5-31-2022

Optimization opportunities in human in the loop computational paradigm

Dong Wei
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

OPTIMIZATION OPPORTUNITIES IN HUMAN IN THE LOOP COMPUTATIONAL PARADIGM

by
Dong Wei

An emerging trend is to leverage human capabilities in the computational loop at different capacities, ranging from tapping knowledge from a richly heterogeneous pool of knowledge resident in the general population to soliciting expert opinions. These practices are, in general, termed human-in-the-loop (HITL) computations.

A HITL process requires holistic treatment and optimization from multiple standpoints considering all stakeholders: a. applications, b. platforms, c. humans. In application-centric optimization, the factors of interest usually are latency (how long it takes for a set of tasks to finish), cost (the monetary or computational expenses incurred in the process), and quality of the completed tasks. Platform-centric optimization studies throughput, or revenue maximization, while human-centric optimization deals with the characteristics of the human workers, referred to as human factors, such as their skill improvement and learning, to name a few. Finally, fairness and ethical consideration are also of utmost importance in these processes.

This dissertation aims to design solutions for each of the aforementioned stakeholders. The first contribution of this dissertation is the study of recommending deployment strategies for applications consistent with task requesters' deployment parameters. From the worker's standpoint, this dissertation focuses on investigating online group formation where members seek to increase their learning potential via collaboration. Finally, it studies how to consolidate preferences from different workers/applications in a *fair manner*, such that the final order is both consistent with individual preferences and complies with a group fairness criteria.

The technical contributions of this dissertation are to rigorously study these problems from theoretical standpoints, present principled algorithms with theoretical guarantees, and conduct extensive experimental analysis using large-scale real-world datasets to demonstrate their effectiveness and scalability.

OPTIMIZATION OPPORTUNITIES IN HUMAN IN THE LOOP COMPUTATIONAL PARADIGM

by
Dong Wei

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2022

APPROVAL PAGE

OPTIMIZATION OPPORTUNITIES IN HUMAN IN THE LOOP COMPUTATIONAL PARADIGM

Dong Wei

Dr. Senjuti Basu Roy, Dissertation Advisor — Date
Associate Professor, Computer Science, NJIT

Dr. Chase Wu, Committee Member — Date
Professor, Computer Science, NJIT

Dr. Yiannis Koutis, Committee Member — Date
Associate Professor, Computer Science, NJIT

Dr. Pan Xu, Committee Member — Date
Assistant Professor, Computer Science, NJIT

Dr. Sihem Amer-Yahia, Committee Member — Date
Research Director, Centre National de Recherche Scientifique, FRANCE

BIOGRAPHICAL SKETCH

Author: Dong Wei

Degree: Doctor of Philosophy

Date: May 2022

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 2022
- Master of Science in Information Science
  University of Pittsburgh, Pittsburgh, PA, 2016
- Bachelor of Engineering in Transportation
  Central South University, Changsha, China, 2014

Major: Computer Science

Presentations and Publications:

Dong Wei, Md Mouinul Islam, Baruch Schieber, and Senjuti Basu Roy, "Rank Aggregation with Proportionate Fairness" *Proceedings of the 2022 International Conference on Management of Data (SIGMOD' 22)*, ACM.

Dong Wei, Ioannis Koutis, and Senjuti Basu Roy, "Peer Learning Through Targeted Dynamic Groups Formation" *2021 IEEE 37th International Conference on Data Engineering (ICDE' 21)*, IEEE.

Dong Wei, Senjuti Basu Roy, and Sihem Amer-Yahia, "Recommending Deployment Strategies for Collaborative Tasks" *Proceedings of the 2020 International Conference on Management of Data (SIGMOD' 20)*, ACM.

Dong Wei, Senjuti Basu Roy, and Sihem Amer-Yahia, "Task Deployment Recommendation with Worker Availability" *2020 IEEE 36th International Conference on Data Engineering (ICDE' 20)*, IEEE.

To my beloved family

ACKNOWLEDGMENT

I am extremely grateful for the chance I have had at NJIT over the past five years, during which I have gotten great instruction, formed lifelong friendships, and pursued my goals freely. First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Senjuti Basu Roy for her invaluable inspiration, encouragement, and supervision along the way during the course of my PhD degree.

I would express my sincere thanks to my dissertation committee members: Dr. Sihem Amer-Yahia, Dr. Chase Wu, Dr. Yiannis Koutis, and Dr. Pan Xu. I am truly honored to have them in my dissertation committee.

I would also like to express my sincere gratitudes to Dr. Sihem Amer-Yahia, Dr. Yiannis Koutis, Dr. Schieber Baruch for their guidance. I would like to thank Dr. Sihem Amer-Yahia who provided me with many examples of finding and formalizing a good human-in-the-loop problem and how to construct a research paper based on these problems. I am grateful to Dr. Yiannis Koutis uses his brilliant graph theory knowledge and math intuition that helped me design solutions for the dynamic group formation problem. Finally, I would like to thank Dr. Schieber Baruch for his valuable guidance on my research. I would also like to thank all my collaborators who are part of this thesis: Dr. Sihem Amer-Yahia, Dr. Yiannis Koutis, Dr. Schieber Baruch, Dr. Mohammadreza Esfandiari (Payam), Mr. Md Mouinul Islam.

I would like to extend my sincere thanks to the Department of Computer Science and to the National Science Foundation #1942913. It would be impossible for me to complete my Ph.D. degree without their generous support.

I also wish to thank the Big Data Analytics Lab (BDaL) and my lab members, Dr. Mohammadreza Esfandiari (Payam), Mrs. Sepideh Nikookar, Mr. Md Mouinul Islam, Ms. Mahsa Asadi and Mr. Md Rakibul Hasan, for their support, help, and research collaboration.

From the bottom of my heart I would like to say big thank you for all the friends and peers I have met at NJIT for the thoughts we exchanged, the discussions we had, and the days and nights we fought together. An incomplete list includes Dr. Mohammadreza Esfandiari (Payam), Mr. Songlin He, Dr. Wuji Liu, Dr. Huiyan Cao, Mr. Yucong Shen, Mr. Yufei Zhang, Mr. Shibo Yao, and more.

Last but not least, I would like to express my deepest love to my parents, 魏碧海, 王文玲. They have always been encouraging and supporting me to achieve my dream.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Overview

"Human-in-the-loop" is a computational paradigm that is introduced for a plethora of compelling applications that benefit both from human cognition and machine intelligence [10, 11, 143]. An emerging trend there is to leverage human capabilities in the computational loop at different capacities, ranging from tapping knowledge from a richly heterogeneous pool of knowledge resident in the general population, to soliciting opinion from experts. These practices are in general termed as human-in-the-loop (HITL) computations [57, 68, 93, 97–99]. The human in the loop problems range from simpler ones, such as, sorting [97], filtering, sentiment analysis [102], top-k ranking [120], entity recognition, to more complex ones, like, task planning [132], feature engineering [16], or human machine decision making. A HITL process requires holistic treatment and optimization from multiple standpoints considering all stakeholders: a. applications, b. platforms, c. humans. In application-centric optimization, the factors of interest usually are latency (how long does it take for a set of tasks to finish), cost (the monetary or computational expenses incurred the process), and quality of the completed tasks. Platform-centric optimization studies throughput, or revenue maximization. On the contrary, human centric optimization deals with the characteristics of the human workers, referred to as human factors, such as, their skill improvement and learning, to name a few. Finally Fairness and ethical consideration are also of utmost importance in these processes.

1.1.1 Background and Motivations

Existing HITL systems tend to push humans further in the loop and largely ignore their role and contribution to the ecosystem. Human workers are mere agents and are

used to pursue broader computational goals. Bringing humans back to the frontier of HITL will improve their quality of life, ensure better work performance, and in turn, improve the computational goals. In fact, the existing literature on HITL computation has a clear bifurcation. The computational world of research has mostly focused on treating humans as agents [97, 107]. Contrarily, psychologists and social scientists have acknowledged and reasoned about the necessity to incorporate human characteristics in the computational loop but treat these problems mostly empirically, and their contributions are limited to field study and laboratory experiments [15]. Clearly, there is a need about designing quantitative models and algorithms that recognize and capture human characteristics and study optimization in the HITL process. Combining these two aspects is the broader goal of this dissertation.

1.1.2 Contributions

This dissertation makes several non-trivial contributions in human-in-the-loop computational paradigm.

It begins by discussing how to aid workers in improving their skill through on-job collaboration. In Chapter 2, it studies how to improve learning of workers while they are collaborating with machines and with each other on online platforms. It enables powerful and effective ways to improve individuals' knowledge and promote learning, such as MovieLens, Quora, and Coursera. Online group formation has been studied where members seek to increase their learning potential via collaboration. We capture two common learning models: LpA where each member learns from all higher skilled ones, and LpD where the least skilled member learns from the most skilled one. We formulate the problem of forming groups with the purpose of optimizing peer learning under different affinity structures: $A_{FF}D$ where group affinity is the smallest between all members, and $A_{FF}C$ where group affinity is the smallest between a designated member (e.g., the least skilled or the most skilled) and all others. This

gives rise to multiple variants of a multi-objective optimization problem. We propose principled modeling of these problems and investigate theoretical and algorithmic challenges. We first present hardness results, and demonstrate the $Lp*, A_{FF}*$ problems are NP-Complete. Then, we develop computationally efficient algorithms with constant approximation factors. Our real-data experiments demonstrate with statistical significance that forming groups considering affinity improves learning. Our extensive synthetic experiments demonstrate the qualitative and scalability aspects of our solutions.

In Chapter 4, as the follow-up work of the previous peer learning project [61], we study the dynamic version of peer learning problems. Imagine a peer learning scenario in a physical classroom or on an online learning platform. Separating the participants into equal-sized groups for homework assignments or projects is common practice to ensure a relatively similar workload among students, while enabling effective interaction among the peers. However, in the case when there is a series of homework assignments or group projects, fixed groups may not be optimal. In this paper we initiate a *dynamic* variant of the problem that, unlike previous works, allows the change of group composition over time while still targeting to maximize the aggregated knowledge level. The problem is studied in a principled way, using a realistic learning gain function and for two different interaction modes among the group members (similar to LpA and LpD in [61]).

On the algorithmic side, we present DyGroups, a generic algorithmic framework that is greedy in nature and highly scalable. We then introduce an optimal solution for a special case of DyGroups with non-trivial derivation. We demonstrate the different aspects of problem and how other methods lead to sub-optimal. Our detailed real and synthetic experiment results manifest our proposed model indeed improves the actual learning gains of individual and worker retention.

In Chapter 5, we focus on the trend of aiding requesters in deploying tasks on crowdsourcing platforms. We initiate the study of recommending deployment strategies to task requesters that are consistent with deployment parameters they desire: a lower-bound on the quality of the crowd contribution, an upper-bound on the latency of task completion, and an upper-bound on the cost incurred by paying workers. A deployment strategy is a choice of value for three dimensions: *Structure* (whether to solicit the workforce sequentially or simultaneously), *Organization* (to organize it collaboratively or independently), and *Style* (to rely solely on the crowd or to combine it with machine algorithms). We propose StratRec, an optimization-driven middle layer that recommends deployment strategies and alternative deployment parameters to requesters by addressing multi-faceted modeling and computational challenges by accounting for worker availability. We develop computationally efficient algorithms to recommend deployments that maximize task throughput and pay-off, as well as develop ADPaR to recommend alternative deployment. Our solutions are grounded in discrete optimization, and computational geometric techniques that produce results with theoretical guarantees. We present extensive experiments through multiple deployments with real workers in Amazon Mechanical Turk, and conduct synthetic experiments to validate the qualitative and scalability aspects of StratRec.

In Chapter 6, this dissertation contributes to rank aggregation with proportionate fairness. In this work we revisit some concepts in resource allocation to model a notion of fairness, namely proportionate fairness or p-fairness to ensure proportionate representation of every group based on a protected attribute in every position of the aggregated ranked order [26, 134]. Given multiple individual rank orders over a set of candidates or items, where the candidates belong to multiple (non-binary) protected groups, rank aggregation under p-fairness (**RAPF** in short) is designed to produce an aggregated ranked order that minimizes disagreements among

the individual rankings and ensures that the accumulated number of representation of each group in the aggregated ranking is proportionate to their representation in the original data for every position in the aggregated rank. We revisit the p-fair rank aggregation considering a popular rank aggregation objective, namely the Kemeny optimal aggregation. We observe that **RAPF** is NP-hard. We present a randomized computational framework RandAlgRAPF that is highly scalable and a deterministic computational framework AlgRAPF that produce a solution for **RAPF**. Both algorithms rely on producing a p-fair Kemeny optimized ranking for an individual ranking - we refer to this problem as Individual p-Fairness or **IPF** in short. We make several non-trivial algorithmic contributions: (i) we prove that when the group protected attribute is binary, **IPF** could be solved exactly using a greedy technique; (ii) when the group protected attribute is non-binary, solving **IPF** in non-trivial; (iii) we present two algorithmic frameworks RandAlgRAPF and AlgRAPF and show that they produce an $\alpha+2$ approximation factor, if **IPF** can be solved with approximation factor α; (iv) We present two different solutions for **IPF**, ExactMultiValuedIPF is optimal (i.e., $\alpha = 1$) and ApproxMultiValuedIPF admits 2 approximation factor respectively, leading to 3 and 4 approximation factors for the **RAPF** problem. We run extensive experiments using multiple real world and large scale synthetic datasets and compare our proposed solutions against multiple state-of-the-art related works to demonstrate the effectiveness and efficiency of our studied problem and proposed solution.

In Chapter 7, we discuss the margin finding problem under single ballot substitutions with considering various protected group attribute settings to promote fairness. The goal in this work is to optimize preference substitution (minimize the number of single ballot substitutions) to satisfy complex top-k fairness constraints. We formalize several margin finding problems via single ballot (preference) substitutions considering complex fairness constraints: (i) In MFBinaryS, proportionate

representation is required over a single binary protected attribute, such as male and female of the protected attribute gender; (ii) In MFMultiS, it is defined over a single multi-valued protected attribute, such as, race that contains more than 2 different values; (iii) Contrarily, in MFMulti2, proportionate representation is required over two different protected attributes, such as gender and race; and finally, (iv) in MFMulti3+, we study the *margin finding problem via preference substitutions* considering three or more protected attributes, such as, race, gender, and ethnicity. Then, we study the defined problems theoretically and make principled algorithmic contributions. We prove that both MFBinaryS and MFMultiS are computationally easy. Next, we consider MFMulti2 and MFMulti3+ in which two or more attributes are involved in defining fairness requirement. To the best of our knowledge we are the first to study these problems rigorously from computational standpoint. we prove that the decision version of MFMulti2is (weakly) NP-hard by reducing the well known NP-hard Partition problem to our problem. On the other hand, for MFMulti3+, we prove that the satisfiability problem itself is (strongly) NP-hard through a reduction from the three-dimensional matching (3DM). We conduct rigorous large scale experiments involving three real world (involving election and movie applications) and one synthetic datasets and compare multiple state-of-the-art solutions [66, 129] after appropriate adaptation.

