

AMERICAN UNIVERSITY OF BEIRUT

A FRAMEWORK TO MAXIMIZE GROUP
FAIRNESS FOR WORKERS ON ONLINE
LABOR PLATFORMS

by

ANIS SAMI EL RABAA

A thesis

submitted in partial fulfillment of the requirements
for the degree of Master of Science
to the Department of Computer Science
of the Faculty of Arts and Sciences
at the American University of Beirut

Beirut, Lebanon
January 2022

AMERICAN UNIVERSITY OF BEIRUT

A FRAMEWORK TO MAXIMIZE GROUP
FAIRNESS FOR WORKERS ON ONLINE
LABOR PLATFORMS

by
ANIS SAMI EL RABAA

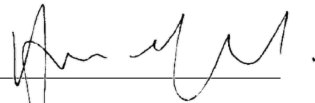
Approved by:



Dr. Shady Elbassuoni, Associate Professor

Advisor

Computer Science



Dr. Amer Abdo Mouawad, Assistant Professor

Member of Committee

Computer Science

Dr. Wassim El Hajj, Associate Dean

Member of Committee

Computer Science



Date of thesis defense: January 25, 2022

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

Student Name:

El Rabaa	Anis	Sami
Last	First	Middle

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of my thesis; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes

- As of the date of submission of my thesis
- After 1 year from the date of submission of my thesis .
- After 2 years from the date of submission of my thesis .
- After 3 years from the date of submission of my thesis .



31/01/2022

Signature

Date

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis advisor, Professor Shady Elbassuoni, for all his helpful advice, support and massive patience with me throughout this thesis journey. It was an honor to work under his supervision, and I would not have it any other way. I would also like to thank Professor Amer Abdo Mouawad for his helpful assistance, answering my questions relating to complexity theory, as well as accepting to serve on my thesis committee. I also thank Professor Wassim El Hajj, who accepted to serve on my thesis committee and provided insightful remarks that helped improve my work. Thanks as well to Jihad Hanna for the helpful insight on complexity theory and dynamic programming.

Last but absolutely not least, I would like to thank everyone who happily supported me throughout this journey, from parents to family members, friends, classmates and professors. Thanks to everyone who cheered me up when feeling low or stuck, to everyone who poked my interest and opened my eyes to a certain field of knowledge, to everyone who pushed me and challenged me to achieve more, to everyone who did not forget me in their thoughts and prayers. I cannot imagine how I would have made it this far without every single one of you.

ABSTRACT OF THE THESIS OF

Anis Sami El Rabaa for Master of Science
Major: Computer Science

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

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). In this thesis, 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.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	1
ABSTRACT	2
1 INTRODUCTION	9
1.1 Motivation	9
1.2 Objectives and Contributions	10
1.3 Thesis Outline	13
2 LITERATURE REVIEW	14
3 FRAMEWORK	17
3.1 Maximizing Fairness for Job Seekers	19
3.2 Maximizing Fairness for Job Providers	28
4 EXPERIMENTS	34
4.1 Semi-Synthetic Dataset (for Qualitative Experiments)	35
4.2 Job Seeker Experiments	36
4.2.1 Algorithms Implementation	36
4.2.2 Unconstrained Scalability Experiments	37
4.2.3 Unconstrained Qualitative Experiments	41

4.2.4	Constrained Scalability Experiments	45
4.2.5	Constrained Qualitative Experiments	47
4.3	Job Provider Experiments	54
4.3.1	Algorithms Implementation	54
4.3.2	Job Provider with Global Budget Scalability Experiments	54
4.3.3	Job Provider with Global Budget Qualitative Experiments	56
4.3.4	Job Provider with Local Budgets Scalability Experiments .	60
4.3.5	Job Provider with Local Budgets Qualitative Experiments	63
5	CONCLUSION	66
	Bibliography	67

ILLUSTRATIONS

3.1	Sample Bipartite Graph for the Unconstrained Job Seeker Problem	18
3.2	Sample Bipartite Graph for the Constrained Job Seeker Problem. Note the addition of a reward r for each job-platform pair	25
3.3	Sample 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	28
4.1	Unconstrained Job Seeker Experiment: Naive vs. Top-k performance for different values of k	39
4.2	Unconstrained Job Seeker Experiment: Naive vs. Top-k Times for $k = 20$	39
4.3	Top-k algorithm times wrt. Number of Protected Attributes, n	40
4.4	Occurrences of each World in the Seekers' Top-5 Results	41
4.5	Occurrences of each Job in the Seekers' Top-5 Results	42
4.6	Sum of fairness values of the top-five (j, p) pairs, per seeker	44
4.7	Comparing worker counts in the twenty selected jobs: <i>world2</i> vs. <i>world7</i>	45

4.8	Constrained Job Seeker: Comparing performance of the ORTools solver (ILP) versus the proposed Dynamic Programming algorithm (DP)	46
4.9	DP algorithm times wrt. Number of Protected Attributes, n	47
4.10	Variation in Largest Frequency Observed wrt. Precision used (lower is better)	49
4.11	A closer look from the plot of Figure 4.10	50
4.12	Variation in Entropy wrt. Number of Precision Digits (higher is better)	50
4.13	Constrained Experiment: Distribution of top-5 pairs among worlds, for each seeker	52
4.14	Constrained Experiment: Distribution of top-5 pairs among jobs of interest, for each seeker	52
4.15	Constrained Experiment: Sums of fairness values of the top five (j, p) pairs, per seeker and world.	53
4.16	Job Provider with Global Budget: DP algorithm vs. ILP solver scalability wrt. number of pairs N	55
4.17	Job Provider with Global Budget: DP algorithm vs. ILP solver scalability wrt. budget limit B	56
4.18	Job Provider with Global Budget: First qualitative experiment; Optimal fairness value obtained per platform(s) of interest.	58
4.19	Job Provider with Global Budget: Second qualitative experiment; Optimal fairness value obtained for the same problem instance, but with varying budget limits.	59
4.20	Job Provider with Local Budgets: Exact algorithms' time performance wrt. N (number of pairs)	61

4.21 Job Provider with Local Budgets: Heuristic algorithms' time and quality performance wrt. N (number of pairs) 62

4.22 Job Provider with Local Budgets: First qualitative experiment; Optimal fairness value obtained with different number of platforms of interest. 64

4.23 Job Provider with Local Budgets: First qualitative experiment; Optimal fairness value obtained per platform(s) of interest. 65

TABLES

4.1	Platform statistics for the alternative worlds (in percentages) . . .	36
4.2	Constrained Experiment: 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.	53

CHAPTER 1

INTRODUCTION

1.1 Motivation

Online labor platforms such as TaskRabbit and Upwork are gaining popularity as mediums 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 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 might 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 can commonly happen since such ranking might depend on the ratings of workers on the platform and the number of their past jobs, both of which can perpetuate bias against certain groups of workers.

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.

1.2 Objectives and Contributions

In this thesis, 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 [1, 2], 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 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. More precisely, we assume that a job j for worker group g on platform p is associated with a fairness value $f(j, p, g)$. Without loss of generality, we assume that $f(j, p, g)$ is a value between 0 and 1, and that the higher the value is, the more fair job j is considered for group g on platform p .

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. More precisely, our first and second optimization problems aim to maximize worker fairness for job seekers. Given a set of worker 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 in the first optimization problem we propose 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 . We also consider the case where jobs are associated with rewards. That is, we assume that each job j available on platform p is associated with a reward $r(j, p)$. 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.

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 j is associated with a cost $c(j, p)$ on a platform p it is available

on, and that this cost differs from one platform to the other. Given a set of jobs J to be deployed on a set of platforms P and a budget B , our goal in the third optimization problem 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. A slight variation of this optimization problem is our fourth and final optimization problem we define. 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 to each platform p does not exceed its budget b_p , and the total worker fairness of the assigned jobs is maximized.

We prove that three of our four optimization problems are computationally hard, and propose algorithms to solve all four of them in reasonable time. 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 and number of platforms increase and to compare them to suitable baseline ones. 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.

To the best of our knowledge, our framework is the first to address the problem of finding the fairest jobs on multiple platforms for job seekers with and without reward constraints. Our framework is also the first to address the problem of deploying multiple jobs on multiple platforms such that worker fairness is maximized with different budget constraints.

Our main contributions in this thesis can thus be summarized as follows:

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

1.3 Thesis Outline

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

CHAPTER 2

LITERATURE REVIEW

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 [3], 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 [4] 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 [5].

To address fairness of ranking in such platforms, various methods have been proposed to actively generate fair rankings. Many of them are *post-processing* methods (e.g., [6, 5, 7, 8]), 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 [9].

In contrast, other notable works focused on assessing fairness of a worker ranking *in its current state*, rather than trying to adjust it. For instance, the authors in [10] 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 [11], 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 [12, 13, 14, 15]. In [12, 13, 14], the authors formulated an optimization problem to find the partitioning of workers (based on their protected attributes) that exhibits the highest unfairness based on a given scoring function. They used Earth Mover’s Distance (EMD) between score distributions as a measure of unfairness. In [16], 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.

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 have 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.

CHAPTER 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

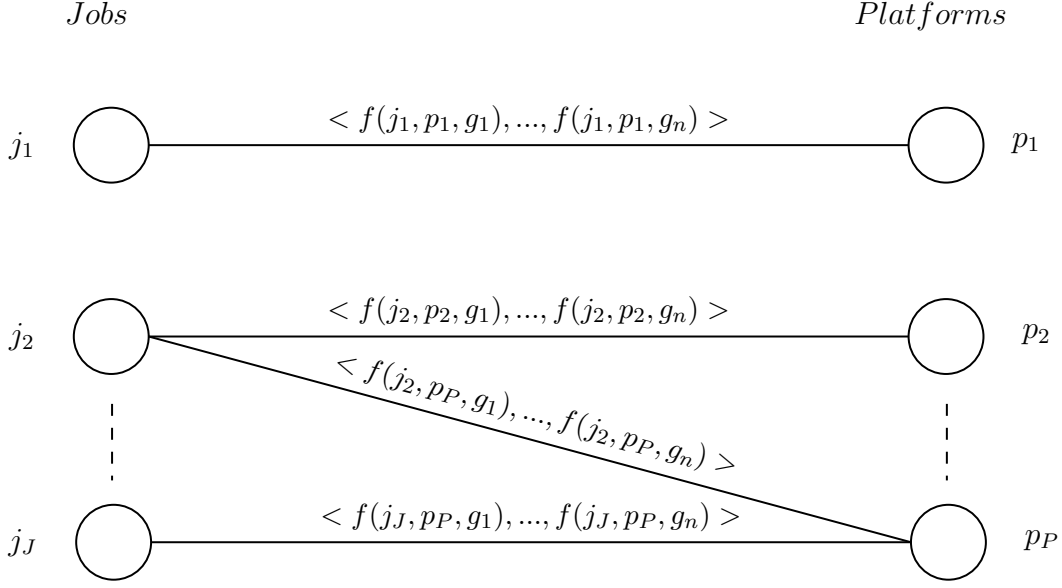


Figure 3.1: Sample Bipartite Graph for the Unconstrained Job Seeker Problem

into our framework. In our experiments, we make use of the framework in [16], 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 . That is, we assume that not all jobs are available on all platforms and that not all worker groups are available on all platforms. 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 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 . To better illustrate, a sample of such graph is shown in Figure 3.1.

