Tech. rep. (1993)
16. Keane, M.T., O'Brien, M., Smyth, B.: Are people biased in their use of search engines? Communications of the ACM **51**(2), 49–52 (2008)
17. Lagoudakis, M.G.: The 0-1 knapsack problem–an introductory survey (1996)
18. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc., USA (1990)
19. Osman, I.H.: Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. Operations-Research-Spektrum **17**(4), 211–225 (1995)

20. Posta, M., Ferland, J.A., Michelon, P.: An exact method with variable fixing for solving the generalized assignment problem. Computational Optimization and Applications **52**(3), 629–644 (2012)
21. Rosenblat, A., Levy, K.E., Barocas, S., Hwang, T.: Discriminating tastes: Uber's customer ratings as vehicles for workplace discrimination. Policy & Internet **9**(3), 256–279 (2017)
22. Singh, A., Joachims, T.: Fairness of exposure in rankings. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2219–2228 (2018)
23. Yagiura, M., Ibaraki, T.: Generalized assignment problem. In: T.F. Gonzalez (ed.) Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series). Chapman & Hall/CRC (2007)
24. Zehlike, M., Bonchi, F., Castillo, C., Hajian, S., Megahed, M., Baeza-Yates, R.: Fa* ir: A fair top-k ranking algorithm. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 1569–1578 (2017)
25. Zehlike, M., Castillo, C.: Reducing disparate exposure in ranking: A learning to rank approach. In: Proceedings of The Web Conference 2020, pp. 2849–2855 (2020)
26. Zehlike, M., Sühr, T., Baeza-Yates, R., Bonchi, F., Castillo, C., Hajian, S.: Fair top-k ranking with multiple protected groups. Information Processing & Management **59**(1), 102,707 (2022)
27. Zliobaite, I.: A survey on measuring indirect discrimination in machine learning. CoRR **abs/1511.00148** (2015). URL `http://arxiv.org/abs/1511.00148`

Appendix A Proof of Theorem 1

Theorem 1 *The Constrained Job Seeker problem is polynomial-time reducible to the optimization variant of the Knapsack problem and is therefore at least as hard.*

Proof Note that by having only one group and one platform, the 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.

Shady: slightly confused by the notion of J_i being a pair. Anis: Renamed J to M, j_i to m_i. Running out of letters...

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 ($s \leq n$) of total value X and total weight $W_A \leq W$. 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 $n - s \leq n$ 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,

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

Assume now that M has a subset B of size $k = n$ of total fairness X and total reward $R_B \geq R$. 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 $R_{B'} = R_B - sW \geq (n - s - 1)W$. 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.

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

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

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