We note that there are many interesting ongoing and future research directions that one could explore. As an example, what if we can not precisely inquiry the worker availability for deployment strategy recommendation? How to guarantee the fairness of task assignment for workers in HILT? These are discussed in length in Chapter 8.

CHAPTER 2

OPTIMIZING PEER LEARNING IN ONLINE GROUPS WITH AFFINITIES

2.1 Introduction

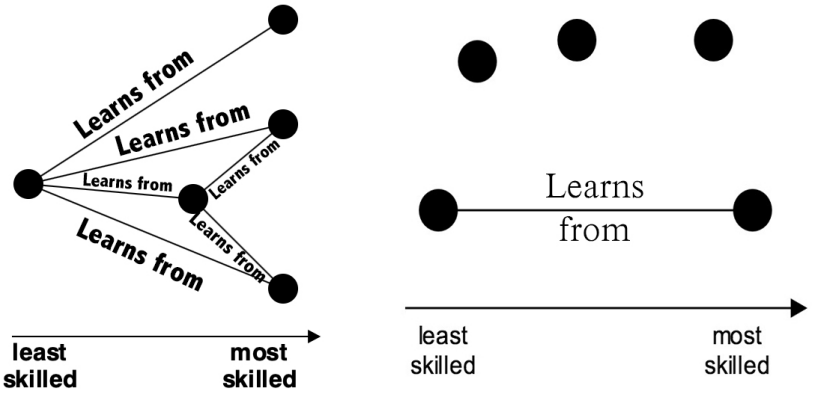
The emergence of platforms that support online networked technologies has changed the way we communicate, collaborate, and learn things together. Existing works have focused on how to identify and rank groups and communities [40], how to efficiently form a set of groups to optimize different group recommendation semantics [123], or form groups for task assignment [14, 15, 28, 88, 116]. The effect of online collaboration however goes beyond, as it enables powerful and versatile strategies to improve knowledge of individuals and promote learning. For example, online critiquing communities,[1] social Q&A sites,[2] and crowdsourcing platforms[3] investigate how collaboration can promote knowledge and skill improvement of individuals. *Learning potential*, is a key reason behind effective collaboration. It has been shown that the increase in learning one expects from collaboration yields fruitful coordination and higher quality contributions [3, 4]. For instance, in online fan-fiction communities, informal mentoring improves people's writing skills [62]. In this paper, we propose to explore how affinity between group members improves peer learning and address modeling, theoretical, and algorithmic challenges. To the best of our knowledge, our work is the first to examine algorithmic group formation with affinities for peer learning.

Group formation in online communities has been studied primarily in the context of task assignment [14, 15, 28, 88, 116]. The problem is often stated as: given a set of individuals and tasks, form a set of groups for the tasks that optimize some

[1] `https://movielens.org/` Retrieved on May/01/2019
[2] `http://quora.com/` Retrieved on May/01/2019
[3] `https://www.figure-eight.com/` Retrieved on May/01/2019

aggregated utility subject to constraints such as group size, maximum workload etc. Utility can be aggregated in different ways: the sum of individual skills, their product, etc [15]. Group formation is combinatorial in nature and proposed algorithms solve the problem under different constraints and utility definitions (e.g., [88]). Unlike these problems, we study how to form groups with the goal of maximizing peer learning under different affinities.

Our first contribution is to present principled models to formalize peer learning and affinity structures. We assume that a peer can only learn from another peer if the skill of the latter is strictly higher than the skill of the former [4]. The learning potential of a peer from a more skilled peer can then naturally be defined as the skill difference between the latter and the former [3,4]. The learning potential of the latter from the former is null. We use that to formulate two common learning models (see picture below): LpA where each member learns from all higher skilled ones, and LpD where the least skilled member (resp., the most skilled) learns from (resp. teaches to) all others.

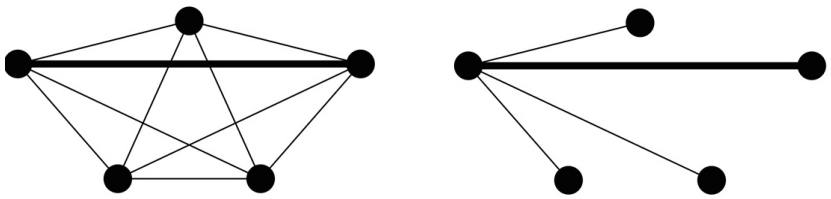


LpA: members learn from higher-skilled ones

LpD: the least skilled member learns from the most skilled one

Figure 2.1 Visual depiction of learning potential

Affinity, on the other hand, depends on the application and can be expressed using common socio-demographic attributes or more generally, using models that capture psychological traits. We study our two learning models in conjunction with two common affinity structures (see picture below): AffD where group affinity is the smallest between all members, and AffC where group affinity is the smallest between a designated member (e.g., the least skilled or the most skilled) and all others. We investigate these two affinity scenarios through fact-checking and fact-learning applications.



AffD: smallest affinity between all pairs

AffC: smallest affinity between one member and others

Figure 2.2 Visual depiction of affinities.

Our second contribution is to study the formalized models systematically and present our theoretical findings. In its general form, our problem formulation is a bi-objective optimization, with the goal to build k equi-sized groups over a set of n members that maximize both learning potential and affinity. Interestingly, we prove that no variant of optimizing learning potential alone is hard to solve (LpD and LpA), however, the problems become NP-hard when affinity and group size constraints are considered. Therefore, our solution first finds k groups that yield the highest possible learning potential value and then transforms our two-objective problem into *a constrained optimization* that looks for k groups that optimize affinity, with that learning potential value as a constraint.

Our third contribution is algorithmic. We present a suite of scalable algorithms that form groups to maximize learning potential and optimize affinity within *constant approximation factors*. To attain their approximation guarantees, these algorithms assume that affinity satisfies triangle inequality [88]. Many similarity/distance measures such as Jaccard distance and edit distance are known to satisfy metric properties and these properties are usually assumed to design algorithms with guarantees [88]. Our technical contributions are summarized in Table 2.2.

This chapter is organized as follows. Our model and problem are provided in Section 2.2. Section 2.3 discusses various optimization problems: optimizing learning potential alone, optimizing affinity alone, and combining the two. Section 2.4 contains our algorithms. Real and simulated experiments are reported in Section 2.5. The related work and conclusion are given in Sections 2.6 and 2.7.

2.2 Modeling and Problem Definition

We present our models following which we define the problems we tackle in this work.

Example 1. *We have a set of fact-checking tasks to be completed by* 12 *individuals with varying skills. We design questions to compute one skill per individual (e.g., on the British royal wedding) and obtain the skill values:* $\{2, 3, 1, 5, 6, 4, 9, 8, 10, 12, 14, 17\}$. *Each pair of individuals has an affinity that reflects how effectively they can collaborate based on their socio-demographics. Therefore, there are* $\binom{12}{2}$ *pairs of affinities forming a complete graph. We show a subset of that graph in Table 2.1 where each worker is identified by her skill. Our goal is to divide the workers into* 3 *equi-sized groups of* 4 *members each.*

2.2.1 Modeling

Group. A group is a set of individuals who will complete a task together. The group size is constant throughout task completion.

Table 2.1 Partial Affinity Table for Example 1

member	2	3	1	5	6	4	9	8	10	12	14	17
2	-	20	-	-	-	-	3	-	-	-	11	17
3	20	-	13	-	-	-	17	-	-	-	11	4
1	-	13	-	-	-	-	10	-	-	-	14	21
...												
17	17	4	21	-	-	-	-	-	-	-	6	-

Skill. Each individual has an approximated skill reflecting an ability to perform a task. We obtain skills from standard tests and questionnaires to assess expertise level. Other approaches such as inferring skills from completed tasks [115], are also possible.

Affinity. Between every pair of individuals working on a task, affinity captures how well they get along. We express affinity as a similarity measure (higher values are better). We assume affinity satisfies triangle inequality [88]. Group affinity is the aggregation of affinities between its members.

Learning Potential. We define the learning potential between two individuals as the difference between their skill values. The learning potential is not a metric since for the person with the higher skill value, we set it to 0 [3, 4]. The learning potential for a group is the sum of learning potentials of its members.

We have a set of n individuals who are working on a collaborative task. Each w_i has a skill value $w_i^s \in R \geq 0$ representing an ability to complete a task. Our goal is to group them into k equi-sized groups such that the aggregated learning potential and affinity of the groups are maximized. Before formalizing the problem, we investigate variants of learning potential and affinity.

Learning Potential Models Intuitively, the higher the learning potential of a group, the more likely its members will learn from each other. We examine two definitions.

Learning Potential - Diameter. We define LPD as the difference in skills between the most skilled and the least skilled members. This reflects that for a group, we are interested in maximizing the highest learning potential of the least skilled individual in that group.

$$\text{LPD}(g) = \max_{w_i \in g}(w_i^s) - \min_{w_j \in g}(w_j^s) \tag{2.1}$$

In Example 1, if we create the following three groups (we use a member's skill to represent her). $\mathcal{G} = \{g_1 = (2, 3, 1, 5), g_2 = (6, 4, 9, 8), g_3 = (10, 12, 14, 17)\}$ then, $\text{LPD}(g_1) = 5 - 1 = 4$, $\text{LPD}(g_2) = 9 - 4 = 5$, and $\text{LPD}(g_3) = 17 - 10 = 7$. The aggregated learning potential of the grouping $\mathcal{G}$ is 16.

Learning Potential - All. We define LPA as the sum of differences between each member's skill and that of all other members with higher skills.

$$\text{LPA}(g) = \sum_{w_i \in g} \sum_{(w_j \in g, s.t. w_i^s < w_j^s)} (w_j^s - w_i^s) \tag{2.2}$$

Given the previous grouping, we can compute $\mathcal{G} = \{g_1 = (2, 3, 1, 5), g_2 = (6, 4, 9, 8), g_3 = (10, 12, 14, 17)\}$, $\text{LPA}(g_1) = |1 - 2| + |1 - 3| + |1 - 5| + |2 - 3| + |2 - 5| + |3 - 5| = 13$, $\text{LPA}(g_2) = 17$, and $\text{LPA}(g_3) = 23$. The aggregated learning potential of the grouping under LPA is 53.

Affinity Models It is important to look at the effect of affinity on learning since members with higher affinities are likely to learn better from each other and

collaborate more effectively [62, 108, 116]. We examine two affinity variants.

Affinity - Diameter. We can formalize affinity as a complete graph $G = (V, E)$ where V is the set of n individuals and E contains weighted edges that correspond to the affinity between every pair of them. In this case, affinity satisfies triangle inequality. We refer to this case as AffD and define the affinity of a group as the minimum pairwise affinity of all its members as follows:

$$\text{AffD}(g) = \min_{w_i, w_j \in g} \mathit{aff}(w_i, w_j) \tag{2.3}$$

According to Example 1, if $g = \{2, 12, 14\}$ then
$\text{AffD}(g) = \min\{\mathit{aff}(2, 12), \mathit{aff}(2, 14), \mathit{aff}(12, 14)\}$, i.e., 4.

Affinity - Center. Affinity can also be defined based on the relationship between one member and all others. We refer to that as AffC and capture it as a graph $G = (V, E)$ where edges are defined between one designated member w_D and all others.

$$\text{AffC}(g) = \min_{w_D, w_j \in g} \mathit{aff}(w_D, w_j) \tag{2.4}$$

AffC captures the cases where the designated member is the least skilled or the most skilled. Similarly to AffD, in Example 1, if $g = \{2, 12, 14\}$ and the group center is 14 then
$\text{AffC}(g) = \min\{\mathit{aff}(2, 14), \mathit{aff}(12, 14)\}$ which corresponds to
$\mathit{aff}(12, 14) = 6$.

For instance, when the task is collaborative fact-checking (e.g., check facts related to the British royal wedding), LpA reflects that each group member will learn from other members with higher skills and AffD captures agreement between

the most two disagreeing members in the group. When LpA is combined with AffC, we can capture a task such as text editing where group members collaborate to correct grammar and spelling mistakes in text. In that case, one can intuitively assume that each group member will learn from other members with higher skills and that everyone must have affinity with the highest skilled member. Another example, AffD LpD captures a task where group members are asked to produce facts they believe to be true. In that case, the least skilled member learns from all others and all get along when stating facts.

2.2.2 Problem Definition

Given a set $W = \{w_1, ..., w_n\}$ of individuals with their corresponding skill values w_i^s, our goal is to form a grouping $\mathcal{G}$ that contains k equi-sized groups $g_1, g_2, ..., g_k$ that maximizes two objective functions, aggregated learning potential and aggregated affinity. More formally:

$$\begin{aligned} \underset{\mathcal{G}}{\text{maximize}} \quad & \sum_{i=1}^{k} LP(g_i), \sum_{i=1}^{k} \mathit{Aff}(g_i) \\ \text{s.t.} \quad & |\mathcal{G}| = k, \ |g_i| = \frac{n}{k} \end{aligned} \tag{2.5}$$

where $LP(g_i)$ (resp. $\mathit{Aff}(g_i)$) refers to any of the learning potential (resp. affinity) definitions in Section 2.2.1.

Since the two objectives are incompatible with one another, our problem qualifies as *multi-objective*. Upon examining the learning potential expressions, we notice that these are polynomial time solvable problems, simply because the primary operation that these problems require is sorting. We present exact algorithms for the two learning potential problems in Section 2.3.1. The complexity of our problem lies within the affinity structure and the group size constraint. One way to solve our

Table 2.2 Static Peer Learning NP-Hard Problems and Technical Results

Problem	Algo.	Approx.	Time
(AffC LpD)	GrAffC-LpD	exact LpD, 3 AffC	$O(klogn + nlogk)$
(AffC LpA)	GrAffC-LpA	exact LpD, 3 AffD	$O(nlogn)$
(AffD LpD)	GrAffD-LpD	exact LpA, 6 AffC	$O(klogn + nlogk)$
(AffD LpA)	GrAffD-LpA	exact LpA, 6 AffD	$O(nlogn)$

bi-objective optimization problem is therefore to transform it into a single-objective problem with constraints. We can rewrite Equation (2.5) as follows:

$$\begin{aligned} \underset{\mathcal{G}}{\text{maximize}} \quad & \sum_{i=1}^{k} Aff(g_i) \\ \text{s.t.} \quad & \sum_{i=1}^{k} LP(g_i) \geq OptLP \\ & |\mathcal{G}| = k,\ |g_i| = \frac{n}{k} \end{aligned} \tag{2.6}$$

where *OptLP* is the optimal solution for learning potential maximization. Essentially, we are interested in finding a solution for the affinity objective on the Pareto front, that has the highest learning potential. In Section 2.4, we present approximation algorithms that find a feasible grouping (that maximizes learning potential) and offer provable constant approximation for affinities.