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 could 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. 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 . We formulate this goal as the following optimization problem.

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:*

$$\begin{aligned} & \operatorname{argmax}_S \sum_{(j,p) \in S} \min_{g \in G \wedge e(j,p,g)=\text{true}} f(j,p,g) \\ \text{subject to: } & S \subseteq J \times P \\ & a(j,p) = \text{true} \quad \forall (j,p) \in S \\ & |S| = k \end{aligned}$$

Since each job seeker belongs to different worker groups, we need to aggregate the different fairness values for each group the job seeker belongs to in order to obtain a single fairness value for a job-platform pair. In the optimization problem above, we use minimum as an aggregation operator. This means that for each job-platform (j, p) pair, the aggregated fairness value would be equal to the minimum of the fairness values of job j for all groups that the job seeker belongs to on platform p . Thus, we take a conservative worst-case approach here to quantify the fairness value of a job-platform pair for a given job seeker. Of course other aggregation methods 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 naïve 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 over all groups G the job seeker belongs to. It then returns the k job-platform pairs with the highest minimum fairness over all groups G . We assume that the fairness for a group g and a job j on platform p is precomputed and denoted by $f(j, p, g)$.

A more efficient approach can make use of optimal aggregation algorithms such as Fagin’s algorithm [17] 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 G . 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.

We also consider the case where jobs are associated with rewards. That is, we assume that each job j available on platform p is associated with a reward $r(j, p)$. Thus, each edge in our bipartite graph would include an additional weight representing the reward of a job j on platform p , as shown in Figure 3.2. In this case, the goal of the job seeker is to find the top- k fairest jobs such that their total reward is above a certain threshold. This goal 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:*

Algorithm 1 Top-k Job Seeker Algorithm

```
1: Input: a set of jobs  $J$ , a set of platforms  $P$ , a set of groups  $G$ ,  $k$ 
2: output: the  $k$   $(j, p)$  pairs with the highest minimum fairness over all groups  $G$ 
3:  $topk \leftarrow minHeap()$  ▷ Initialization
4:  $cursor \leftarrow 0$ 
5: while  $topk.minValue() < \tau$  or  $topk.size() < k$  do
6:    $\tau \leftarrow -\infty$ 
7:   for  $g \in G$  do
8:      $((j, p), f(j, p, g)) \leftarrow I_g.getEntry(cursor)$  ▷ Read entry at current line
      $(cursor)$ 
9:     if  $j \in J$  and  $p \in P$  then
10:      if  $\tau < f(j, p, g)$  then ▷ Update threshold value
11:         $\tau \leftarrow f(j, p, g)$ 
12:      end if
13:       $min \leftarrow f(j, p, g)$ 
14:      for  $g' \in G$  and  $g' \neq g$  do ▷ Perform random access on all other
      lists
15:        if  $e(j, p, g')$  is true then
16:           $f(j, p, g') \leftarrow I_{g'}.getValue((j, p))$ 
17:          if  $f(j, p, g') < min$  then
18:             $min \leftarrow f(j, p, g')$ 
19:          end if
20:        end if
21:      end for
22:      if  $topk.size() < k$  then ▷ Update top-k set (if needed)
23:         $topk.insert((j, p), min)$ 
24:      else
25:        if  $topk.minValue() < min$  then
26:           $topk.pop()$ 
27:           $topk.insert((j, p), min)$ 
28:        end if
29:      end if
30:    end if
31:  end for
32:   $cursor \leftarrow cursor + 1$ 
33: end while
34: return  $topk$ 
```

$$\begin{aligned} & \operatorname{argmax}_S \sum_{(j,p) \in S} \min_{g \in G \wedge e(j,p,g)=\text{true}} f(j,p,g) \\ & \text{subject to: } S \subseteq J \times P \\ & a(j,p) = \text{true} \quad \forall (j,p) \in S \\ & |S| = k \\ & \sum_{(j,p) \in S} r(j,p) \geq R \end{aligned}$$

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

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

Theorem 1. *The 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 [18], this theorem gives us a lower bound on the hardness of the CONSTRAINED JOB SEEKER problem.

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

to the following: Given a list J of pairs $j_i = (f_i, r_i)$, where f_i is the assigned fairness value and r_i the reward value, select k pairs such that fairness is maximized and the total reward is at least R . Using this version of the problem, we give a polynomial-time reduction from the optimization version of KNAPSACK. Given a list L of pairs $a_i = (v_i, w_i)$, where v_i represents the value of the pair and w_i its weight, and an integer W , the KNAPSACK problem asks for a subset of L of maximum value such that the total weight is at most W .

Given an instance of the KNAPSACK problem where $|L| = n$, create a list J of n pairs $j_i = (f_i, r_i)$ where $f_i = v_i$ and $r_i = W - w_i$. Moreover, add n additional pairs $(0, W)$ to J . Set $k = n$ and $R = (n - 1)W$. We prove 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 J contains a subset of size n with total fairness X , satisfying the CONSTRAINED JOB SEEKER constraints.

Assume L contains a subset A of 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 J by taking $\forall p_i \in A$ its equivalent $j_i \in J$, 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.

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

$$R_B = \sum_{j_i \in B} r_i = sW - \sum_{p_i \in A} w_i + (n - s)W = nW - W_A \geq nW - W = (n - 1)W = R$$

Assume now that J 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

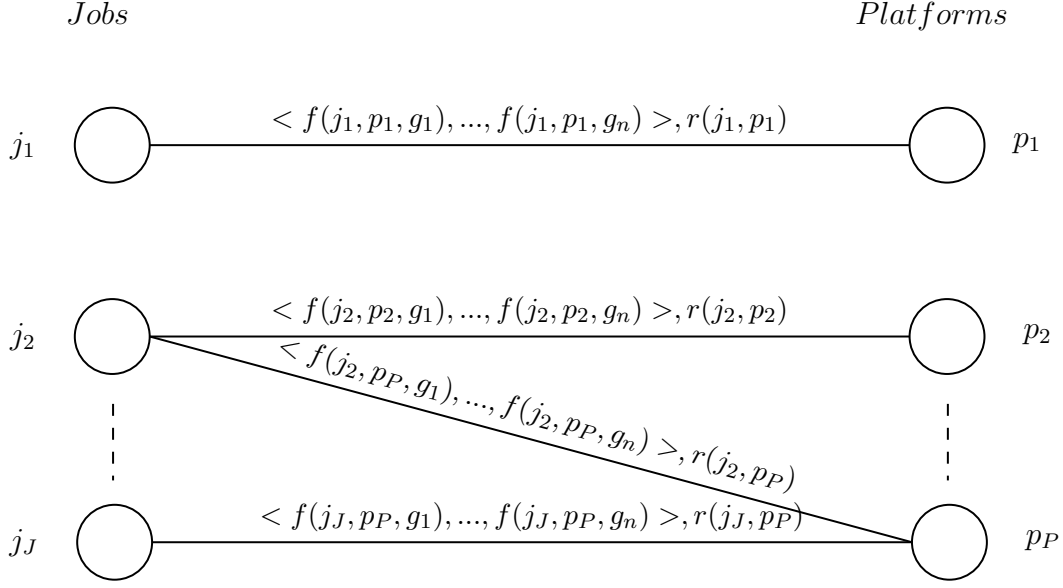


Figure 3.2: Sample Bipartite Graph for the Constrained Job Seeker Problem. Note the addition of a reward r for each job-platform pair

reward $R_{B'} = R_B - sW \geq (n - s - 1)W$. Construct the set $A = \{p_i : j_i \in B'\} \subseteq L$.

Let V_A denote the total value of A and W_A its total weight.

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

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

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

The next question that arises is how to solve this problem efficiently in practice, and the similarity with the KNAPSACK problem gives a nearly immediate dynamic programming (DP) solution that we describe below (Algorithms 2 and 3).

Algorithm 2 Constrained Job Seeker Algorithm

1: **Input:** A set of jobs J , a set of platforms P , a set of groups G , and two integers K and R .

2: **Output:** The K (j, p) pairs with the highest minimum fairness over all groups G having reward at least R . Running time is $\mathcal{O}(JPKR)$.

▷ Step 1: Initialization + aggregation of fairness values

3: $minFair[1..len(J)][1..len(P)] \leftarrow$ new 2D array initialized to $+\infty$.

4: **for** $j \in J$ and $p \in P$ and $g \in G$ **do**

5: **if** $e(j, p, g) = true$ **then**

6: $minFair[j][p] \leftarrow \min(minFair[j][p], f(j, p, g))$

7: **end if**

8: **end for**

9: $L \leftarrow$ Empty list

10: **for** $j \in J$ and $p \in P$ **do**

11: $(j, p, f, r) \leftarrow (j, p, minFair[j][p], r(j, p))$

12: $L.append((j, p, f, r))$

13: **end for**

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

14: $DP[0..len(L)][0..K][0..R] \leftarrow$ new 3D array initialized to -1 .

15: $choice[0..len(L)][0..K][0..R] \leftarrow$ new 3D array initialized to -1 .

16: $maxFairness \leftarrow \text{RECURSIVEMAXFAIRNESS}(1, L, K, R, DP, choice)$

17: **if** $maxFairness = -\infty$ **then return** ϕ

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

18: $i \leftarrow 0$, $result \leftarrow \phi$

19: **while** $i \neq len(L)$ **do**

20: **if** $choice[i][K][R] = 0$ **then**

21: $i \leftarrow i + 1$

22: **continue**

23: **end if**

24: $result.add((j, p))$

25: $K \leftarrow K - 1$

26: $R \leftarrow \max(0, R - L[i].r)$

27: $i \leftarrow i + 1$

28: **end while**

29: **return** $result$

Algorithm 3 Recursive Maximum Fairness Algorithm

```
1: procedure RECURSIVEMAXFAIRNESS( $i, L, K, R, DP, choice$ )
2:   if  $K = 0$  then return  $R = 0 ? 0 : -\infty$ 
3:   if  $i > N$  then return  $-\infty$ 
4:   if  $DP[i][K][R] \neq -1$  then return  $DP[i][K][R]$ 

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

7:   if  $dontTakePair = -\infty$  and  $takePair = -\infty$  then return  $DP[i][K][R] =$ 
    $-\infty$ 
8:   if  $dontTakePair \neq -\infty$  and  $takePair \neq -\infty$  then
9:      $choice[i][K][R] \leftarrow (dontTakePair < L[i].f + takePair)$ 
10:    return  $DP[i][K][R] \leftarrow \max(dontTakePair, L[i].f + takePair)$ 
11:  end if
12:  if  $dontTakePair \geq 0$  then
13:     $choice[i][K][R] \leftarrow 0$ 
14:    return  $DP[i][K][R] \leftarrow dontTakePair$ 
15:  end if
16:   $choice[i][K][R] \leftarrow 1$ 
17:  return  $DP[i][K][R] \leftarrow L[i].f + takePair$ 
18: end procedure
```

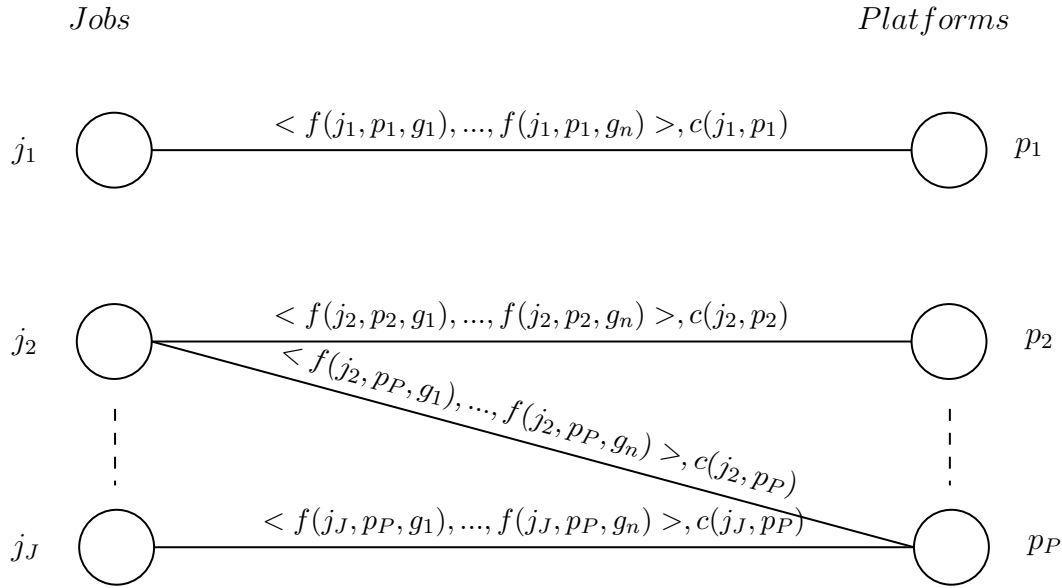


Figure 3.3: Sample Bipartite Graph for the Job Provider problem. In addition to the fairness values per group, each edge between a job j and a platform p has a weight $c(j, p)$ equal to the cost of deploying job j on platform p .

3.2 Maximizing Fairness for Job Providers

A job provider is a person looking to deploy a set of jobs on different platforms. We assume that each job j is associated with a cost $c(j, p)$ on a platform p it is available on, and that this cost differs from one platform to the other. This extends our bipartite graph from before so that each edge is now associated with an additional weight that represents the cost of deploying job j on platform p . A sample of such graph is depicted in Figure 3.3. 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(s). We assume that each job can be 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):*

$$\begin{aligned} & \max \sum_{j \in J} \sum_{p \in P} \min_{g | e(j,p,g)=true} f(j,p,g) \times x(j,p) \\ & \text{subject to: } x(j,p) \in \{0,1\} \quad \forall j \in J, \forall p \in P \\ & x(j,p) = 1 \rightarrow a(j,p) = true \quad \forall j \in J, \forall p \in P \\ & \sum_{j \in J} \sum_{p \in P} c(j,p) \times x(j,p) \leq B \\ & \sum_{p \in P} x(j,p) \leq 1 \quad \forall j \in J \end{aligned}$$

A slight variation of the previous optimization problem for the job provider can be specified as follows. 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 to each platform p does not exceed its budget b_p , and the total worker fairness of the assigned jobs is maximized. This optimization problem can be formulated as follows.

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*

Algorithm 4 Job Provider Problem with Global Budget Algorithm

1: **Input:** A set of jobs J , a set of platforms P , a set of groups G , and an integer B .

2: **Output:** The maximum size subset of (j, p) pairs with the highest minimum fairness over all groups G having cost at most B and where each job is assigned to at most one platform. Running time is $\mathcal{O}(JPB)$.

▷ Step 1: Initialization, aggregation of fairness values

3: $minFair[1..len(J)][1..len(P)] \leftarrow$ new 2D array initialized to $+\infty$.

4: **for** $j \in J$ and $p \in P$ and $g \in G$ **do**

5: **if** $e(j, p, g) = true$ **then**

6: $minFair[j][p] = \min(minFair[j][p], f(j, p, g))$

7: **end if**

8: **end for**

▷ Step 2: Iterative DP: For each subproblem containing the first i jobs, $DP[i][t]$

▷ will store the optimal fairness obtainable from these jobs at budget limit t .

9: $DP[0..len(J)][0..B] \leftarrow$ new 3D array initialized to 0.

10: **for** $i \in [0, len(J))$ and $t \in [0, B]$ **do**

11: $dp[i + 1][t] \leftarrow \max(dp[i + 1][t], dp[i][t])$

12: **for** $j \in [1..len(P)]$ **do**

13: $(f, c) \leftarrow (minFair[J[i]][P[j]], c(J[i], P[j]))$

14: **if** $c + t \leq B$ **then**

15: $dp[i + 1][c + t] \leftarrow \max(dp[i + 1][c + t], f + dp[i][t])$

16: **end if**

17: **end for**

18: **end for**

▷ Step 3: Get total cost of the optimal assignment found

19: $maxFairness \leftarrow 0$

20: $b \leftarrow 0$

21: $N \leftarrow len(J)$

22: **for** $t \in [0..B]$ **do**

23: **if** $dp[N][t] > maxFairness$ **then**

24: $maxFairnes \leftarrow dp[N][t]$

25: $b \leftarrow t$

26: **end if**

27: **end for**

```

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

42: return result

```

maximized. Our second version of the job-provider problem can be formulated as the following optimization problem:

$$\begin{aligned}
 & \max \sum_{j \in J} \sum_{p \in P} \min_{g | e(j,p,g) = \text{true}} f(j, p, g) \times x(j, p) \\
 & \text{subject to: } x(j, p) \in \{0, 1\} \quad \forall j \in J, \forall p \in P \\
 & x(j, p) = 1 \rightarrow a(j, p) = \text{true} \quad \forall j \in J, \forall p \in P \\
 & \sum_{j \in J} c(j, p) \times x(j, p) \leq b_p \quad \forall p \in P \\
 & \sum_{p \in P} x(j, p) \leq 1 \quad \forall j \in J
 \end{aligned}$$

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

Theorem 2. *The JOB PROVIDER WITH GLOBAL BUDGET and the JOB PROVIDER WITH LOCAL BUDGET problems are 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, known to be NP-Hard. \square

Like for Problem 2, the similarity between Problem 3 and the KNAPSACK problem gives a near-immediate dynamic programming algorithm, described in Algorithm 4.

This is not all for Problem 4, however. Note that, if the aggregation of fairness values for each group is done a priori, then Problem 4 becomes equivalent to LEGAP, a variant of the Generalized Assignment Problem (GAP) where each job must be assigned to *at most* one platform instead of *exactly* one [19]. And since LEGAP is proven to be equivalent to the "standard" GAP [19, 20], 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. 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 $c(j, p_{dummy}) = 1 \forall j \in J$), 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 $\mathcal{P} = \mathcal{NP}$ [19]. 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 outline for solving GAP is the branch-and-bound (BB) method. We examine three algorithms from this category: 1) the BB with multiplier adjustment method (MAM) by Fisher et al. [21, 19], 2) the BB with steepest descent MAM by Karabakal et al. [22], and 3) the BB with variable fixing by Posta et al. [23]. 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 [23]). A scalability comparison of these algorithms is included in Chapter 4.

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 [19]; 2) a Local Search Descent method by Osman [24]; and 3) a Tabu Search method by Osman [24]. A comparison of these algorithms, both in terms of performance and solution quality, can be found in Chapter 4.

CHAPTER 4

EXPERIMENTS

To evaluate our proposed framework described in the previous chapter, 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 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 chapter 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 Semi-Synthetic Dataset (for Qualitative Experiments)

To simulate multiple, semi-synthetic platforms, we use the TaskRabbit dataset from [16], 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 *world1* to *world4* are created based on gender interventions from the original world as follows: *world1* has percentages of males and females switched compared to the original; *world2* is composed of 50% males and 50% females; *world3* is composed of 30% males and 70% females; and finally *world4* is composed of 70% males and 30% females.

The worlds *world5* to *world8* are created based on ethnicity interventions from the original world as follows: *world5* contains 33% black, 33% white, and 34% asian people. Worlds 6 through 8 are created from switching the percentages of two of the ethnicities from the original world. So, *world6* swaps the percentages of whites and blacks, *world7* those of whites and asians, and finally *world8* those of blacks and asians. A summary of the resulting platforms and their worker distributions can be found in Table 4.1.

World	Male	Female	World	Black	White	Asian
Taskrabbit	0.75	0.25	Taskrabbit	0.24	0.69	0.07
World1	0.26	0.74	World1	0.27	0.66	0.07
World2	0.50	0.50	World2	0.25	0.68	0.07
World3	0.30	0.70	World3	0.26	0.67	0.07
World4	0.70	0.30	World4	0.24	0.69	0.07
World5	0.74	0.26	World5	0.33	0.33	0.34
World6	0.72	0.28	World6	0.69	0.24	0.07
World7	0.74	0.26	World7	0.24	0.07	0.69
World8	0.75	0.25	World8	0.07	0.69	0.24

(a) Gender statistics

(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 4.1: Platform statistics for the alternative worlds (in percentages)

4.2 Job Seeker Experiments

4.2.1 Algorithms Implementation

For the Unconstrained Job Seeker problem, both the naive and the top-k algorithms were implemented in Python 3.8, as the function to compute fairness values defined in [16], 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 thesis, 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 [16].

4.2.2 Unconstrained Scalability Experiments

To compare the performance of our two unconstrained job seeker algorithms at various scales, we now build 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. To compute fairness values, we use the two metrics defined in [16], namely Earth Mover Distance (EMD) and Exposure. Therefore, this scalability experiment is run for both metrics.

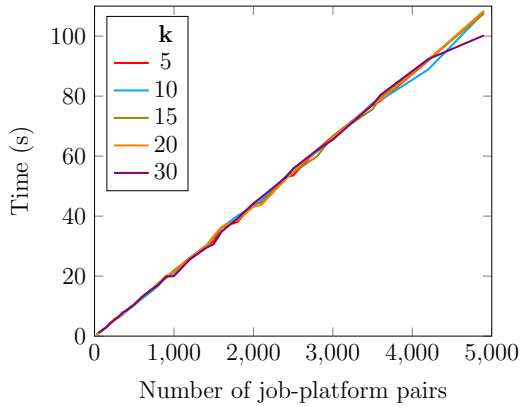
The experiment then goes as follows. For each run, we generate ten fictional

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 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.