2.3 Optimization

In this section, we first study how to optimize each of our two objectives individually, learning potential and affinity, and in the last subsection we begin studying our bi-objective optimizations by translating them into constrained optimization problems. This exercise has many benefits - (a) it offers a deeper understanding of the individual

problems and (b) it provides perspective on how to combine them and design scalable solutions with provable guarantees (refer to Section 2.4).

2.3.1 Optimizing Learning Potential

Our algorithmic endeavor begins by first describing solutions to group formation that maximize learning potential (LP) alone. Once we obtain the optimal LP value, we use that as a constraint when optimizing affinity (Equation (2.6) in Section 2.2.2). Fortunately, both LP problems are computationally tractable, and we present efficient algorithms that form a grouping with exact solutions. While different, our algorithms are designed in the same spirit as those designed to solve the value-based group formation [3] and the p-percentile partitioning problem [4]. A central idea to those algorithms is to create a grouping based on sorting group members on skill values.

Learning Potential LpD We want to form k groups that maximize the aggregated learning potential which in LpD is the maximum pairwise skill difference (Equation (2.4)). LpD of a group is always determined by a single pair of its members, the least skilled and the most skilled ones. Therefore, if we have to form a single group, we just need to select the most and least skilled members and make them part of that group. The other members in the group could be any as their participation does not increase or decrease the LpD value. This seemingly simple logic sufficiently extends to forming k groups. To form k groups, we need to find two buckets with a total of $2k$ people, the most skilled bucket containing the k highest skilled workers (the i-worker in that bucket is referred to as $w_i^{s.high}$), and the least skilled bucket containing the k least skilled workers (the i-worker in that bucket is referred to as $w_i^{s.low}$). We can then form k pairs by grouping one member in the least skilled bucket with one in the most skilled bucket and placing them in the same group. The remaining $n - 2k$ workers can be distributed arbitrarily across the k groups, while keeping the group size the same (pseudo-code in Algorithm 1).

Applied to Example 1, this is akin to forming the least skilled bucket with participants of skill values $\{1, 2, 3\}$, the most skilled one with values $\{12, 14, 17\}$, and forming three pairs, each one representing a group of size 2, by pairing members across the buckets. We can state the following theorem:

Theorem 1. *Any pairing across the least skilled and most skilled buckets produces the optimal aggregated value for* LpD*.*

Proof. (sketch) Consider the set of k least skilled members and k most skilled members. It is easy to see that changing the assignment of the least skilled members would not change the overall sum of the skill difference. LpD of the grouping is:

$$OPT\text{LpD} = (w_1^{s.high} - w_1^{s.low}) + (w_2^{s.high} - w_2^{s.low}) + \\ \ldots + (w_k^{s.high} - w_k^{s.low})$$

Indeed, any possible grouping across the buckets over these $2k$ members will not affect the sum, and thus the LpD value. □

Based on Theorem 1, we can state that multiple groupings maximize the LpD value. This corollary is important, because it provides intuition on the challenges that arise when combining affinity with learning potential.

Corollary 1.1. *There are* $\frac{k! \times (n-2k)!}{(n/k-2)!^k}$ *possible groupings to maximize* LpD*.*

Proof. The members in the highest and the lowest skilled buckets could be paired in $k!$ groupings. The remaining $(n-2k)$ members are to be placed over $(n/k-2)$ positions in each group, and a total over k groups. This gives rise to $\frac{k! \times (n-2k)!}{(n/k-2)!^k}$ groupings. □

Lemma 1. *Computing one optimal grouping for* LpD *takes* $O(n + klogn)$*.*

Proof. If we maintain a bottom-up heap like data structure, finding the top-k and bottom-k members (based on skills) will take $O(n + klogn)$ time. □

Learning Potential LpA The LpA of a group is the sum of skill differences between every member with every other more skilled member (Equation 2.3). The LpA of a set of k groups is the sum over the LpA of each group. What becomes

Algorithm 1 Algorithm to maximize LpD

input: set of workers W, k

output: a grouping $\mathcal{G}$, OPTLpD

procedure `LpD`(W, k)

OPTLpD $\leftarrow 0$

create highest and lowest skill buckets with k workers each

$\mathcal{G} \leftarrow$ a set of k empty groups

for i in $(1, ..., k)$ **do**

pick $w_m \in most\_skilled$ and $w_l \in least\_skilled$

$g_i \leftarrow \{w_m, w_l\}$

$W \leftarrow W \setminus \{w_l, w_m\}$

OPTLpD $\leftarrow OPT$LpD $+ (w_m^s - w_l^s)$

end for

while W is not empty **do**

Assign $w_i \in W$ in g_i, s.t $g_i \leq n/k$.

end while

end procedure

intuitively apparent is that if one has to form one group to maximize LPA, one should always group the most skilled member with the remaining less skilled ones. This logic extends to creating k groups by sorting members on skills (in increasing or decreasing order), and creating n/k buckets, each with k members. To form a group of size n/k, we choose a member from each bucket and repeat this process k times.

Using Example 1, this is akin to sorting the skills of the participants and forming a total of four buckets:

$$\{1, 2, 3\}, \{4, 5, 6\}, \{8, 9, 10\}, \{12, 14, 17\}$$

We form the first group by arbitrarily selecting one member from each of these 4 buckets, for example, those with skills $\{1, 4, 8, 12\}$. Then we repeat the process twice to get the two other groups, e.g., $\{2, 5, 9, 14\}$ and $\{3, 6, 10, 17\}$.

This algorithm turns out to be optimal - moreover, just like for LPD, all possible groupings across n/k buckets are permissible and will produce the same optimal LPA value.

Theorem 2. *Any grouping across the n/k buckets produces the optimal aggregated value for* LPA*.*

Proof. The proof is very similar to the proof of LPD. Consider a grouping that we get by running the above algorithm. We sort the members based on their skill values and we create x buckets. The first k workers will go into the first bucket and so on. We create a group by choosing 1 member of each bucket. After k iterations, we obtain our grouping. Without loss of generality, assume that the highest skilled member of group i has the skill value of S_i and the other members have s_i^j where j denotes the bucket that this member has been chosen from. Consider the grouping $G = \{g_1 = (S_1, s_1^1, s_1^2), g_2 = (S_2, s_2^1, s_2^2), \ldots, g_k = (S_k, s_k^1, s_k^2)\}$. We can show that swapping two members from the same bucket will not change the LPA optimal value. Without loss of generality, consider a new grouping G' where the position of s_i^j and $s_{i'}^j$ is swapped. Now consider the LPA value for G and G'. We can show that the difference between these two scores is 0. This holds when we pick the lowest skilled member.

$$LP(\mathcal{G}) = \sum_{i=1}^{k} \sum_{j=1}^{x-1} (S_i - s_i^j) \tag{2.7}$$

In the Equation (2.7), the only difference between G and G' is in group i and i'. More accurately, we need to show that $(S_i - s_i^j) + (S_{i'} - s_{i'}^j)$ in the grouping G is equal to $(S_i - s_{i'}^j) + (S_{i'} - s_i^j)$. It's easy to see that these two are identical hence the proof. □

Corollary 2.1. *There are $k!^{n/k}$ possible groupings for* LPA.

Proof. Since we have n/k buckets with k members each, we have $k!$ ways of choosing a member for every group. This results in $k!^{n/k}$ different groupings. □

Lemma 2. *Computing one optimal grouping for* LPA *takes $O(nlogn)$.*

Proof. The algorithm is dominated by the sorting time over n members, which is $O(nlogn)$. □

2.3.2 Optimizing Affinity

Since we express affinity as similarity, optimizing it amounts to minimizing distance. AFFC takes the affinity graph over n members and a subset of k members as centers (teachers) as input, and intends to partition the remaining $n-k$ members into k equi-sized groups such that the sum of the radii (maximum distance between the center and a member in each group) is minimized. AFFD, on the other hand, only takes the affinity graph over n members and k to partition the members into k equi-size groups, such that, the sum of the diameters (diameter of a group is the maximum pairwise distance in the group) of the grouping is minimized. We first present some theoretical results on the hardness of these two problems.

Even though it intuitively appears that AFFC is an easier problem than AFFD, both problems are NP-hard. The hardness of AFFC is due to the group size constraint.

Theorem 3. *The decision version of the* AFFC *problem in NP-Complete.*

Proof. (sketch) It is easy to see that the problem is in NP. To prove the NP-hardness, the reduction is straightforward (there is a one-to-one correspondence) if we consider the uniform p-centered min-max partition problem as the source problem, which is proved to be NP-hard [89] for general graphs. We omit the rest of the details for brevity. □

Theorem 4. *The decision version of the* AffD *problem is NP-Complete.*

Proof. For simplicity, we consider a simpler scenario, where affinity (distance) is binary - 0/1.

For this binary scenario, the decision version of the Affinity-All problem is as follows: given a set of n members, is there a grouping of k equal sized groups, such that the sum of diameters of the grouping is k?

It is easy to see that the problem is in NP. To prove NP-hardness, we use the well-known exact cover by 3-Sets (X3C) for reduction. The decision version of X3C is as follows: given a finite set X with $|X| = 3q$ elements and a collection C of 3-element subsets of X, does C contain an exact cover for X, that is, a sub-collection $C' \subseteq C$, where C' contains exactly q subsets, such that every element of X occurs in exactly one member of C'?

Given an instance of X3C, we reduce it to an instance of AffD in the following way: Each element in X is a member. Therefore, the total number of members $n = 3q$. The affinity graph is a weighted complete graph among the n members and it involves adding edge weights between every pair of members. Each subset of 3 elements in C represents 3 members in this graph, and the edge weights between them gets the value 1. This is a polynomial time operation and the number of operations involved in this is the size of C. After that, we need to resolve all the edge weights that are across the subsets. For that, we start considering all the triangles with some unresolved edge weights.

There are three possible scenarios to handle in this process: (1) all 3 edges unresolved, (2) 2 edges unresolved, (3) 1 edge unresolved. For the first case, we can safely add the weight of 0 to each of such edge (this happens when the subsets are fully disjoint). For the second scenario (this happens when 2 nodes in the triangle are part of the same subset but the third node is part of a different subset), one of the unresolved edges gets 1 and the other gets 0. Finally, for the third scenario (this happens when one member in the triangle is part of both subsets), the unresolved edge gets a 0. We note that this step is again fully polynomial and takes at most $O\binom{n}{3}$ time. After completing this step, we will have assigned all the edge weights in the affinity graph. It could be shown that the affinity graph constructed this way satisfies triangle inequality.

After that, we set $k = q$. Now, the reduction is complete. Notice that X3C $\leq_P$ AffD. There exists a solution to the X3C problem if and only if a solution to our instance of AffD problem exists with the total diameter value q (or k). This completes the proof. □

2.3.3 Optimizing Affinity with Learning Potential as a Constraint

Finally, we turn our attention to studying the four constrained optimization problems, with the objective to optimize affinity, while satisfying the learning potential value obtained from the algorithms in Section 2.3.1. Since affinity is modeled as a distance,

our goal is to minimize that distance, considering the underlying affinity structure. Recall that LpD and LpA are polynomial-time problems and that we presented exact solutions for both in the previous section. To ease exposition, we will henceforth call the optimal values obtained for the LP problems as Lp-* (it is either LpD or LpA). Our focus now is to study how to optimize Aff-* (AffC or AffD), with Lp-* as constraints.

Theorem 5. *The decision versions of* Aff-* Lp-* *problems are NP-Complete.*

Proof. (sketch): Since the AffC and AffD problems are individually NP-hard, the hardness remains true when the LP constraints are added. We omit the details for brevity. □

Our technical deep dive into these four problems is described in Section 2.4. We develop greedy algorithms that are extremely lean in computational time with constant approximation factors. Table 2.2 summarizes the 4 problem variants and our technical results.

2.4 Constrained Optimization

We now present a suite of algorithms with theoretical guarantees to solve the four different variants of optimizing affinity with learning potential as a constraint. As our problems are NP-hard, we develop approximation algorithms that are scalable and bear theoretical guarantees.

Our algorithms are greedy and use the following intuition: Lp-* problems are first solved and these solutions produce an intermediate grouping that has the optimal LP values. Our algorithms start from these solutions and greedily choose the rest of the members to output the final grouping that is guaranteed to have optimal LP values and provable constant approximation factors for affinity.

2.4.1 Algorithm for AffC LpD

Our discussion of LpD in Section 2.3.1 stated that only $2k$ members (k most skilled and k least skilled) are needed to produce the optimal grouping. Our proposed Algorithm GrAffC-LpD starts from there (recall Algorithm 1) - that is, it first identifies the $2k$ members which will guarantee the optimal LpD value (thereby satisfying the constraint of the optimization problem). These outputs are referred to as boundary members. That means, 2 members in each group are decided by now and a total of $2k$ members are decided for the grouping. In each group, the highest skilled member is the teacher and acts as the center for that group since AffC is formalized as the maximum distance between that member and anyone else in the group. The rest of the grouping is performed in a greedy manner. For the remaining $n - 2k$ members, all we have to do is assign them to their respective closest center. Since each group has a size constraint, this greedy assignment may lead to sub-optimality - since for a member w_i, the group with the closest distance between its center and w_i may have reached its size and w_i may need to be assigned to a group such that the distance between w_i and its center c_i is larger (potentially worsening the AffC value). But as we shall prove later, this greedy assignment cannot be arbitrarily worse, since affinity between members satisfies triangle inequality (pseudo-code in Algorithm 2).

Going back to Example 1, based on GrAffC-LpD, initially we will have the following partial grouping : $g_1 = \{1, 12\}$, $g_2 = \{2, 17\}$, $g_3 = \{3, 14\}$. After that, Algorithm GrAffC-LpD greedily adds 2 more members in each group that are not yet part of any group. For example, for g_1, it will add the member who has the highest affinity with 12 and the process will repeat.

Theorem 6. *Algorithm* GrAffC-LpD *accepts a* 3 *approximation factor to optimize* AffC.