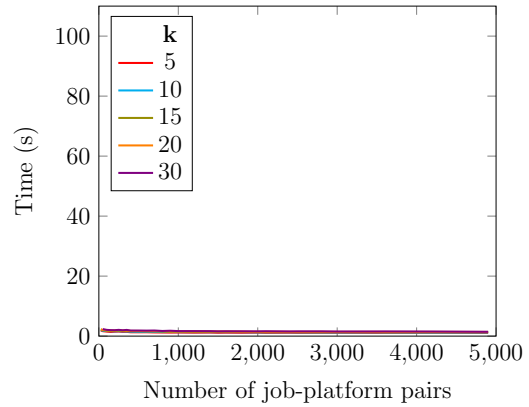
After performing all the runs, we first plot the time results for all values of k in Figure 4.1. As the curves for the different values of k show very similar trends, we only focus on $k = 20$ for comparing Naive vs. Top- k performance. A plot comparing runtimes for both algorithms at $k = 20$ is shown in Figure 4.2. 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.

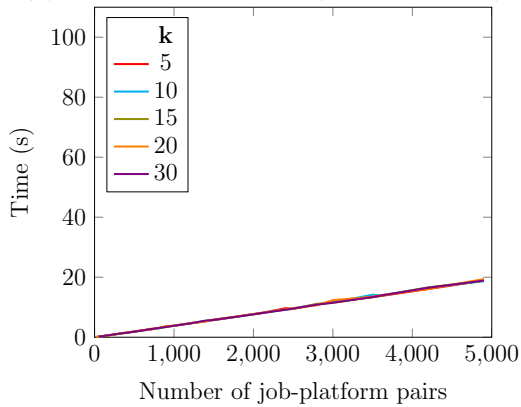
At this stage, it is important to distinguish between a *protected attribute* and a *group*. While a protected attribute is only one attribute or characteristic, such as gender or age, a group represents a combination of one or more protected attributes that are assigned a value, e.g. $\{gender : "female"\}$. 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.



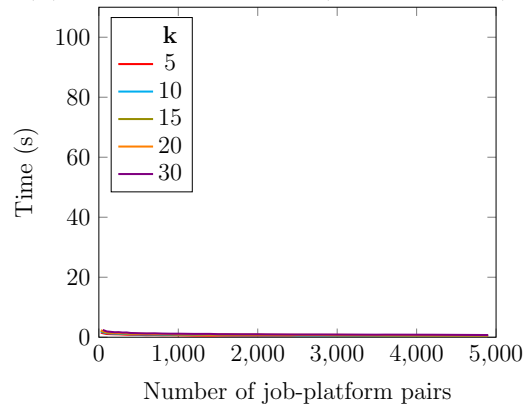
(a) Naive time wrt. N (metric: EMD)



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

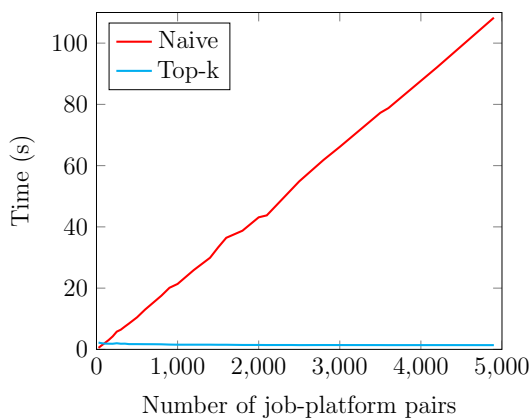


(c) Naive time wrt. N (metric: Exposure)

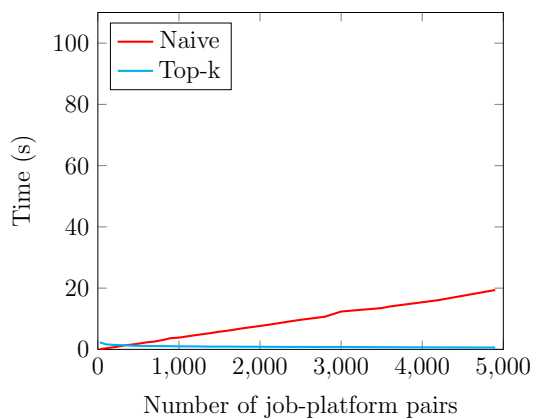


(d) Top-k time wrt. N (metric: Exposure)

Figure 4.1: Unconstrained Job Seeker Experiment: Naive vs. Top-k performance for different values of k



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



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

Figure 4.2: Unconstrained Job Seeker Experiment: Naive vs. Top-k Times for $k = 20$

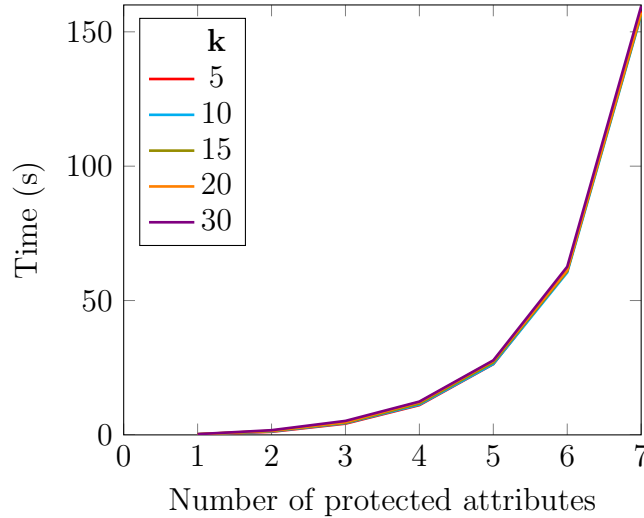


Figure 4.3: Top-k algorithm times wrt. Number of Protected Attributes, n .

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 consolidated by the obtained scalability results of Figure 4.3.

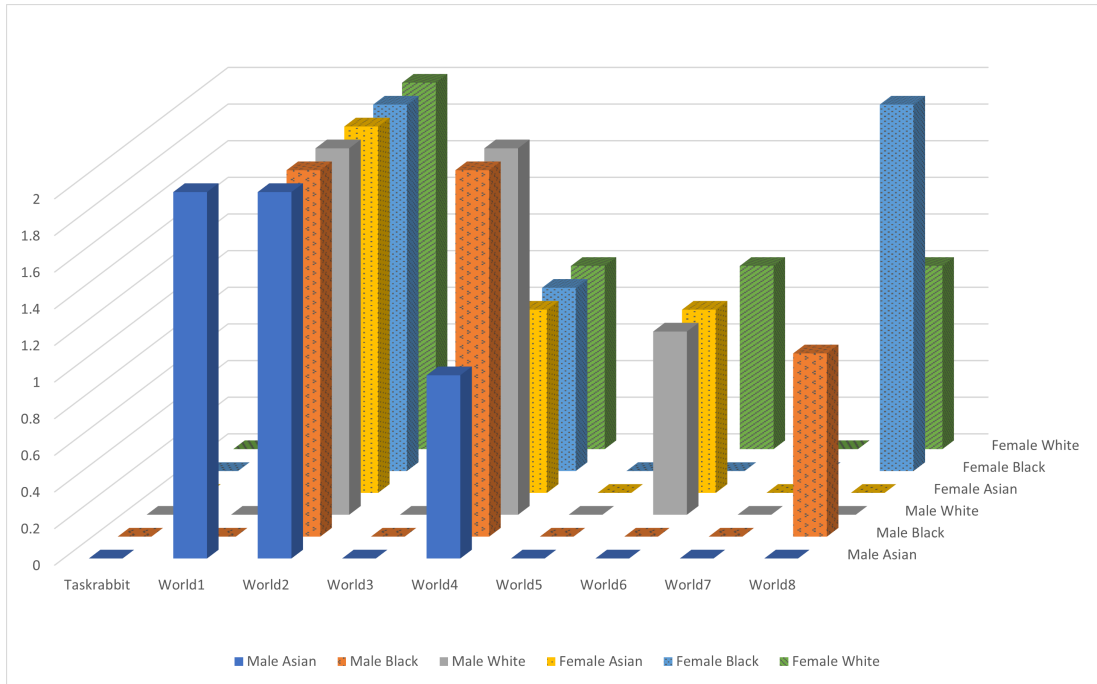


Figure 4.4: Occurrences of each World in the Seekers' Top-5 Results

4.2.3 Unconstrained Qualitative Experiments

The 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 4.4, and the number of occurrences of each job in Figure 4.5.

Looking at the world frequency results, we see that platforms *world2* and

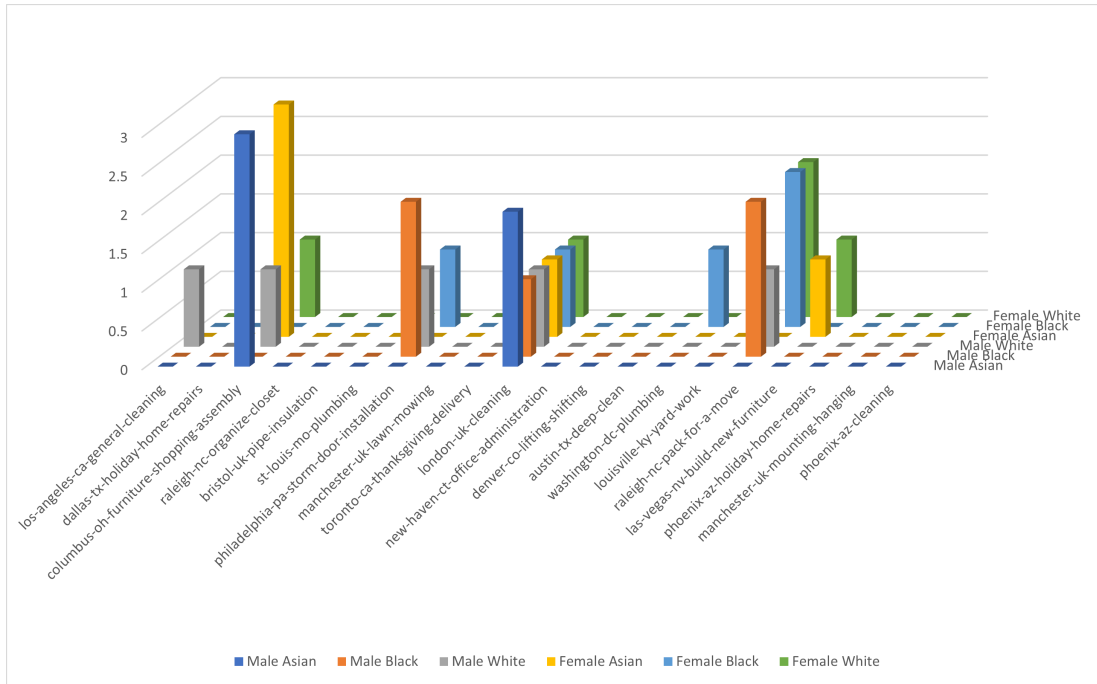


Figure 4.5: Occurrences of each Job in the Seekers' Top-5 Results

world4 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 *world7* 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 Columbus, 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 and each alternative

world p_i , we retrieve the seeker’s top five fairest jobs in platform p_i . Then, we also retrieve the seeker’s overall top-five fairest jobs in all platforms combined. Finally, for each top-five 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 4.6.