Proof. (sketch) Without loss of generality, let us assume a worst case scenario of 3 groups as shown in Figure 2.3, where members p_1, p_2, p_3 dictate the AffC score of these three groups that are centered around c_2,c_3,c_1, respectively. Because of the

Algorithm 2 Algorithm GrAffC-LpD

input: a set W of n participants, k groups

output: a grouping $\mathcal{G}$

B = Call LpD(W, k)

C = the k highest skilled members in B that are k centers

Assign $w_i \in \{W - B\}$ to the closest center c_j s.t., $|g_j| \leq n/k$

greedy assignment, p_1 is assigned to the center c_2, p_2 is assigned to c_3, but at the end because of the size constraints p_3 gets a really bad assignment of c_1. Distance between p_1 and c_2, i.e., $d(p_1, c_2) = \alpha_1$, similarly $d(p_2, c_3) = \alpha_2$, and $d(p_3, c_1) = \alpha_3$. The optimal assignment would have given rise to a different assignment though (as shown in the dotted line), where $p_1 \in c_1, p_2 \in c_2, p_3 \in c_3$. $d(p_1, c_1) = \beta_1, d(p_2, c_2) = \beta_2$, and $d(p_3, c_3) = \beta_3$. Let OPT denote the optimum AffC value, such that $OPT = \beta_1 + \beta_2 + \beta_3$. Of course, $\alpha_1 + \alpha_2 + \alpha_3 \geq \beta_1 + \beta_2 + \beta_3$. But it is easy to notice that

$$\alpha_3 \leq (\alpha_1 + \beta_1 + \alpha_2 + \beta_2 + \beta_3) \tag{2.8}$$

$$\alpha_3 \leq (2\beta_1 + 2\beta_2 + \beta_3) \tag{2.9}$$

Because of the triangle inequality, this is indeed true, $\alpha_1 + \beta_1 \leq 2\beta_1$ (because $\alpha_1 \leq \beta_1$ what the greedy algorithm GrAffC-LpD will ensure. Therefore, we can write that,

$$\alpha_1 + \alpha_2 + \alpha_3 \leq (3\beta_1 + 3\beta_2 + \beta_3) \tag{2.10}$$

$$\leq 3(\beta_1 + \beta_2 + \beta_3) \tag{2.11}$$

$$\leq 3 \times OPT \tag{2.12}$$

It is easy to notice that this argument easily extends to an arbitrary number of groups. Hence the proof. □

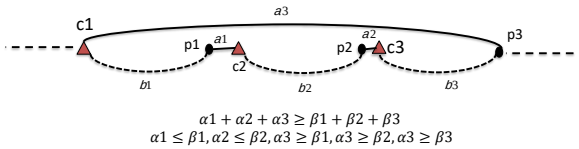


Figure 2.3 Upper bound of approximation factor.

Corollary 6.1. *Running time of* GrAffC-LpD *is* $O(klogn + nlogk)$

Proof. Finding the boundaries takes $O(klogn)$ time. For the remaining $n - 2k$ members, we maintain a $logk$ size heap and perform min heap operation over them. This results in $O(klogn + nlogk)$ running time. □

2.4.2 Algorithm for AffC LpA

The idea of this greedy algorithm GrAffC-LpA is similar to the previous one, that is start with the partial grouping that LpA returns. However, unlike LpD, LpA creates a set of n/k buckets (or partitions) (see Section 2.3.1) that dictate that forming an intra-partition group is forbidden, and any possible inter-partition groups will result in the same optimal LpA value. In fact, as Corollary 2.1 suggests, there are $(k!)^{n/k}$ possible groupings that yield the optimal LpA value. The challenge is to find one grouping that optimizes affinity.

GrAffC-LpA begins by invoking the LpA procedure to compute n/k buckets that are sorted in increasing order of skills. It selects the teachers as the k members in the last buckets (they are the centers and they have the k highest skills). After that, GrAffC-LpA operates in a greedy fashion. For the remaining $n - k$ members, it follow a similar approach as Algorithm GrAffC-LpD. At each iteration, it chooses a member from the bucket and assigns it to the closest center. In Example 1, we create a total of 4 buckets:

$$\{1, 2, 3\}, \{4, 5, 6\}, \{8, 9, 10\}, \{12, 14, 17\}$$

Next, we assign each high skilled member in the last bucket to a group and consider them as centers. As an example, $g_1 = \{12\}$, $g_2 = \{14\}$, and $g_3 = \{17\}$. Next, for the members of the first bucket, based on their affinities, 1 is assigned to g_1, 2 to g_3, and 3 to g_2. The process continues until all the buckets are empty.

Since each group has a size constraint, this greedy assignment may lead to sub-optimality - as it happened in GrAffC-LpD. However, this greedy assignment

cannot be arbitrarily worse, because affinity between members satisfies triangle inequality.

Theorem 7. *Algorithm* GrAffC-LpA *accepts a* 3 *approximation factor for* AffC.

Proof. The proof is very similar to the proof given for GrAffC-LpD. We omit the details for brevity. □

Corollary 7.1. *Running time of Algorithm* GrAffC-LpA *is* $O(nlogn)$.

Proof. Sorting the skill values takes $O(nlogn)$, dictating the complexity of this algorithm. The rest of the algorithm takes $O(nlogk)$ running time, akin to what is described in Corollary 6.1 - but sorting dominates the overall running time. □

2.4.3 Algorithms for AffD Lp-*

There is an interesting relationship between AffC and AffD that merits further delineation. In the AffC problem, we want to minimize the distance from a center to the farthest member in the group (i.e., minimizing the radii). In AffD, we do not have any member as the center, rather we are interested to form groups to minimize the maximum distance (i.e., the diameter). The next theorem states that any solution for the former problem is a solution for the latter that is at most 2 times worse. Based on that, the greedy algorithms in Sections 2.4.1 and 2.4.2 could be used to solve AffD,Lp-* problems. We refer to these algorithms as GrAffD-LpD and GrAffD-LpA, respectively for the AffD LpD and AffD LpA problems. GrAffD-LpD is identical to GrAffC-LpD, and GrAffD-LpA is identical to GrAffC-LpA operationally. Their respective running times are the same as their counterparts.

Theorem 8. *Any solution for* AffC *gives a* 2 *approximate solution for* AffD.

Proof. (sketch:) Consider a solution to AffC. Consider that for a group g_i, the distance from the center c_i to the farthest member is α_i. Assume that w_i is the member with this distance equal to α_i. Based on the triangle inequality, we can easily show that in the worst case, there is another member $w_j \in g_i$ where $d(w_i, w_j) < d(w_i, c_i) + d(w_j, c_i) < 2 \times \alpha_i$. Hence, any algorithm that solves AffC also solves AffD with a 2 approximation factor. □

Corollary 8.1. *Algorithms* GrAffD-LpD *and* GrAffD-LpA *have a* 6 *approximation factor for* AffD.

Proof. (sketch): Proofs are direct derivatives of Theorem 8. □

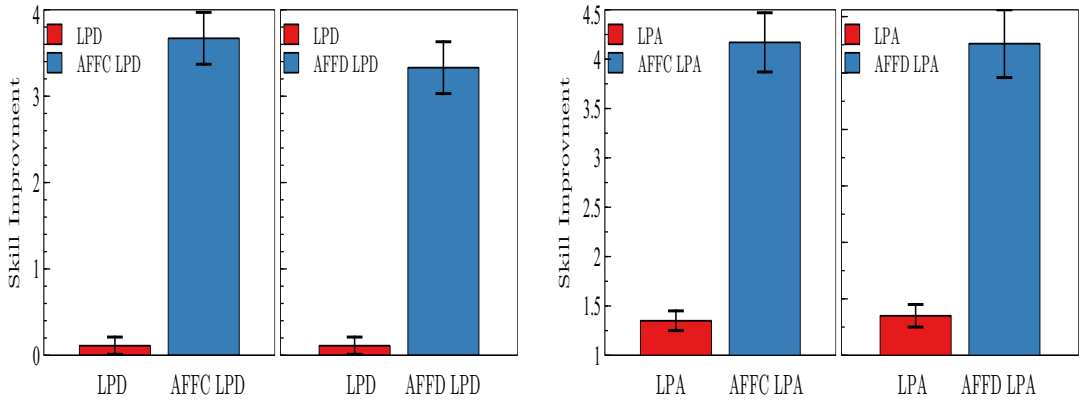
2.5 Experimental Evaluations

Our experimental effort goes in two directions. In Section 2.5.1, we involve actual human workers and collaborative tasks. In the remaining three subsections, we describe synthetic data experiments.

2.5.1 Real Data Experiments

These experiments examine *if affinity brings added utility in peer learning.* They are designed for *collaborative fact-checking and fact-learning* and involve Amazon Mechanical Turk (AMT) workers in three stages: 1) pre-task skill assessment, 2) task completion in a group, and 3) post-task completion skill assessment.

Experimental setup and design. One of our fact-checking tasks is about the British royal family. These experiments are run in three different stages. We first run a pre-task skill test to assess the skills of each worker using eight true/false questions for which we know the true answer. We set questionnaires for that purpose. Next stage, we set up a collaborative document that contains five facts about the royal family and where workers in the same group can collaborate, comment, and edit. Workers are asked to discuss if these facts are true, and provide further evidence that support their answer. Finally, each worker takes a post-task completion skill test that is again 8 true/false questions on the royal family. We also explicitly ask each worker what they have learned by completing that task. We design similar studies for fact-learning, where workers have to actually propose facts with supporting evidence. To keep this experiment tractable, we form groups of size 3 and run 3 different samples of the same experiment. This also allows us to analyze results with statistical significance.



(a) (b)

Figure 2.4 Skill improvement with and w/o affinity in LPD (a) and LPA (b).

We pay each worker $2, if all three stages are completed. Each experiment must run over a window of 24 hours to account for differences in time.

Affinity calculation. For simplicity, we capture affinity as the Euclidean distance between their socio-demographic data (specifically, age, country, education) obtained from AMT. There exists other sophisticated measures such as MBTI tests for project-based learning [108]. We nevertheless note that the simple measures that we have used have been shown to be useful affinity indicators [116].

Evaluation criteria. In order to evaluate the effectiveness of affinity in peer learning, we measure the difference between each member's skills before and after task completion, and refer to that as *skill improvement*. We also measure the average number of comments in each group and the quality of contributions. These two criteria help us interpret worker engagement and skill improvement.

Summary of results. We compare with and without affinity counterparts for each problem variant (e.g., AFFC LPD with LPD). Our results confirm that affinity improves learning potential substantially with statistical significance. Figure 2.4(a) contains the average skill improvement comparison of LPD with and without affinity and shows the important role of affinity. This is consistent with LPA (Figure 2.4(b)).

We also observe that LPA has higher improvements, possibly because facts have many facets that one learns from more skillfull peers. Additionally, Figure 2.5 presents two sample interactions between two workers during task completion. In the first question, *Worker 2* provides a new piece of information about the Queen, which is set as one of the questions in the post-task skill assessment. This additional information helps *Worker 1* improve her skill during post-task assessment.

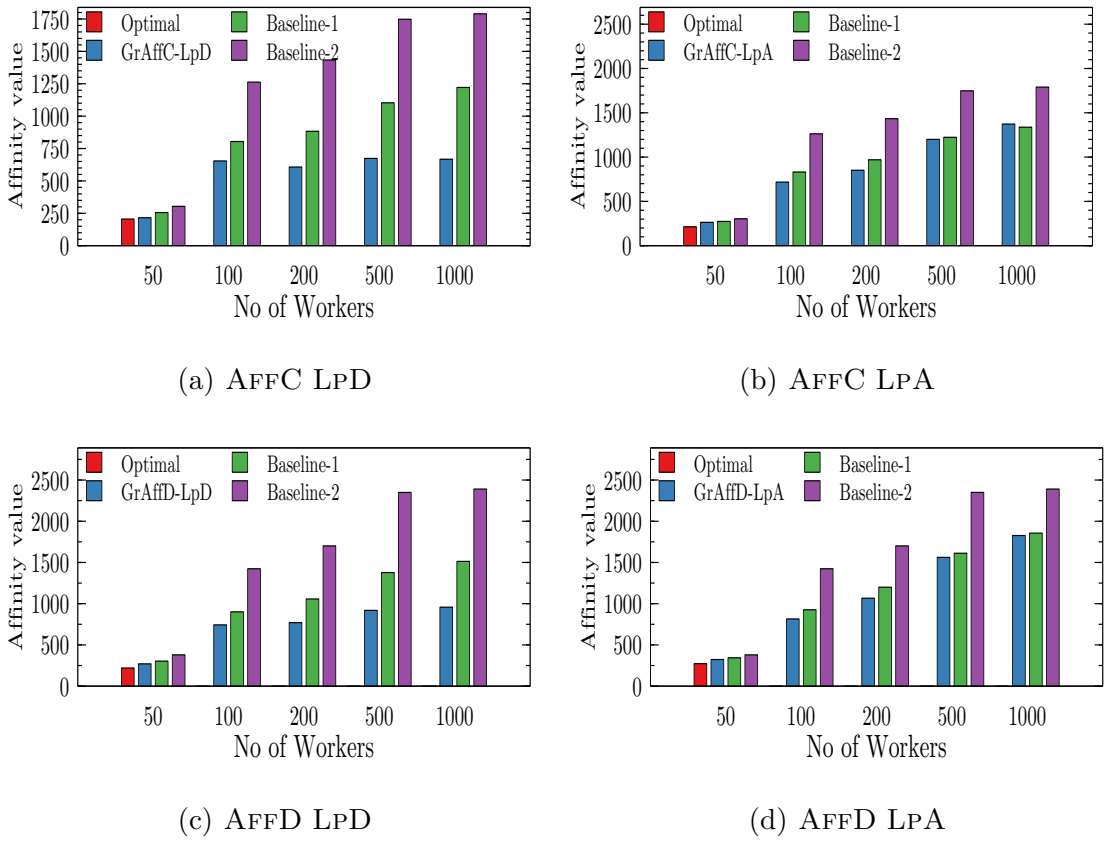
The Queen does not need a passport to travel True or False ?	• Worker 1: True. All British Passports are issued in the Name of Her Majesty, The Queen. • Worker 2 : I found an article which agrees with your findings. **Fun fact: she also doesn't need a driver's license or a license plate on her car.**
Members of the royal family have to accept absolutely all gifts.	• Worker 1 : (Mostly false; Large true in practice.) While I couldn't find any law requiring the Royals to accept all gifts. • Worker 2 : I found an article which says they make a list of all gifts they receive throughout the year and release it publicly. In addition, they donate many of their gifts.

Figure 2.5 A sample worker interaction.

Finally, we anecdotally observe that higher learning potential yields higher quality task outcomes. On average, quality (computed as the average number of facts correctly identified by the groups), is higher for groups built with affinity (4.3 facts out of 5 are correct), compared to their counterparts built without affinity (3.9 out of 5).

2.5.2 Synthetic Experiments Setup

These experiments evaluate the qualitative guarantees and the scalability of our algorithms. All algorithms are implemented in Python 3.6 using Intel Core i7 4GHz



(a) AffC LpD (b) AffC LpA

(c) AffD LpD (d) AffD LpA

Figure 2.6 AFF-* LP-* values varying n for Normal distribution.

CPU and 16GB of memory and Windows operating system. All numbers are averages of 10 runs.

Implemented Algorithms. The closest works to ours [3,4] do not consider affinity and cannot be used for synthetic data experiments. Hence, we implement three additional baselines:

Optimal. An Integer Linear Programming (ILP) algorithm that produces optimal values for Aff-* Lp-*. We use the PuLP library to build the ILP model

and formalize the ILP solution using the below formulation.

$$\underset{\mathcal{G}}{\text{optimize}} \sum_{i=1}^{k} \textsc{Aff}\text{-}^*(g_i)$$

$$\text{s.t.} \sum_{i=1}^{k} LP(g_i) \geq \textsc{Lp}\text{-}^*$$

$$\textsc{Aff}\text{-}^*(g_i) = \sum_{j=1}^{n} \sum_{m=1}^{n} x_{i,j} * x_{i,m} * \mathit{Aff}(w_j, w_m), \forall i = 1 \ldots k$$

$$\sum_{j=1}^{n} x_{i,j} = n/k, \forall i = 1 \ldots k$$

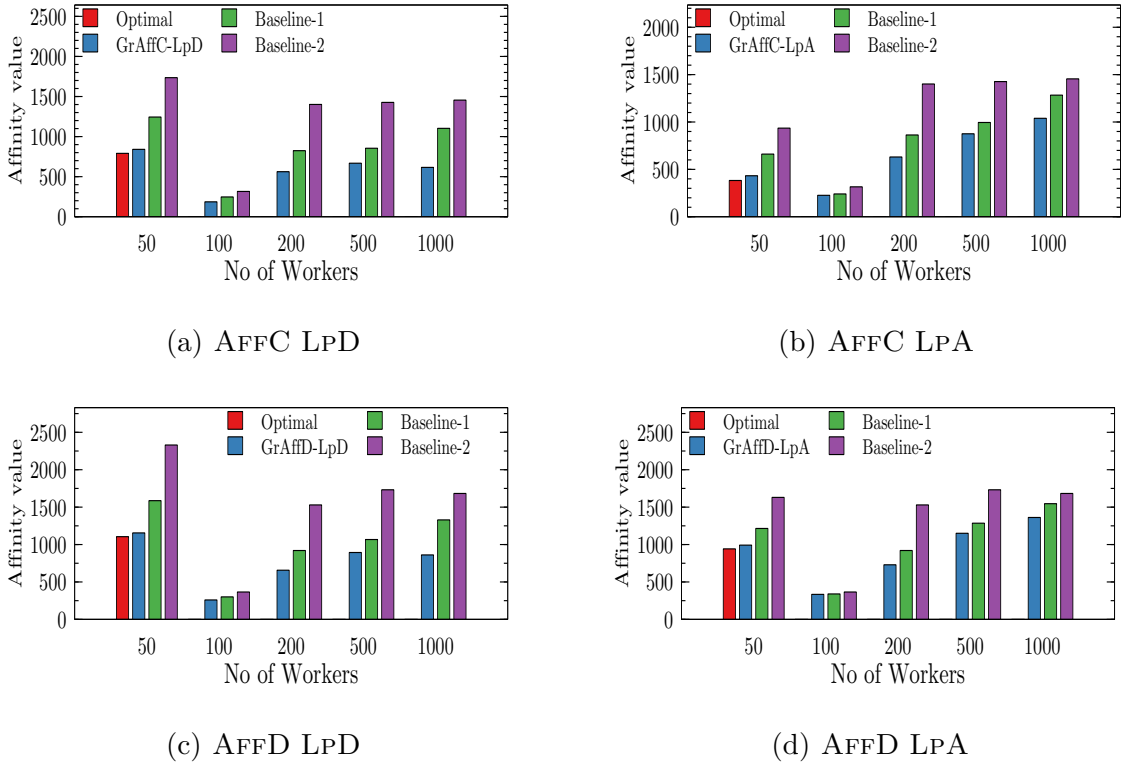
$$x_{i,j} = 0/1 \quad (i = 1...k \ \& \ j = 1 \ldots n)$$

The rationale behind implementing ILP is to demonstrate that the theoretical approximation factors of our algorithms hold in practice. Since ILP is NP-hard, the algorithm does not terminate beyond $k = 3$ and $n = 50$.

Baseline-1 (clustering-based). This baseline is motivated by the popular k-means algorithm. It starts with a random grouping and greedily swaps members across groups as long as that improves affinity, while satisfying group size. Once the grouping converges based on affinity, we check if it satisfies the optimum learning potential value (which could be derived efficiently in polynomial time). If not, we perform another set of swaps to move members across the groups until we find a grouping that reaches the optimal learning value.

Baseline-2. This is a simpler and efficient baseline. It first solves the learning potential problem and finds the seed members in each group that dictate the optimal learning value. The rest of the members are assigned randomly to groups by considering group size.

These solutions are compared with 4 of our algorithms GrAffC-LpD, GrAffC-LpA, GrAffD-LpD, GrAffD-LpA (refer to Section 2.4, whenever applicable).



(a) AffC LpD

(b) AffC LpA

(c) AffD LpD

(d) AffD LpA

Figure 2.7 AFF-* LP-* values varying n for Zipf distribution.

Experimental Setup. We simulate a group of workers with two functions that capture the relationship between skill and affinity. Specifically, there are two random number generators, one produces the skill of each member and the other generates pairwise affinities that satisfy triangle inequality.

We consider two skill and affinity distributions: (a) *Normal*, where the mean and standard deviations are set to $\mu = 100$, $\sigma = 20$, respectively; (b) *Zipf*, where the value of the exponent α is set to 1.5.

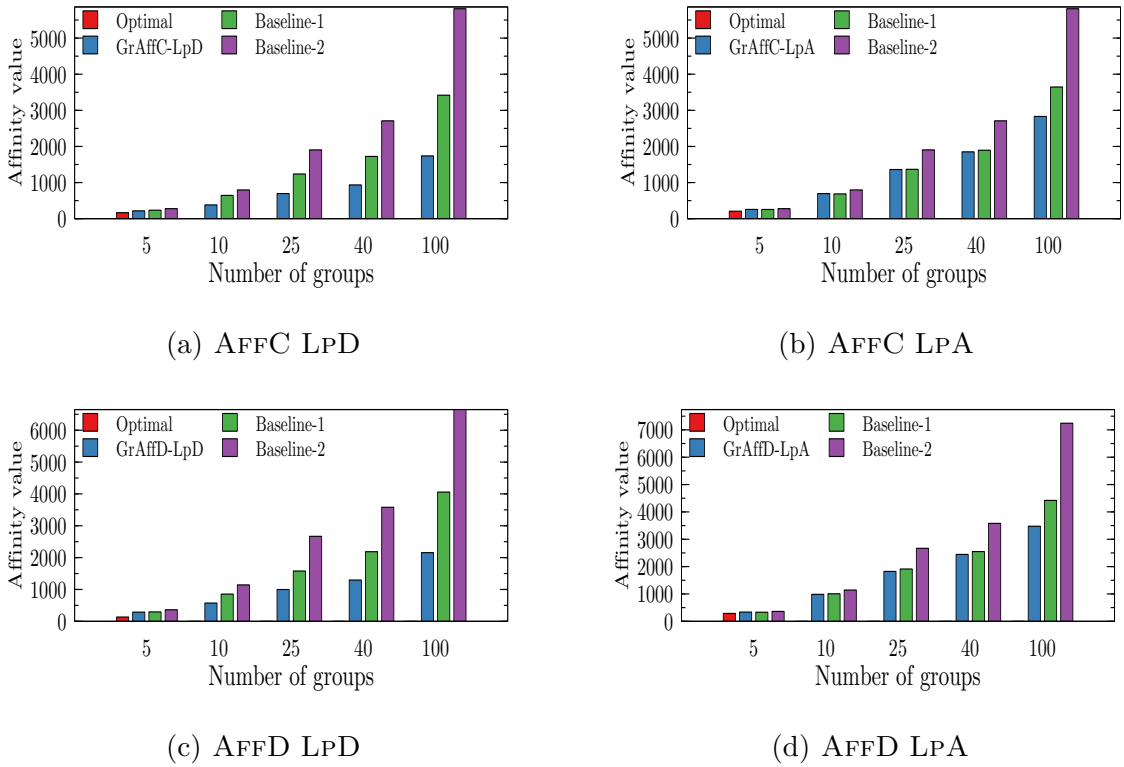
Parameters: We vary n (the total number of individuals), k (the number of groups), and the skill and affinity distributions.

Summary of Results.

- Our algorithms exhibit tighter approximation factors than the bounds we proved. Our algorithms also outperform the two baselines.
- The approximation factors of the algorithms with a Normal skill distribution are better than Zipf.
- All algorithms are highly scalable considering up to 10^6 members and 160 groups and only take seconds to run.

Table 2.3 Approximation Factors of GrAff*-Lp* Algorithms

Algo.	Parameters	App. Factor
(GrAffC-LpD)	$[n = 15, k = 3]$	1.13(0.12)
	$[n = 50, k = 3]$	1.23(0.02)
(GrAffC-LpA)	$[n = 15, k = 3]$	1.04(0.07)
	$[n = 50, k = 3]$	1.02(0.03)
(GrAffD-LpD)	$[n = 15, k = 3]$	1.21(0.14)
	$[n = 50, k = 3]$	1.31(0.04)
(GrAffD-LpA)	$[n = 15, k = 3]$	1.18(0.11)
	$[n = 50, k = 3]$	1.19(0.12)



(a) AffC LpD (b) AffC LpA

(c) AffD LpD (d) AffD LpA

Figure 2.8 AFF-* LP-* values varying k for Normal distribution.

2.5.3 Quality Experiments (Synthetic)

We assess quality by measuring the approximation factor and the objective function value. Both of these are described considering affinity, per our problem definition. LP values are always optimal (as the algorithms for LP are exact) and we skip those details for brevity.

Default parameter setting. Unless otherwise stated, k is set to 25 and n to 1000.

Comparison against ILP: Table 2.3 presents the approximation factor of the 4 algorithms on a small dataset generated from Normal Distribution. For all the 4 algorithms, the approximation factor in practice is very tight and the deviation is always between 1 and 2.

Varying n : Figure 2.6 reports the results of varying n for Normal distribution. Of course, ILP does not scale beyond $n = 50$, but it is easy to notice that for all the

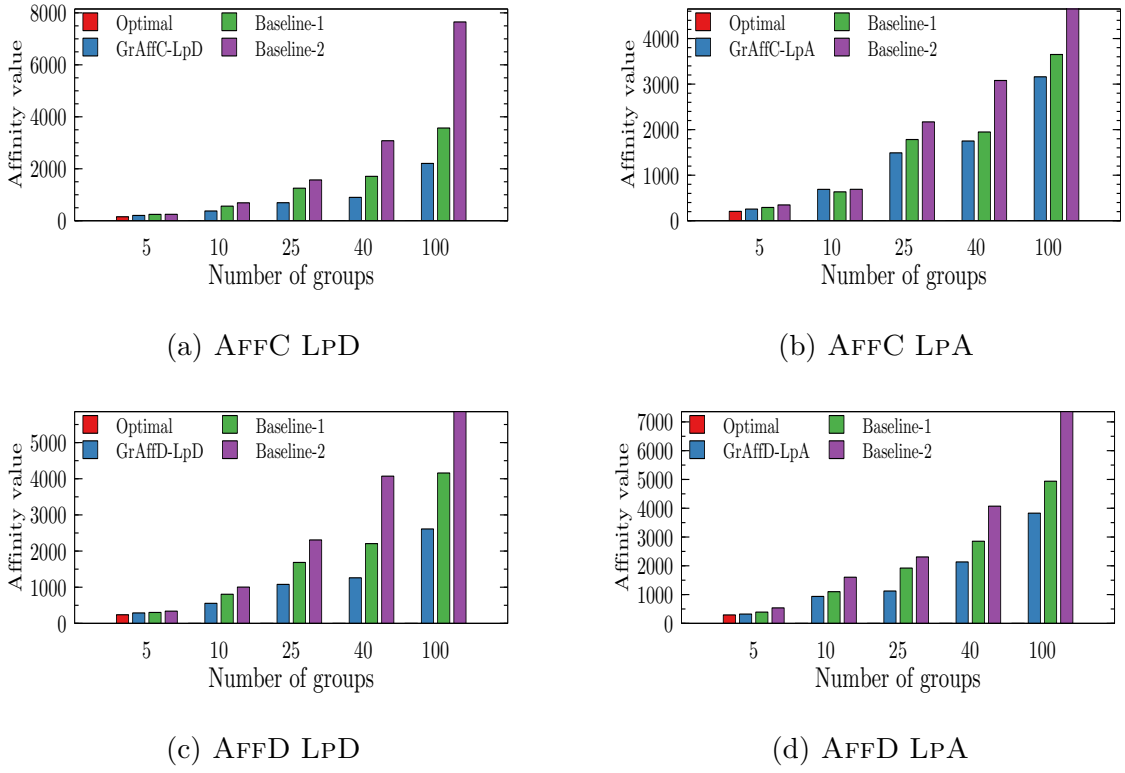
cases we could compare, our greedy solutions attain a very tight approximation factor (close to 1.5). Figure 2.7 shows the affinity values for Zipf distribution. Similar observations hold. Our proposed algorithms perform significantly better. There are two interesting observations in both Figures 2.6 and 2.7. Firstly, the grouping generated by our algorithm attains smaller objective value as the number of workers grow. Secondly, for Normally distributed data, we observe a consistent growth of objective value as the number of workers increases. This is not the case for the Zipfian distributed data. We conjecture that this is caused by the skew in the values generated from the Zipfian distribution. Some values are very large and others are vary small. Another important factor is that the data sampled from a Zipfian distribution consists of mostly duplicate values.

Varying k : Figures 2.8 and 2.9 present the results of varying k for Normal and Zipf distributions. The ILP for $k = 5$ is ran on $n = 50$. Our presented algorithms consistently outperform the other baselines. We observe that the change in k affects the objective value significantly more than change in the number of workers (n). We believe this is because larger k signifies more centers to assign workers to. Remember in Algorithm 2, we need to assign workers to the closest center. This means for larger values of k, we would diverge from the optimal solution easier. In fact, k impacts our algorithm more than n.

2.5.4 Scalability Experiments (Synthetic)

We measure running times and compare with **Baseline-1**. We exclude ILP since it does not scale, and **Baseline-2** since it produces inferior objective values. Running time is reported in seconds.