We notice that *world7* has the lowest sum of fairness values across the board, which indicates that *world7* is the least fair platform for the chosen set of jobs. Recall that *world7* 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.

World2 vs. World7: Further Analysis

To further understand the reason behind *world7*’s relatively poor fairness performance on the selected jobs, we compare statistics between this world and *world2*, one of the worlds that fared the best in our previous tests. We first compare the number of workers between *world2* and *world7* for the twenty jobs, which results are shown in Figure 4.7. The plot shows that the twenty jobs in *world7* have in general very few workers compared to *world2*, 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.

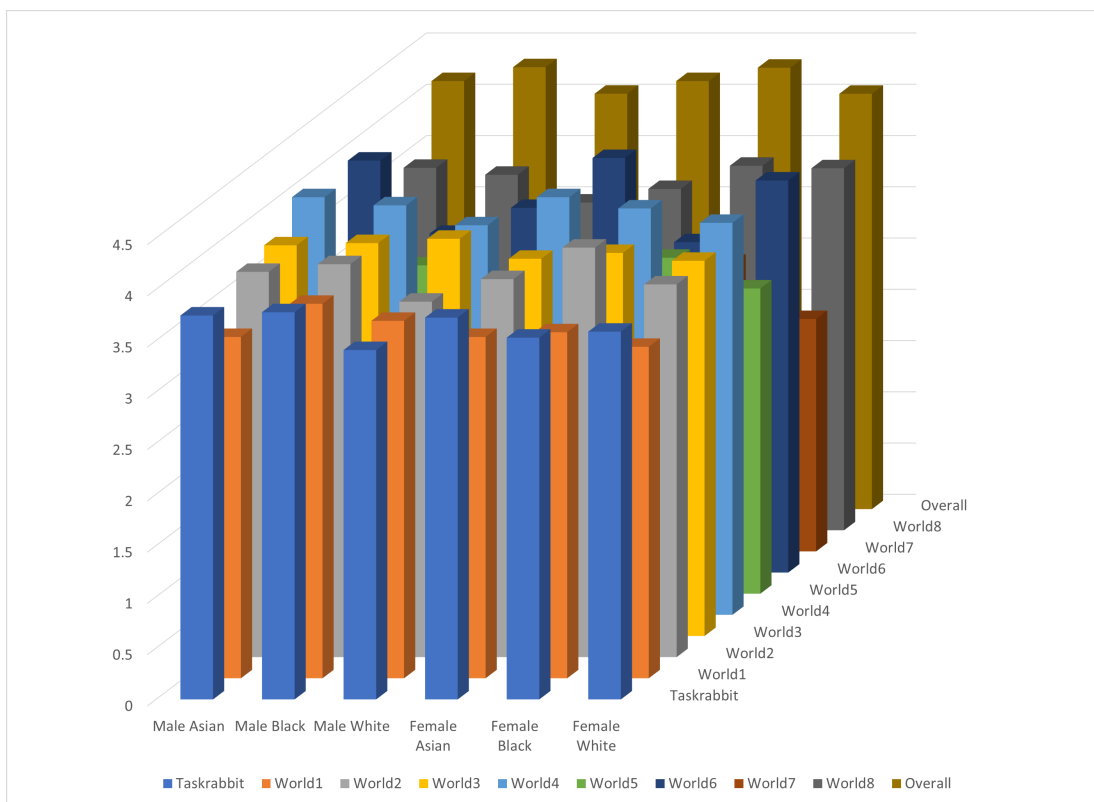


Figure 4.6: Sum of fairness values of the top-five (j, p) pairs, per seeker

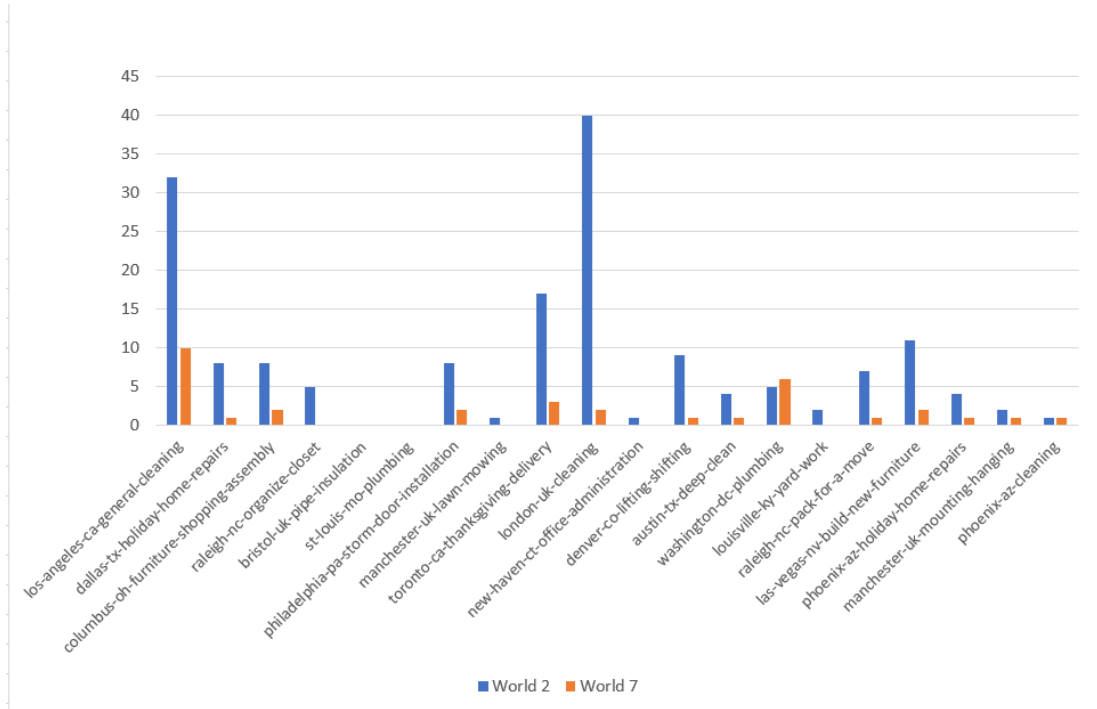


Figure 4.7: Comparing worker counts in the twenty selected jobs: *world2* vs. *world7*

4.2.4 Constrained Scalability Experiments

We now test the scalability of our Dynamic Programming algorithm proposed in Chapter 3. For this, we create 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 $2^2 - 1 = 3$).

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 Chapter 3, and 2) an off-the-shelf Integer

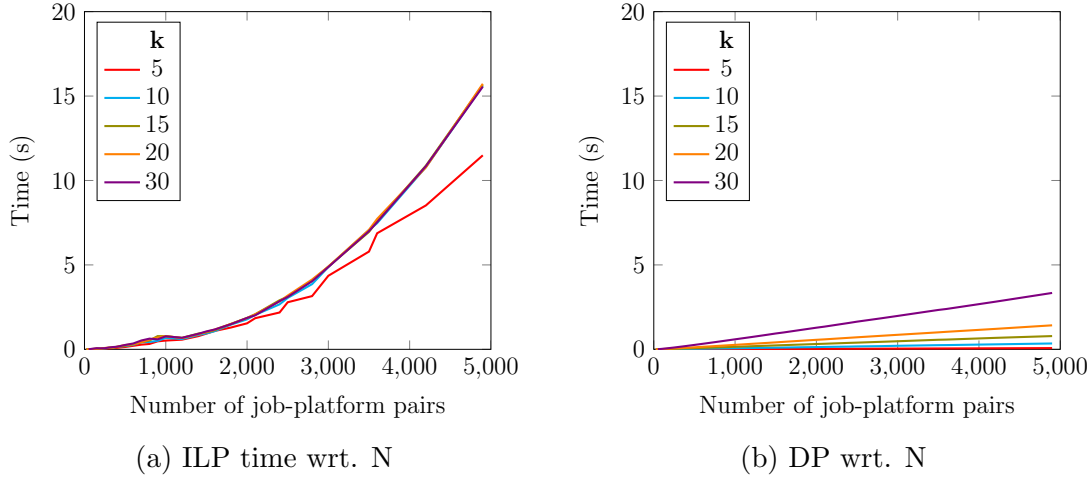


Figure 4.8: Constrained Job Seeker: Comparing performance of the ORTools solver (ILP) versus the proposed Dynamic Programming algorithm (DP)

Linear Programming (ILP) solver (Google's ORTools). We execute ten such runs for every (N, k) combination, and record the average solving time of each algorithm over the ten runs. The results are summarized and compared in the plots of Figure 4.8.

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 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 seem 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 time results are recorded in Figure 4.9.

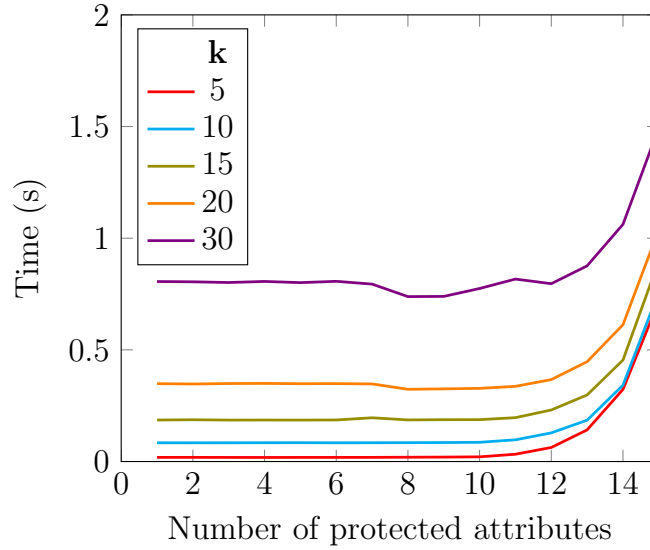


Figure 4.9: DP algorithm times wrt. Number of Protected Attributes, n .

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 Qualitative Experiments

Precision

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 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 large 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 $\{0, 1, 2, \dots, 10^d - 1\}$, 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" (smaller collisions) there is going from one precision d to the next. The observed values are shown in Figure 4.10 and Figure 4.12.

Looking at Figure 4.10 (with a clearer view in Figure 4.11), 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

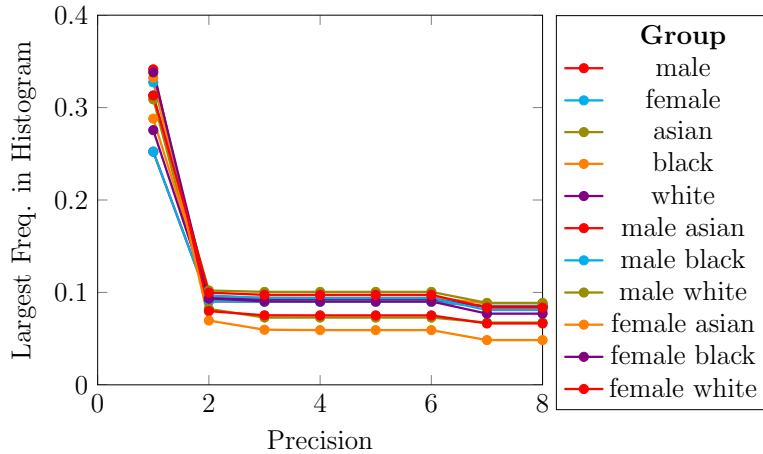


Figure 4.10: Variation in Largest Frequency Observed wrt. Precision used (lower is better)

we see lie between $d = 1$ and $d = 3$, with a relative gain starting from $d = 7$ onwards.