Default parameter settings. We found that Normal and Zipf skill distributions have identical running times for each variant of AFF-* LP-* problems. We also note that, as proved in Section 2.4.3, the running time of AFFD is identical to



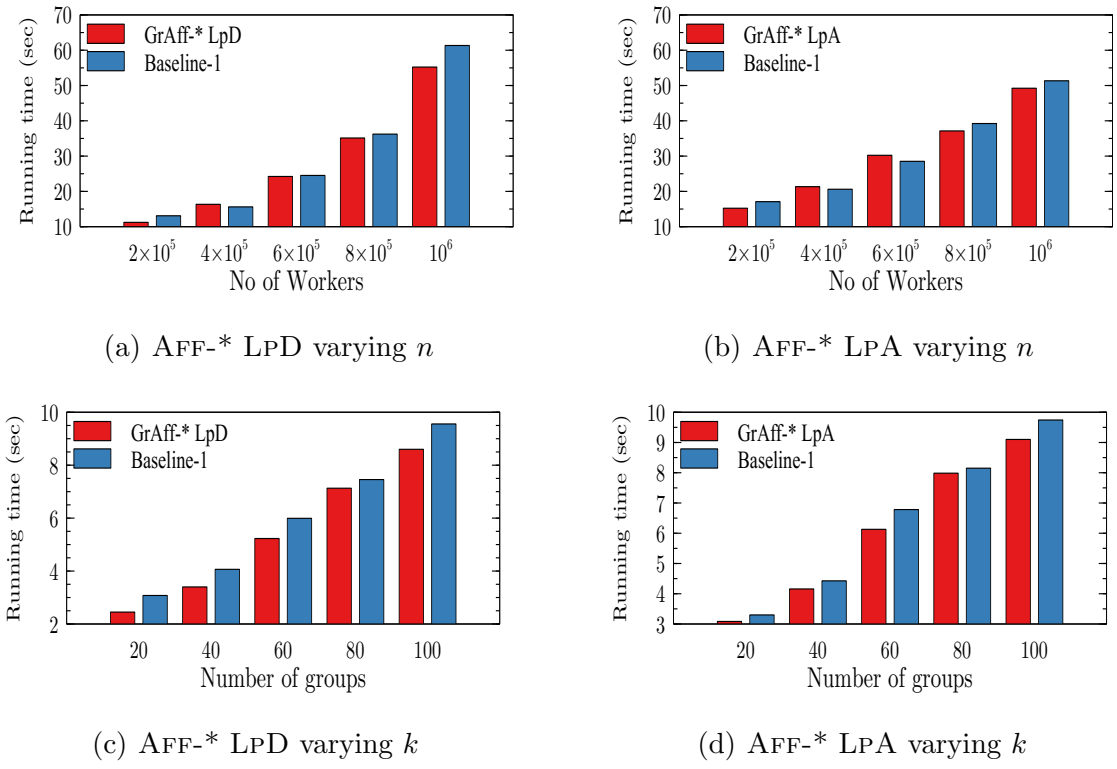
(a) AffC LpD

(b) AffC LpA

(c) AffD LpD

(d) AffD LpA

Figure 2.9 AFF-* LP-* values varying k for Zipf distribution.



(a) AFF-* LPD varying n

(b) AFF-* LPA varying n

(c) AFF-* LPD varying k

(d) AFF-* LPA varying k

Figure 2.10 Results of scalability experiment.

that of AFFC, considering their respective LP-* counterparts. Therefore, we only present results for AFF-* LPD and AFF-* LPA. We vary n and k with defaults set to $n = 100000$ and $k = 5$.

Results. Figure 2.10 presents results. Our algorithms are highly scalable and take seconds only. The algorithms run linearly with varying n and k which confirms our theoretical analysis.

2.6 Related Work

Our work studies computational aspects and relates to team formation and computer-supported learning.

Team formation was first studied to form a single group with one objective and later a 2-approximation algorithm was proposed for bi-criteria team formation in social networks [88]. In [14, 15], Anagnostopoulos et al. propose online algorithms for

the balanced social task assignment problem. Capacitated assignment was studied in a follow up work [94]. Generalized density sub-graph algorithms were later proposed [118]. [28, 116] study the problem of forming teams for task assignment considering affinity. In [123], the hardness of forming groups to optimize group satisfaction is studied under different group satisfaction semantics.

Unlike ours, none of these works study the problem of peer learning, hence their proposed modeling and algorithmic solutions do not apply.

Computer-Supported Collaborative Learning (CSCL). Social science has a long history of studying non-computational aspects of computer-supported collaborative learning [51, 54]. With the development of online educational platforms (such as, Massive Open Online Courses or MOOCs), several parameters were identified for building effective teams: (1) individual and group learning and social goals, (2) interaction processes and feedbacks [128], (3) roles that determine the nature and group idiosyncrasy [54].

To the best of our knowledge, the closest to our work are [2–4], where quantitative models are proposed to promote group-based learning, albeit without affinity.

Our work is grounded in social science and takes a computational approach to the design of scalable solutions with guarantees.

2.7 Conclusion

We examine online group formation where members seek to increase their *learning potential* via task completion with two learning models and affinity structures. We formalize the problem of forming a set of k groups with the purpose of optimizing peer learning under different affinity structures and propose constrained optimization formulations. We show the hardness of our problems and develop 4 scalable algorithms

with constant approximation factors. Our experiments with real workers demonstrate that considering affinity structures drastically improves learning potential, and our synthetic data experiments corroborate the qualitative and scalablity aspects of our algorithms.

CHAPTER 3

TASK DEPLOYMENT RECOMMENDATION WITH WORKER AVAILABILITY

3.1 Introduction

In crowdsourcing, task deployment is an important process that requesters undertake with very little help. Task deployment requires to specify not only the tasks, but also identify *appropriate deployment strategies*. A strategy involves the interplay of three dimensions: *Structure* (whether to solicit the workforce sequentially or simultaneously), *Organization* (to organize it collaboratively or independently), and *Style* (to rely on the crowd alone or on a combination of crowd and machine algorithms). A strategy needs to be commensurate to *deployment parameters* that are typically provided as thresholds on quality (lower-bound), latency (upper-bound), and budget (upper-bound). For example, for a sentence translation task, a task designer wants the translated sentences to be at least 80% as good as the work of a domain expert, in a span of at most two days, and at a maximum cost of 100$. Till date, the burden is entirely on requesters to design deployment strategies that are consistent with desired deployment parameters. Our effort in this chapter is to present a formalism and a computationally efficient algorithm to assist requesters by recommending k strategies that best achieve the desired deployment parameters.

A recent work [35] has empirically investigated the deployment of text creation tasks in Amazon Mechanical Turk (AMT). The authors validated the effectiveness of different strategies for different types of tasks such as text summarization and text translation, and provided empirical evidence for the need to guide requesters in choosing the right strategy. In this work, we propose *to automate this strategy recommendation process*. This is particularly challenging because the estimation of the cost, quality and latency of a strategy must account for *worker availability* on the

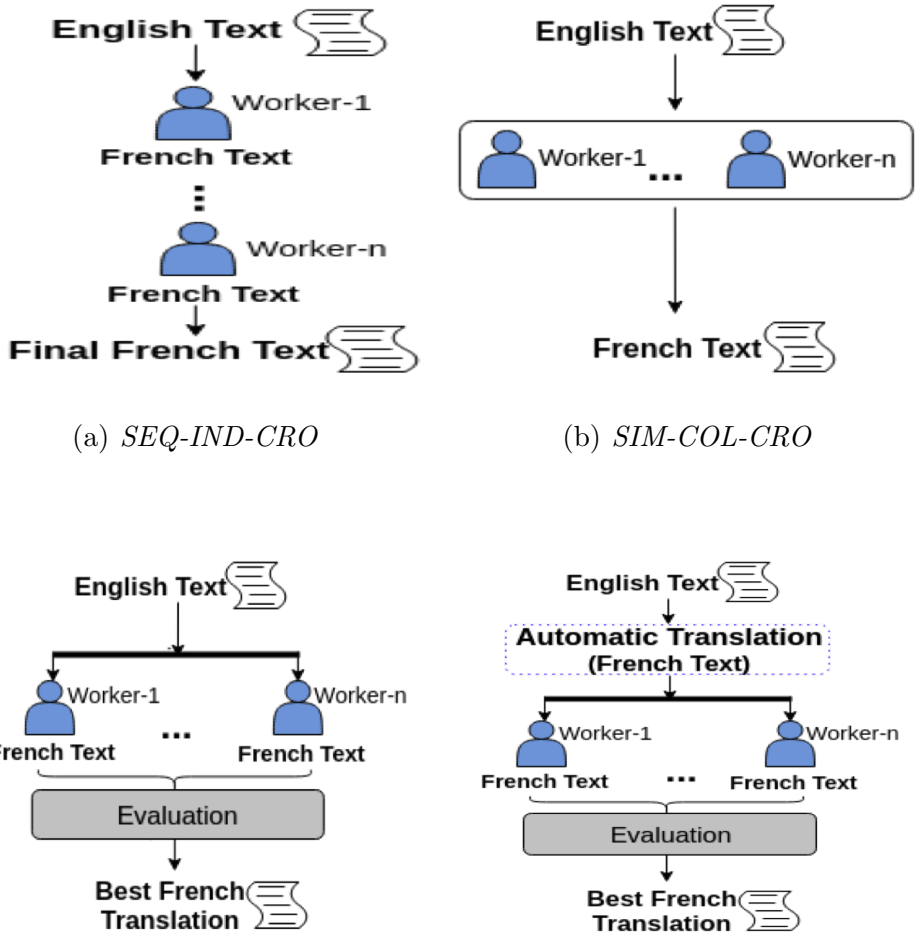
platform. Our contributions are: *we express deployment strategies as a function of worker availability, we formalize optimization problems to enable recommendation to a batch of requests, present principled algorithms, and validate them experimentally.*

We propose BatchStrat that consumes a batch of deployment requests, coming from different requesters and recommends strategies to them to optimize different goals. BatchStrat studies these incoming requests to obtain best deployment strategies given worker availability. If the platform does not have enough workers for all requests, it triages them by optimizing platform-centric goals, i.e., to maximize throughput or pay-off.

3.2 Data Model and Problem

Deployment Strategies: A deployment strategy [81] instantiates three dimensions: *Structure* (sequential or simultaneous), *Organization* (collaborative or independent), and *Style* (crowd-only or crowd and algorithms). We rely on common deployment strategies [35, 81] and refer to them as $\mathcal{S}$. Figure 3.1 enlists some strategies that are suitable for text translation tasks (from English to French in this example). For instance, *SEQ-IND-CRO* in Figure 3.1(a) dictates that workers complete tasks sequentially (*SEQ*), independently (*IND*) and with no help from algorithms (*CRO*). In *SIM-COL-CRO* (Figure 3.1(b)), workers are solicited in parallel (*SIM*) to complete a task collaboratively (*COL*) and with no help from algorithms (*CRO*). The last strategy *SIM-IND-HYB* dictates a hybrid work style (*HYB*) where workers are combined with algorithms, for instance with Google Translate. We assume that for a given platform, the set of strategies is given and bounded by $|\mathcal{S}|$.

Deployment Parameters: A deployment request d has three parameters, $d.cost$, $d.quality$, and $d.latency$. Using Example 2, the minimum quality of request is 40%, the latency 2 days, and the budget \$100. These thresholds are referred to as deployment parameters. The goal is find one or more strategies (notationally k, a



(a) *SEQ-IND-CRO* (b) *SIM-COL-CRO*

(c) *SIM-IND-CRO* (d) *SIM-IND-HYB*

Figure 3.1 Deployment Strategy Examples.

small integer) for each d, such that, for each strategy s, when d is deployed using s, it satisfies the deployment parameters.

Worker Availability: Worker availability/ available workforce W is a value in $[0, 1]$ which represents the proportion of workers who are available to undertake a task deployed by a requester within the specified time $d.latency$.

Modeling Strategy Parameters: The deployment recommendation process must be capable to estimate the parameters of a strategy s (cost, quality, latency if s is used to deploy d) to check if it is suitable for a deployment d. Since the deployed tasks are to be done by workers who are available and qualified to undertake the

Table 3.1 Deployment Requests and Strategies

	Quality	Cost	Latency
d_1	0.4	0.17	0.28
d_2	0.8	0.2	0.28
d_3	0.7	0.83	0.28
s_1	0.5	0.25	0.28
s_2	0.75	0.33	0.28
s_3	0.8	0.5	0.14
s_4	0.88	0.58	0.14

deployed tasks, the estimated strategy parameters, i.e., (estimated) quality, cost and latency of a strategy is a function of worker availability and task type.

Example 2. *Assume there are* 3 *($m = 3$) task deployment requests for different types of text editing tasks. Requester-1 d_1 is interested in deploying sentence translation tasks for* 2 *days (out of* 7 *days), at a cost up to \$*100 *(out of* \$600 *max), and expects the quality of the translation to reach at least* 40% *of domain expert quality. Table 3.1 presents all* 3 *deployment requests after normalization between* $[0-1]$. *For the purpose of this example, we assume worker availability W to be* 0.8 *for the next* 7 *days and set $k = 3$.*

For the purpose of illustration, $\mathcal{S}$ consists of the set of 4 *deployment strategies, as shown in Figure 3.1:* SIM-COL-CRO, SEQ-IND-CRO, SIM-IND-CRO, SIM-IND-HYB. *To ease understanding, we name them as s_1, s_2, s_3, s_4, respectively. s_1 costs* \$150 *and takes* 2 *days (out of* 7 *days) and ensures at least a* 50% *quality. s_2, s_3, s_4 have corresponding parameters. Strategy parameters are normalized and presented in the last column of Table 3.1.*

Problem 1. Batch Deployment Recommendation: *Given an optimization goal F, a set $\mathcal{S}$ of strategies, a batch of m deployment requests from different requesters, where the i-th task deployment d_i is associated with parameters $d_i.quality$, $d_i.cost$ and $d_i.latency$, and worker availability W, distribute W among these requests by recommending k strategies for each request such that F is optimized as follows:*

$$\begin{aligned} &\textit{Maximize } F = \sum f_i \\ &s.t. \sum \vec{w}_i \leq W \ \textit{AND} \\ &d_i \textit{ is successful} \end{aligned} \tag{3.1}$$

f_i is the optimization value of deployment d_i and $\vec{w}_i$ is the workforce required to successfully recommend k strategies it. A deployment request d_i is successful, if for each of the k strategies in the recommended set of strategies S_d^i, the following three criteria are met: $s.cost \leq d_i.cost$, $s.latency \leq d_i.latency$ and $s.quality \geq d_i.quality$.