Also, Figure 4.12 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.

The Qualitative Experiments

We now conduct two experiments on the Constrained Job Seeker algorithm, using the exact same setting as the previous Unconstrained qualitative runs: same seekers, same jobs J and platforms P of interest. The experiments themselves are very similar to their Unconstrained counterparts, the only difference being that here, each job-platform pair is associated with a reward value between 1

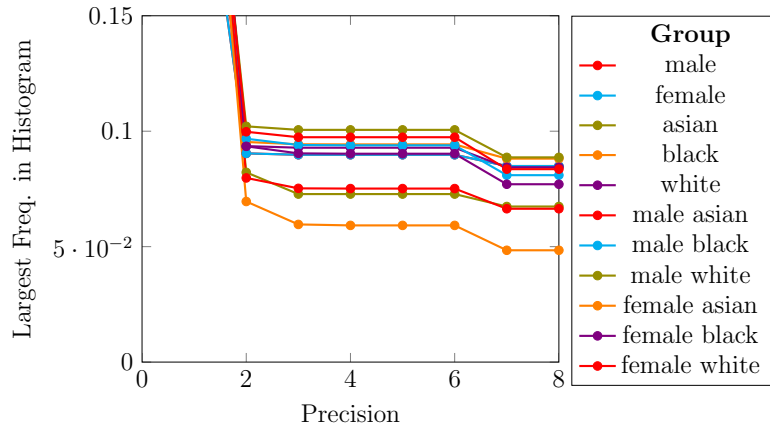


Figure 4.11: A closer look from the plot of Figure 4.10

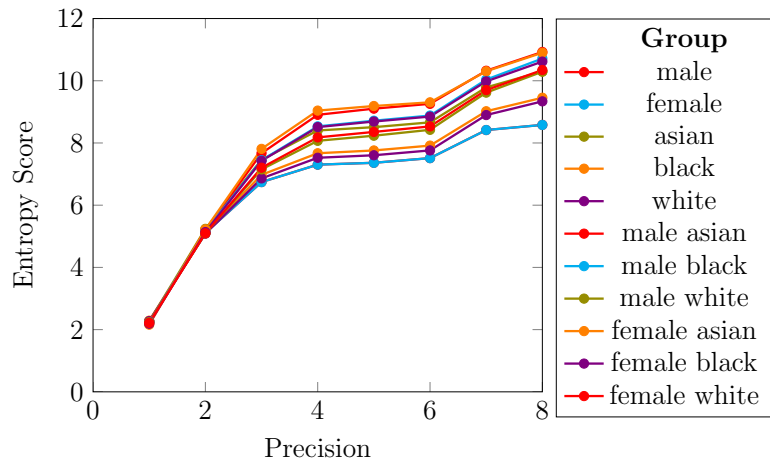


Figure 4.12: Variation in Entropy wrt. Number of Precision Digits (higher is better)

and 100, and now each seeker aims to select the top five 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 ensure the relevance of our provided algorithm.

For the first experiment, we use the same seekers as the first Unconstrained qualitative run, and find the top-five job-platform pairs for each seeker while satisfying the new reward threshold of 400. Then, we record the number of times each world and job occurs in every seeker's result set, as shown in the plots of Figures 4.13 and 4.14. Also recorded are the sum of (four-digit) fairness values and the sum of rewards for each result set, which can be seen in Table 4.2.

Looking at Figures 4.13 and 4.14, we notice that for both plots, the results shown are different from those of the corresponding Unconstrained 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 4.2 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 previously. 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 4.15, 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.

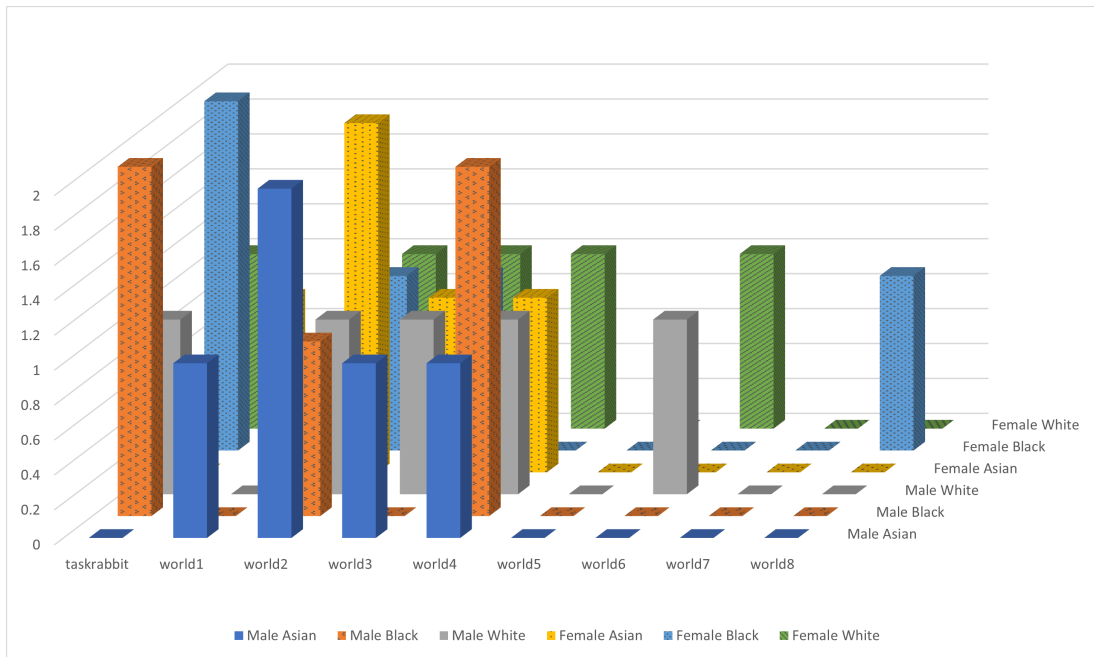


Figure 4.13: Constrained Experiment: Distribution of top-5 pairs among worlds, for each seeker

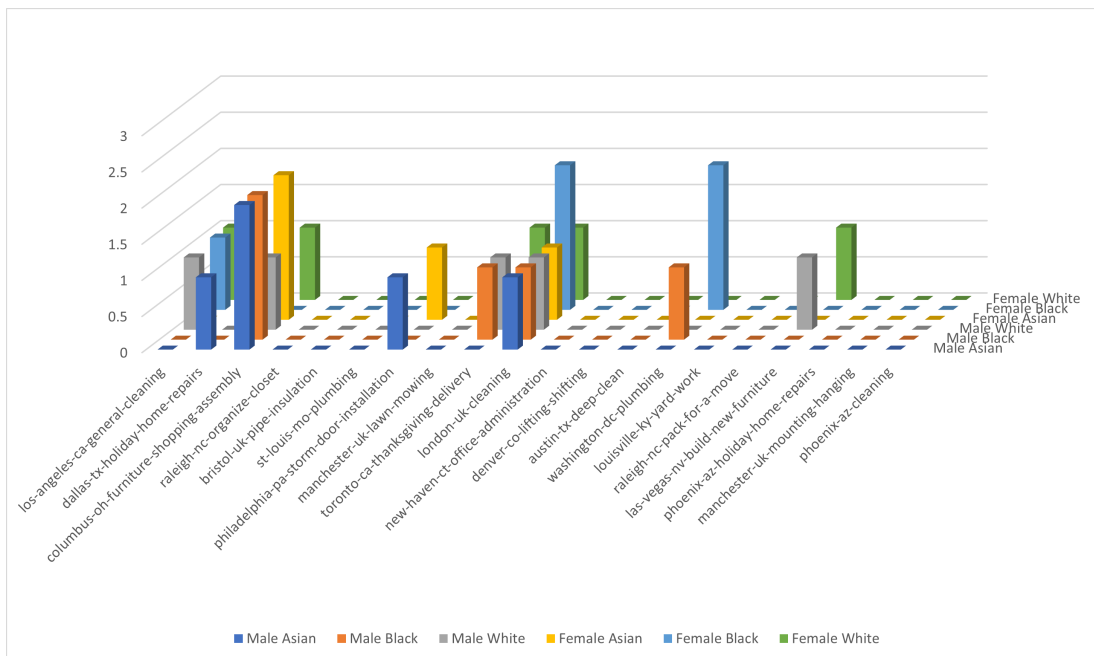


Figure 4.14: Constrained Experiment: Distribution of top-5 pairs among jobs of interest, for each seeker

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 4.2: Constrained Experiment: 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.

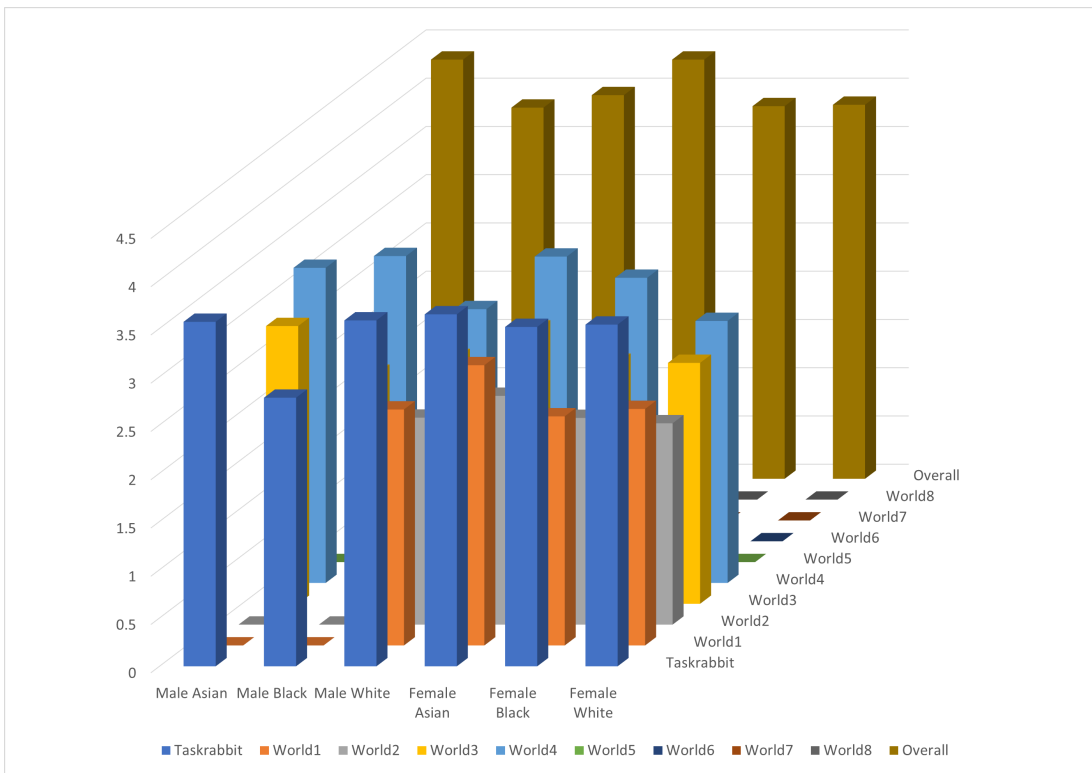


Figure 4.15: Constrained Experiment: Sums of fairness values of the top five (j, p) pairs, per seeker and world.

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, which C code was taken from the authors’ GitHub repository ¹, and Karabakal et al.’s algorithm, for which we used the C code from the technical report in [25].

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 one hundred such problem instances per (J, P) combination. This time we went for a hundred instances

¹<https://github.com/postamar/gap-solver>

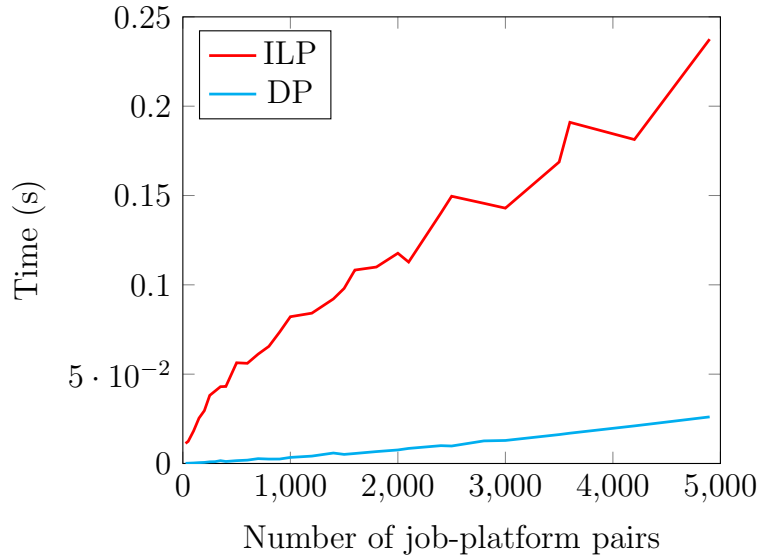


Figure 4.16: Job Provider with Global Budget: DP algorithm vs. ILP solver scalability wrt. number of pairs N .

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 solver. The average solving time of each method over the hundred instances is then recorded. The results are shown in Figure 4.16, 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 one hundred instances worth of fairness values and cost values in the same way as the experiment above. Then, each instance (i.e. set of fairness and cost values) is solved with increasing values of B , by both the DP algorithm

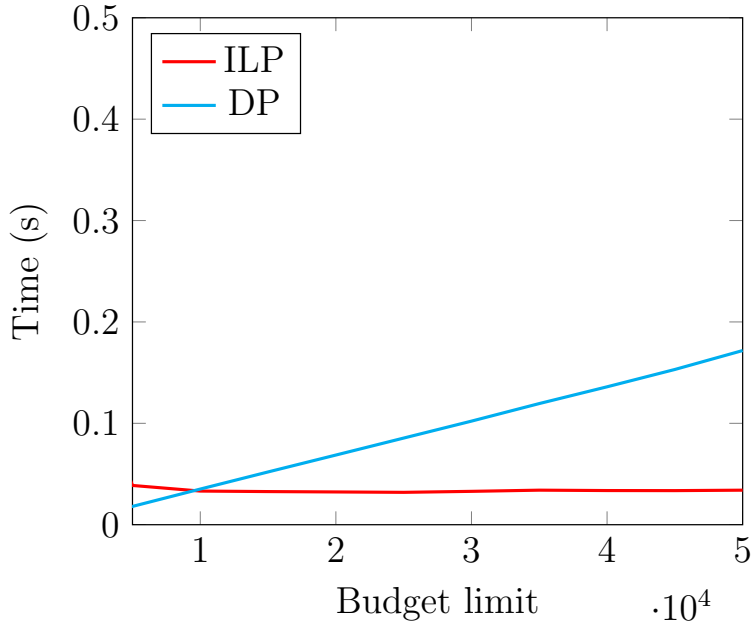


Figure 4.17: Job Provider with Global Budget: DP algorithm vs. ILP solver scalability wrt. budget limit B .

and the ORTools solver as a reference. Average time results are shown in Figure 4.17, 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 for 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 twenty jobs, selected at random. Each of the twenty 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 twenty 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 4.18. (Note that unlike for Job Seeker problems, the plot here is two-dimensional, since the "seeker" dimension is not relevant for Job Provider problems). The plot shows that the optimal fairness 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. Then, our conclusion here is twofold: 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 twenty 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 4.19. 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.

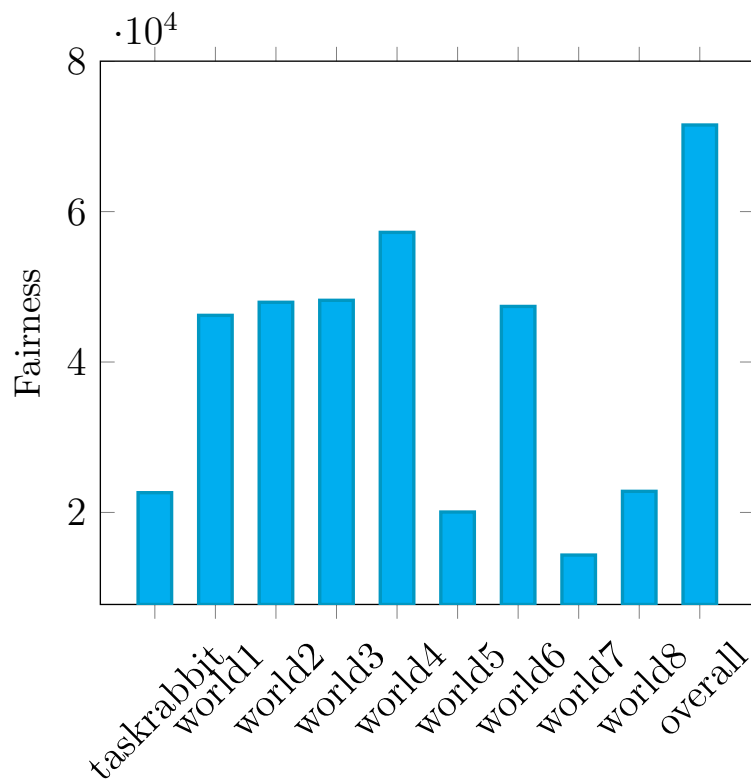


Figure 4.18: Job Provider with Global Budget: First qualitative experiment; Optimal fairness value obtained per platform(s) of interest.

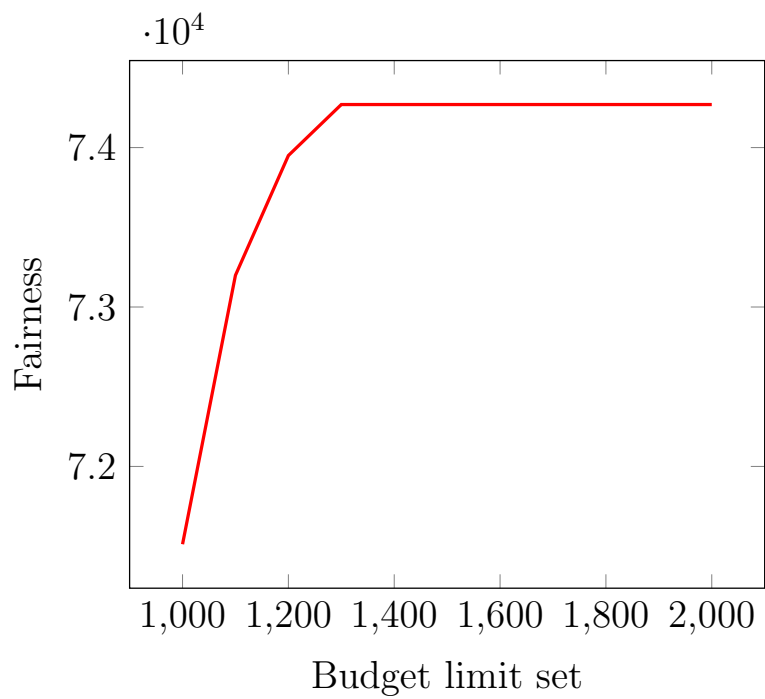


Figure 4.19: Job Provider with Global Budget: Second qualitative experiment; Optimal fairness value obtained for the same problem instance, but with varying budget limits.

4.3.4 Job Provider with Local Budgets Scalability

Experiments

Recall that for this problem, we are considering a selection of both exact and heuristic algorithms from the literature. To compare these algorithms' performance, 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 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:

$$b_p = \left\lceil \frac{100 \times |J|}{|P|} \right\rceil + \epsilon; \quad \epsilon \text{ random integer between 0 and 49.}$$

The point of this 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. [21], following the pseudo-code in [19];
- 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

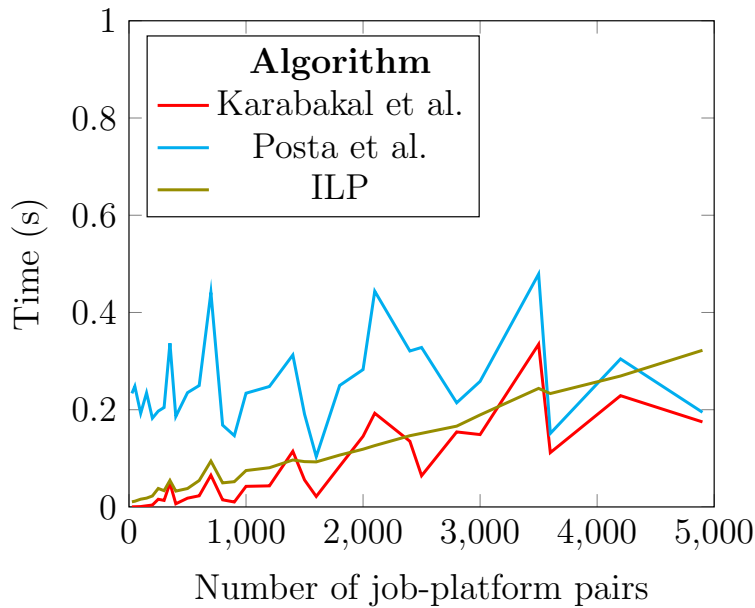


Figure 4.20: Job Provider with Local Budgets: Exact algorithms’ time performance wrt. N (number of pairs)

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 4.20. In the absolute, both Karabakal et al.’s and Posta et al.’s algorithms scale fairly well within our problems’ sizes, with Karabakal et al.’s method having a slight edge, however neither of them is much faster than the ILP solver.