Using Example 2, d_3 is successful for $k = 3$, as it will return $S_d^3 = \{s_2, s_3, s_4\}$, such that $d_3.cost \geq s^4.cost \geq s^3.cost \geq s^2.cost$ and $d_3.latency \geq s^4.latency \geq s^3.latency \geq s^2.latency$ and $d_3.quality \leq s^4.quality \leq s^3.quality \leq s^2.quality$, because it could be deployed with the available workforce $W = 0.8$.

In this work, F is designed to maximize one of two different platform centric-goals: task throughput and pay-off.

- Throughput: *It maximizes the total number of successful strategy recommendations without exceeding W. Formally speaking,*

$$
\begin{aligned}
&Maximize \sum_{i=1}^{m} x_i \\
&s.t. \sum x_i \times \vec{w_i} \leq W \\
&x_i = \begin{cases} 1 & d_i.cost \leq s^j.cost \ AND \\ & d_i.latency \leq s^j.latency \ AND \\ & d_i.quality \geq s^j.quality \ AND \\ & |S_d^i| = k, \forall i = 1..m, j = 1, \ldots, |\mathcal{S}| \\ 0 & otherwise \end{cases}
\end{aligned} \tag{3.2}
$$

- Pay-off: *It maximizes $d_i.cost$, if d_i is a successful deployment request without exceeding W. The rest of the formulation is akin to Equation 3.2.*

3.3 Batch Deployment Recommendation

Before getting into the details, we provide an abstraction which serves the purpose of designing BatchStrat, our proposed solution: given m deployment requests and W workforce availability, Problem 1 could be modeled in the form of a two dimensional matrix $\mathcal{W}$, where there are $|\mathcal{S}|$ columns that map to available deployment strategies and m rows of different deployment requests.

A particular cell $w_{ij} \in \mathcal{W}$ corresponds to how much workforce is needed to deploy request i with strategy j. The challenge, however, is that each d has three different requirements of quality, cost, and latency. Therefore, it has to estimate workforce requirement per (deployment, strategy) pair first. That constitutes step-1 of BatchStrat. Once for every (i, j), a (deployment, strategy) workforce requirement $w_{i,j}$ is computed, step-2 estimates the aggregated workforce requirement per deployment, since it has to recommend k different strategies to each deployment. To do that, it aggregates over each deployment request and produces a vector $\vec{W}$ of length m that estimates the workforce requirement for each strategy to be deployed successfully, meeting the quality, cost, latency thresholds as well as k. Finally, step-3 invokes an optimizer that determines how to allocate the available workforce W among

competing deployment requests to optimize different platform-centric goals. The pseudo-code of BatchStrat is presented in Algorithm 3. We now present the details.

3.3.1 Computing Workforce Requirement per Deployment

Given $\mathcal{W}$, for each deployment request d_i and strategy s^j, the workforce requirement is the maximum of workforce requirement to satisfy the quality, cost, and latency threshold. However, to find k strategies for a deployment d_i, we turn our attention back to matrix $\mathcal{W}$ again and investigate how to compute workforce requirement for all k strategies for d_i.

The objective here is to produce a vector $\vec{W}$ of length m, where the i-th value represents the aggregated workforce requirement for request d_i. To understand how to compute $\vec{W}$, we need to consider the **sum-case:** where the task designer intends to perform the deployment using all k strategies, in which case, the minimum workforce (w_i) needed to satisfy cardinality constraint k_i is $\Sigma_{y=1}^{k} w_{iy}$ (where w_{iy} is the y-th smallest workforce value in row i of matrix $\mathcal{W}$; and the **max-case:** where the designer intends to only deploy one of the k strategies, in which case, $w_i = w_{iy}$, (where w_{iy} is the k-th smallest workforce value in row i of matrix $\mathcal{W}$).

Running Time: The running time of computing the aggregated workforce requirement of the i-th deployment request is $|\mathcal{S}|\ k\ log|\mathcal{S}|$, if we use min-heaps to retrieve the k smallest numbers. The overall running time is again $O(m \times k\ log|\mathcal{S}|)$.

3.3.2 Optimization-Guided Batch Deployment

Since W is limited, it may not be possible to successfully satisfy all deployment requests in a single batch. This requires distributing W judiciously among competing deployment requests and satisfying the ones that maximize platform-centric optimization goals, i.e., throughput or pay-off.

At this point, a keen reader may notice that the batch deployment problem bears resemblance to a well-known discrete optimization problem that falls into the general category of assignment problems, specifically, Knapsack-type of problems [70]. The objective is to maximize a goal (in this case, throughput or pay-off), subject to the capacity constraint of worker availability W. In fact, depending on the nature of the problem, the optimization-guided batch deployment problem could become intractable.

Intuitively, when the objective is only to maximize throughput (i.e., the number of satisfied deployment requests), the problem is polynomial-time solvable. However, when there is an additional dimension, such as pay-off, the problem becomes NP-hard problem, as we shall prove next.

Theorem 9. *The decision version of the Pay-Off maximization problem is NP-Complete.*

Proof. (sketch): An instance of the famous 0/1 Knapsack problem could be reduced to the decision version of the Pay-off Maximization problem. □

Our solution bears similarity to the greedy algorithm of the Knapsack problem [78]. The objective is to sort the deployment strategies in non-increasing order of $\frac{f_i}{\vec{w}_i}$. The algorithm greedily adds deployments based on this sorted order until it hits a deployment d_i that can no longer be satisfied by W, that is, $\Sigma_{x=1..i} d_i > W$. At that step, it chooses the best of $\{d_1, d_2, d_{i-1}\}$ and d_i and the process continues until no further deployment requests could be satisfied for W (lines 4-8 in Algorithm BatchStrat).

Running Time: This step is dominated by the sorting time of the deployment requests, which is $O(m\ log\ m)$.

Maximizing Throughput When task throughput is maximized, the objective function F is computed simply by counting the number of deployment requests that

Algorithm 3 Algorithm BatchStrat

1: **Input:** m deployment requests, $\mathcal{S}$, objective function F, available workforce W

2: **Output:** recommendations for a subset of deployment requests.

3: Compute Workforce Requirement Matrix $\mathcal{W}$

4: Compute Workforce Requirement per Deployment Vector $\vec{W}$

5: Compute the objective function value f_i of each deployment request d_i

6: Sort the deployment strategies in non-increasing order of $\frac{f_i}{\vec{w_i}}$

7: Greedily add deployments until we hit d_i, such that $\Sigma_{x=1..i} d_i > W$

8: Pick the better of $\{d_1, d_2, d_{i-1}\}$ and d_i

are satisfied. Therefore, f_i, the objective function value of deployment d_i is the same for all the deployment requests and is 1. Our solution, BatchStrat-ThroughPut, sorts the deployment requests in increasing order of workforce requirement $\vec{w_i}$ to make $\frac{1}{\vec{w_i}}$ non-increasing. Other than that, the rest of the algorithm remains unchanged.

Theorem 10. *Algorithm BatchStrat-ThroughPut gives an exact solution to the problem.*

Maximizing Pay-Off Unlike throughput, when pay-off is maximized, there is an additional dimension involved that is different potentially for each deployment request. f_i for deployment request d_i is computed using $d_i.cost$, the amount of payment deployment d_i is willing to expend. Other than that, the rest of the algorithm remains unchanged.

Theorem 11. *Algorithm BatchStrat-PayOff has a 1/2-approximation factor.*

Proof. (sketch): The proof directly follows from [91]. □

3.4 Experiments

3.4.1 Batch Deployment Recommendation

We compare differnet algorithms. All algorithms are implemented in Python 3.6 on Ubuntu 18.10. Intel Core i9 3.6 GHz CPU, 16GB of memory.

`Brute Force:` An exhaustive algorithm which compares all possible combinations of deployment requests and returns the one that optimizes the objective function.

`BaselineG:` This algorithm sorts the deployment requests in decreasing order of $\frac{f_i}{\vec{w_i}}$ and greedily selects requests until worker availability W is exhausted.

BatchStrat: Our proposed solution described in Section 3.3.

Observation 1: Our solution BatchStrat returns exact answers for throughput optimization, and the approximation factor for pay-off maximization is always above 90%, significantly surpassing its theoretical approximation factor of 1/2.

Observation 2: Our solution BatchStrat is highly scalable and takes less than a second to handle millions of strategies, and hundreds of deployment requests, and k.

3.4.2 Quality

Goal: We aim to validate*dow does* BatchStrat *fare to optimize different platform-centric goals?*

Strategy Generation: The dimension values of a strategy are generated considering uniform and normal distributions. For the normal distribution, the mean and standard deviation are set to 0.75 and 0.1, respectively. We randomly pick the value from 0.5 to 1 for the uniform distribution.

Worker Availability: For a strategy, we assume there is a linear relationship between parameters and Worker Availability. We generate the *slope* uniformly from an interval $[0.5, 1]$. Then, we set $intercept = 1 - slope$ to make sure that the estimated

worker availability W is within $[0, 1]$. These numbers are generated in consistence with our real data experiments.

Deployment Parameters: Once W is estimated, the quality, latency, and cost - i.e., the deployment parameters, are generated in the interval $[0.625, 1]$. For each experiment, 10 deployment parameters are generated, and an average of 10 runs is presented in the results.

Figure 3.2 shows the results of throughput of BatchStrat by varying k compared with the two baselines (the same could be observed when varying m and $|\mathcal{S}|$). Figure 3.3 shows the approximation factor of BatchStrat and `BaselineG`. BatchStrat achieves an approximation factor of 0.9 most of the time. For both experiments, the default values are $k = 10, m = 5, |\mathcal{S}| = 30, W = 0.5$ because brute force does not scale beyond that.

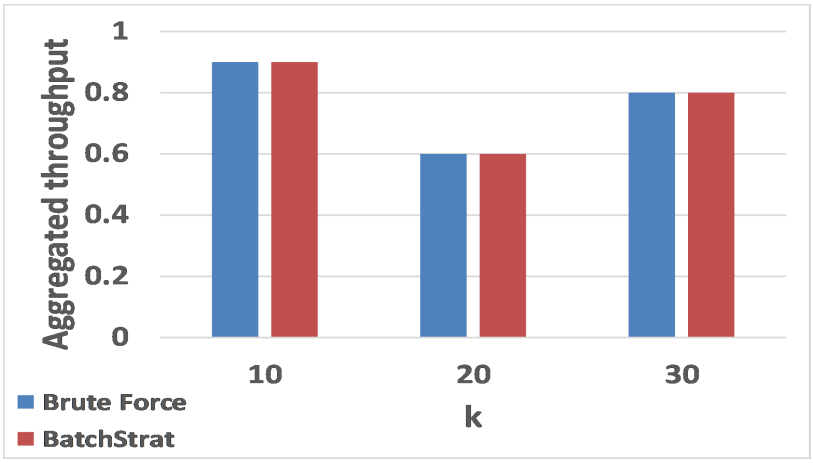


Figure 3.2 Objective Function for Throughput when Varying k.

3.4.3 Scalability

Since the `BaselineG` has the same running time as BatchStrat, we only compare `Brute Force` and BatchStrat. The default setting for $|\mathcal{S}|$, k and W are 30, 10 and 0.75, respectively.

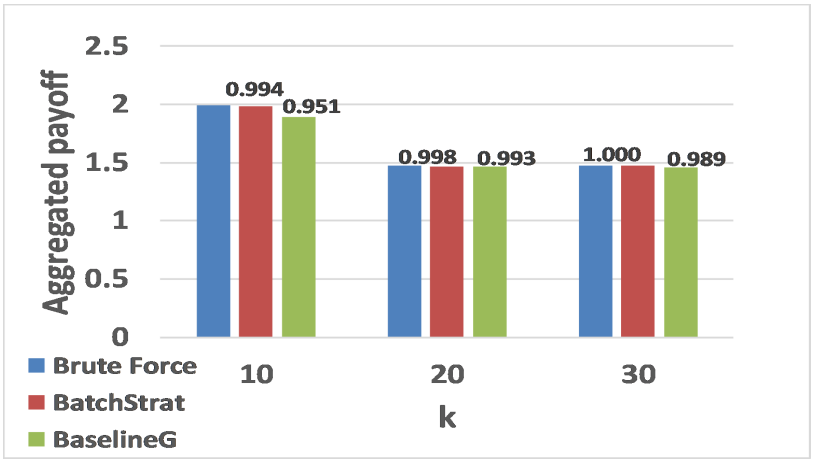


Figure 3.3 Objective Function and Approximation Factor for Payoff when Varying k.

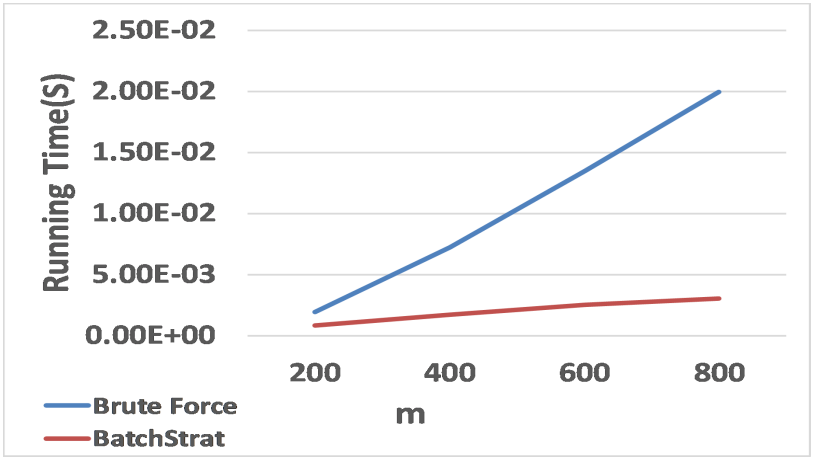


Figure 3.4 Running time for Batch Deployment Varying m.

Figure 3.4 shows that `Brute Force` takes exponential time with increasing m, whereas BatchStrat scales linearly. Clearly BatchStrat can handle millions of strategies, several hundreds of batches, and very large k and still takes only a few fractions of seconds to run. It is easy to notice that the running time of this problem only relies on the size of the batch m (or the number of deployment requests), and not on k or $\mathcal{S}$.

3.5 Future work

Our preliminary work opens up more than one research directions. First and foremost, how to estimate worker availability for different types of tasks is a challenging problem that requires deep investigation in its own merit. Then, an interesting open problem is to come up with principled yet practical models to establish relationship between deployment parameters and strategy parameters. Throughout this chapter, we have assumed that the estimated quality, cost, and latency of a set of tasks deployed using a strategy is a function of the task type and worker availability. However, how to realistically model such functions or learn them from historical data for different types of tasks remains to be a part of our ongoing investigations. Finally, an interesting extension is to explore the recommendation of alternative deployment parameters if a request cannot be satisfied as formulated. This would open the possibility of recommending different deployment parameters for which k strategies are available, thereby guiding requesters further in task deployment.