Next, we move on to heuristics. The heuristics to be compared are:

- MTHG

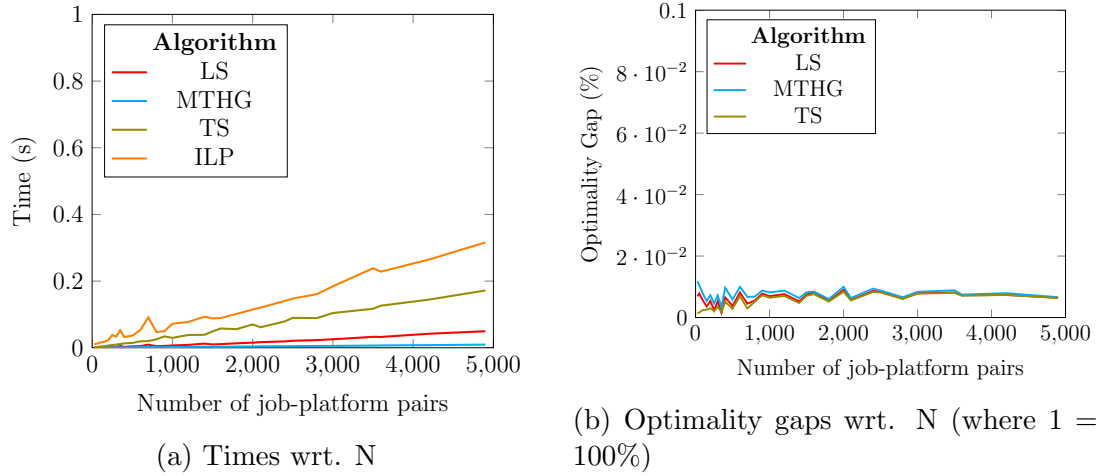


Figure 4.21: Job Provider with Local Budgets: Heuristic algorithms' time and quality performance wrt. N (number of pairs)

- 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 4.21, revealing that all three heuristics perform much faster than the ILP solver at scale, while returning decently accurate solutions within 2% from optimality on average. Therefore, if exact optimality

is not a must, then heuristics can be a solid, more efficient alternative to exact algorithms.

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 [22], and the total fairness value of the optimal assignment is recorded.

The results of this experiment are displayed in Figure 4.22. 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

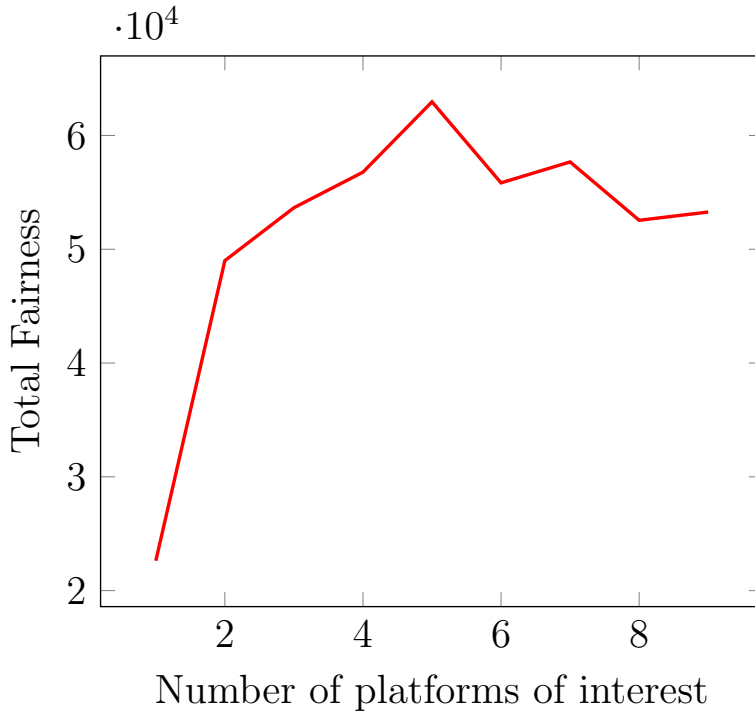


Figure 4.22: Job Provider with Local Budgets: First qualitative experiment; Optimal fairness value obtained with different number of platforms of interest.

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 results. 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 4.23. 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

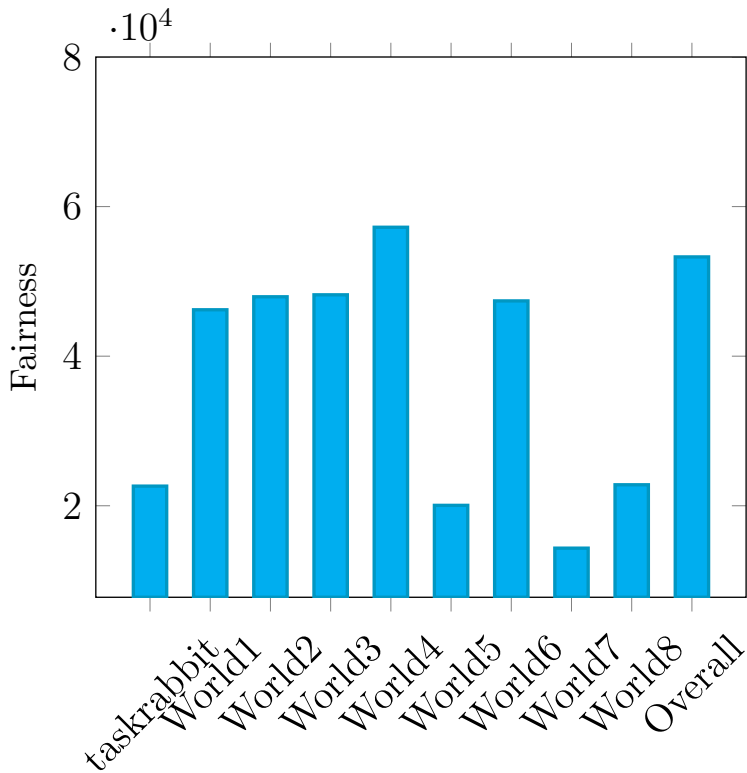


Figure 4.23: Job Provider with Local Budgets: First qualitative experiment; Optimal fairness value obtained per platform(s) of interest.

for the Global Budget variant (Section 4.3.3, Figure 4.18.), 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 problems are equivalent, and thus their algorithms return the same results. For the last run, the fairness 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.

CHAPTER 5

CONCLUSION

In this work, we proposed a framework to assess and compare fairness of ranking on multiple jobs, that can exist on multiple online platforms. We based our framework on realistic use cases for both job seekers and providers, which we formulated as four optimization problems. We also proved that three of these problems 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. Possible future work includes using our framework to conduct real-world case studies, where real jobs and platforms are examined from a ranking bias 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.

BIBLIOGRAPHY

- [1] T. Calders and S. Verwer, “Three naive bayes approaches for discrimination-free classification,” *Data Mining and Knowledge Discovery*, vol. 21, pp. 277–292, Sep 2010.
- [2] I. Zliobaite, “A survey on measuring indirect discrimination in machine learning,” *CoRR*, vol. abs/1511.00148, 2015.
- [3] A. Singh and T. Joachims, “Fairness of exposure in rankings,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2219–2228, 2018.
- [4] M. T. Keane, M. O’Brien, and B. Smyth, “Are people biased in their use of search engines?,” *Communications of the ACM*, vol. 51, no. 2, pp. 49–52, 2008.
- [5] A. J. Biega, K. P. Gummadi, and G. Weikum, “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.
- [6] M. Zehlike, F. Bonchi, C. Castillo, S. Hajian, M. Megahed, and R. Baeza-Yates, “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.

- [7] L. E. Celis, D. Straszak, and N. K. Vishnoi, “Ranking with fairness constraints,” *arXiv preprint arXiv:1704.06840*, 2017.
- [8] M. Zehlike, T. Sühr, R. Baeza-Yates, F. Bonchi, C. Castillo, and S. Hajian, “Fair top-k ranking with multiple protected groups,” *Information Processing & Management*, vol. 59, no. 1, p. 102707, 2022.
- [9] M. Zehlike and C. Castillo, “Reducing disparate exposure in ranking: A learning to rank approach,” in *Proceedings of The Web Conference 2020*, pp. 2849–2855, 2020.
- [10] A. Hannák, C. Wagner, D. Garcia, A. Mislove, M. Strohmaier, and C. Wilson, “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.
- [11] L. Chen, *Measuring algorithms in online marketplaces*. PhD thesis, Northeastern University, 2017.
- [12] S. Elbassuoni, S. Amer-Yahia, A. Ghizzawi, and C. Atie, “Exploring fairness of ranking in online job marketplaces,” in *22nd International Conference on Extending Database Technology (EDBT)*, 2019.
- [13] A. Ghizzawi, J. Marinescu, S. Elbassuoni, S. Amer-Yahia, and G. Bisson, “Fairrank: An interactive system to explore fairness of ranking in online job marketplaces,” in *22nd International Conference on Extending Database Technology (EDBT)*, 2019.

- [14] S. Elbassuoni, S. Amer-Yahia, and A. Ghizzawi, “Fairness of scoring in online job marketplaces,” *ACM Transactions on Data Science*, vol. 1, no. 4, pp. 1–30, 2020.
- [15] S. C. Geyik, S. Ambler, and K. Kenthapadi, “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.
- [16] S. Amer-Yahia, S. Elbassuoni, A. Ghizzawi, R. Borromeo, E. Hoareau, and P. Mulhem, “Fairness in online jobs:{A} case study on taskrabbit and google,” in *International Conference on Extending Database Technologies (EDBT)*, 2020.
- [17] R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” *Journal of computer and system sciences*, vol. 66, no. 4, pp. 614–656, 2003.
- [18] M. G. Lagoudakis, “The 0-1 knapsack problem—an introductory survey,” 1996.
- [19] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. USA: John Wiley & Sons, Inc., 1990.
- [20] M. Yagiura and T. Ibaraki, “Generalized assignment problem,” in *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)* (T. F. Gonzalez, ed.), Chapman & Hall/CRC, 2007.

- [21] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove, “A multiplier adjustment method for the generalized assignment problem,” *Management science*, vol. 32, no. 9, pp. 1095–1103, 1986.
- [22] N. Karabakal, J. C. Bean, and J. R. Lohmann, “A steepest decent [sic] multiplier adjustment method for the generalized assignment problem,” tech. rep., 1993.
- [23] M. Posta, J. A. Ferland, and P. Michelon, “An exact method with variable fixing for solving the generalized assignment problem,” *Computational Optimization and Applications*, vol. 52, no. 3, pp. 629–644, 2012.
- [24] I. H. Osman, “Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches,” *Operations-Research-Spektrum*, vol. 17, no. 4, pp. 211–225, 1995.
- [25] N. Karabakal, “A c code for solving the generalized assignment problem.,” tech. rep., 1992.