CHAPTER 4

PEER LEARNING THROUGH TARGETED DYNAMIC GROUPS FORMATION

4.1 Introduction

Online social networks and learning platforms enable the formation of targeted groups for peer learning. As an example, peer learning associations[1], social Q&A sites[2], even crowdsourcing platforms[3] investigate how interaction between like-minded individuals can improve knowledge and understanding on a topic, or simply promote improved well-being of the individuals. Indeed, systematic targeted groups formation can leverage the presence of knowledgeable individuals in order to educate group participants on a myriad of topics, and support efforts to dispel rumors and misinformation.

4.1.1 Novelty

The importance of targeted groups formation has been recognized in the literature. Recent works have studied the effect of *targeted one-shot* groups formation to optimize peer learning [3, 61], where the objective is to form a set of groups to maximize some type of aggregate learning. These works view groups as *static*, in the sense that every individual is assumed to be a member of only one group through the end of the process. What is not studied *is the effect of time*, and more concretely the potential of allowing the formation of a targeted set of groups to be repeated a certain number of times, as opposed to a single shot process. We hypothesize that *dynamic* group formation will enable more individuals to 'learn from the best', can better utilize intermediate learning gains, and has the potential to improve overall peer learning

[1]https://peerlearningassociation.weebly.com Retrieved on May/01/2020
[2]https://www.quora.com Retrieved on May/01/2020
[3]https://appen.com Retrieved on May/01/2020

outcomes, both in theory and practice. Ours is the **first** systematic effort to model this problem and study algorithms for it.

4.1.2 Practical Motivation

Imagine a peer learning scenario in a physical classroom or on an online learning platform. Separating the participants into equal-size groups for homework assignments or projects is common practice to ensure relatively similar workload among students, while enabling effective interaction among the peers [117]. However, in the case when there are multiple group homework assignments or projects, *fixed* groups may be not optimal. Research has shown that successful groups evolve naturally [20, 135], and that the ability of dynamically altering group composition results in groups that persist for longer [77, 112]. This suggests that dynamic groups may offer benefits. Going back to the classroom example, it would be intuitively better to change the group membership of the students across the assignments so that everyone gets the opportunity to learn from the best participants, with the hope that the total 'educational welfare' is maximized.

4.1.3 Technical Contributions

We undertake the first formal attempt to study dynamic group formation to optimize peer learning. We follow previous related works [3, 5, 23, 139] and adopt common definitions and assumptions used to quantify the single-shot group formation. We then introduce a vast generalization and study *the effect of time and how the flexibility of changing membership and learning from others can improve the outcome.* Specifically, we make the following contributions:

(i) We initiate the study of the targeted dynamic groups formation (referred to as Targeted Dynamic Grouping or TDG) problem. We assume that the process consists of a predefined number of rounds (α). Each round entails a grouping of the participants into groups of size (k), and a defined learning gain for each individual, controlled by a *linear* learning gain function f. We consider two different interaction modes of learning within each group. One induces a *star*, where the interaction of any

group member is limited only to the most knowledgeable individual of that group. The other induces a *clique*-like structure, where all possible pairs of within-group interactions take place. Our goal is to find dynamic groupings that *maximize the aggregate learning gain* after α rounds.

(ii) We delve into an investigation of the nuances of our proposed problem. We present an algorithmic framework DyGroups that is greedy in nature, and highly scalable. DyGroups runs in $\Theta(\alpha n)$ time for both *clique* and *star* interaction modes, where n is the number of members. The greedy approach comes with an *interesting twist*. In each round, there are multiple k-groupings that maximize the aggregate learning for that round. Among them, DyGroups selects the one that also maximizes the variance of the participants' skills after the round. We also present an in-depth proof of the optimality of the proposed algorithm in a special case. We prove that DyGroups can always find the overall optimal solution for the *star* interaction structure when $k = 2$. However, extending the optimality proof to more general cases ($k > 2$) appears to be a far more difficult and interesting problem.

(iii) We run two independent rigorous experiments on real fact-learning in peer groups comparing multiple baseline algorithms. Specifically, we recruited about 200 human subjects from Amazon Mechanical Turk and asked them to learn facts about COVID-19 through peer interaction. The experimental results corroborate two crucial hypotheses with statistical significance that are central to our formulation and proposed solutions. First, it validates that *workers' skills improve through peer interaction*. Second, it experimentally demonstrates that *changing group composition over time is important and that* DyGroups *outperforms baseline solutions*. Additionally, we run large scale synthetic data experiments and implement several baselines (along with recent related works [5, 61]) to validate the theoretical claims and scalability aspects of our proposed solutions.

This work is organized as follows. Our model and problem formalization are discussed in Section 4.2. Our algorithmic framework and its running time properties are discussed in Section 4.3. Section 4.4 contains theoretical proofs of our statements. Experiments using human subjects as well as large scale simulation studies are shown in Section 4.5. The related work is introduced in Section 4.6. We present a discussion and outline future works in Section 4.7, and conclude in Section 4.8.

4.2 Model and Definitions

We consider n participants who undergo a learning process in rounds, or time steps. Before step t each participant is associated with a positive real number, quantifying their *skill level*. In every round, k non-overlapping equi-sized groups are formed.

Each participant interacts 1-on-1 only with members of their group. The learning outcome of a 2-person interaction is determined by the *learning gain function*. The total learning gain within a group is further determined by a specified *interaction mode*. After one round, skill levels are updated. The process continues inductively for α steps.

We now proceed to define the highlighted terms in the above description. We will be using the following example to illustrate the notions.

[Toy Example]. Imagine a small number of $n = 9$ students taking a course on Python Programming that comprises of $\alpha = 4$ assignments during the course of a semester. In every round we will be forming $k = 3$ disjoint groups of size 3. We assume that in the beginning of the course, the skills of the students in Python programming are as follows: $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$ [4].

Learning Gain Function for 2-Person Interactions. The learning outcome for a 2-person interaction between participants i, j depend only on $\Delta = |s_i - s_j|$. Specifically, if $s_i > s_j$, then after their interaction:

(i) s_i is unaltered, and **(ii)** s_j is updated to $s_j + f(\Delta)$

We call $f(\Delta)$ the learning gain function. In the rest of this work we will work with linear functions $f(\Delta) = r\Delta$, where $r \in (0,1)$ is a *learning rate* parameter that is part of the input [5].

As the learning function involves an asymmetry between the two parts, we let $f(i \to j)$ denote the skill gain of person j from their interaction with person i. Note that if the skill of i is lower than that of j, then $f(i \to j) = 0$.

[4]The distribution of the initial skill values can be arbitrary. How to estimate numerical values for the initial skills of the individuals in a real situation is an orthogonal issue. In our experimental evaluation, we demonstrate a realistic way of estimating their initial skills.
[5]The case r=1 is relatively straightforward and we omit it.

In the TOY EXAMPLE, with a learning rate or $r = 0.5$, a pairwise interaction between the 3-rd and the 9-th member with skills $0.3, 0.9$ respectively, s_9 remains unaltered at 0.9. On the other hand, s_3 becomes $0.3 + 0.5 \times (0.9 - 0.3) = 0.6$.

Interaction Modes, Group Learning Gain. In each round, participants are split into non-overlapping equi-sized groups. The size of the groups is fixed throughout the α rounds. Each participant has 2-person interactions with other participants from their group. The learning gain within the groups will be determined by the *interaction mode*, which will be the same for all groups, throughout the process. We consider two possible interactions modes:

(i) *Star Mode.* Every participant of the group learns from the highest-skilled member of the group, and skill levels are updated according to the learning gain function. Specifically if p_i is i^{th} highest-skill participant of a group x, the learning gain of group x is

$$g_{star}(x) = \sum_{p_j \neq p_1} f(p_1 \rightarrow p_j). \tag{4.1}$$

In the TOY EXAMPLE, assume $[0.9, 0.5, 0.3]$ are assigned to a group and the interaction model is star model. In this case 0.9 is unaltered in the group, and $0.5, 0.3$ will both learn from 0.9 and are updated to $0.7, 0.6$ after the learning, assuming a learning rate $r = 0.5$. In this case the total learning gain of the group (soon to be defined formally) is 0.5.

(ii) *Clique Mode.* All possible pairwise interactions take place. Suppose that p_i is the participant with the i^{th} highest learning skill in a group x. That implies that p_i will learn and gain skill from $(i-1)$ persons. Then, we define the learning gain of a group x as follows:

$$g_{cliq}(x) = \sum_{p_i \in x} \frac{1}{i-1} \left(\sum_{p_j \neq p_i} f(p_j \rightarrow p_i) \right) \tag{4.2}$$

In plain words, the total gain for p_i is the average of its positive gains from 2-person interactions within its group. The averaging operation ensures that the order of skill levels is preserved within the group after the round, as it would be expected in practice.

In the TOY EXAMPLE, assume $[0.9, 0.5, 0.3]$ form a group. As before, 0.9 is unaltered. However, 0.3 learns not only from 0.9 but also from 0.5. Therefore, the new skill value for 0.3 after learning is $0.3 + (0.5(0.5 - 0.3) + 0.5(0.9 - 0.3))/2 = 0.5$. Since 0.5 only learns from 0.9, the new skill value of 0.5 is 0.7 as the same in the previous example. The overall group learning gain is 0.4.

Aggregated Learning Gain per Round. Given a grouping $\mathcal{G}_t$ of k groups at round t, the aggregated learning gain of the grouping under either mode is defined as:

$$\mathbf{LG}(\mathcal{G}_t) = \sum_{x=1}^{k} g(x). \tag{4.3}$$

Problem 2. Targeted Dynamic Grouping *(*TDG*): Given as input a set of n individuals and their skills, an integer k representing the number of groups, and an integer α representing the number of rounds, our goal is to compute a sequence of groupings $\mathcal{G}_1 \ldots, \mathcal{G}_\alpha$ that maximizes the aggregated learning gain over α rounds:*

$$\max_{\{\mathcal{G}_1 \ldots, \mathcal{G}_\alpha\}} \sum_{t=1}^{\alpha} \mathbf{LG}(\mathcal{G}_t)$$

4.3 Algorithms and Running Time

We start with the presentation of the generic algorithmic framework DYGROUPS. Then we instantiate it for the Star and Clique modes in Sections 4.3.1 and 4.3.2 respectively. To avoid disrupting the flow of ideas, we defer the formal statements and proofs for our claims to Section 4.4.

DyGroups is fairly simple; we take a greedy stride in solving the problem. Since the process of forming k groups is to be repeated over α rounds, in each round t, DyGroups calls subroutine DyGroups-Local to form a grouping $\mathcal{G}_t$ of k groups so as to *locally* maximize $\mathbf{LG}(\mathcal{G}_t)$ at round t. The new skill values become part of the inputs for round $t+1$ and the process repeats. Algorithm 4 presents the pseudo-code.

Algorithm 4 DyGroups-Mode

1: *input*: Set S of n skills, where s_i is the skill of individual i, number of groups k, learning rate r, number of rounds α.
2: *output*: Collection of α groupings, each consisting of k equi-sized groups, where $\mathcal{G}^t$ is the grouping in round t
3: **for** $t = 1 : \alpha$ **do**
4: $\quad \mathcal{G}^t =$ DyGroups-Mode-Local(S,k)
5: $\quad S =$ Update-Skills-Mode($\mathcal{G}^t$,S)
6: **end for**
7: **return** $\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_\alpha$

Three remarks are in order:

- DyGroups-Mode is a generic framework in the sense that it can be instantiated for different interaction modes, by instantiating the two subroutines it calls. We will see how to do that specifically for the Star and Clique modes.
- The greedy approach is a reasonable choice in this framework. As we will see shortly, DyGroups-Mode-Local can solve the round-local maximization problem very efficiently in both the Star and Clique modes. It thus lends to a highly scalable algorithm, both with respect to n and k. There is additional theoretical support to our choice. As we shall show in Section 4.4.3, DyGroups-Star, does produce the optimal solution for $k = 2$.
- The **running time** of routine DyGroups-Mode is clearly $O(\alpha(T_g + T_u))$, where T_g, T_u are the running times of DyGroups-Mode-Local and Update-Skills-Mode respectively.

We also note that storing a grouping G^t requires $\Omega(n)$ memory, and so each round of the algorithm requires $\Omega(n)$ time. This leads to the following lower bound on the running time of DyGroups-Mode.

Claim 1. DyGroups-Mode *requires* $\Omega(\alpha n)$ *time.*

4.3.1 DyGroups-Star

We begin by noting that Update-Skills-Star has a very straightforward implementation. Each skill update takes $O(1)$ time, because each participant interacts only with the 'teacher' of their group. Thus the total running time is $O(n)$.

On the other had, designing DyGroups-Star-Local is a very interesting problem. Let us call the highest-skilled person in some group, the *teacher* of that group. We first observe that the learning gain is maximized if the teachers of the k groups are selected to be the k highest-skilled participants, and that is true *irrespectively* of the split of the remaining $n-k$ participants into the k groups. This is due to the linearity of the learning function. The proof of this claim appears in Section 4.4, Theorem 12.

We thus have to select from exponential number of locally optimal groupings in each round [6]. Our further insight is to select the grouping $\mathcal{G}$ that has the *maximum variance* of skill values among the locally optimal groupings. This is done as follows. Suppose $p_1, \ldots, p_k$ are the k teachers. Recall that we have to assign the other $n-k$ participants to a teacher. In order to do that we split them into k provisional groups of size $s = n/k - 1$, solely based on their skill level: each person in provisional group i has skill equal or higher relative to every person in group $i+1$. Then we form group i, by assigning the i^{th} provisional group to teacher p_i. The proof that this

[6] e.g. when $k = n/2$ there are $(n/2)!$ locally optimal solutions, and when $k = 2$ there are $\binom{n-2}{(n-2)/2}$ locally optimal solutions.

assignment maximizes variance appears in Section 4.4, Theorem 13. Algorithm 5 gives an implementation.

Algorithm 5 DyGroups-Star-Local

input: Set S of n skills, where s_i is the skill of individual i, number of groups k.
output: Grouping $\mathcal{G}$ consisting of k groups of size n/k.
X = sort(S,descending) // sorted skill values
Let p_i be the participant with skill $X[i]$
$t = k + 1; s = n/k - 1;$
for $i = 1 : k$ **do**
    Assign 'teacher' p_i to group g_i
    for j=1:s **do**
        Assign p_t to group g_i
        t = t+1
    end for
end for
return $\mathcal{G} = \{g_1, g_2, ..., g_k\}$

The **running time** of DyGroups-Star-Local is dominated by $O(n \log n)$ for the sorting step. It is easy to see that the remaining lines take $O(n)$ time. Thus, the overall running time of DyGroups-Star is $O(\alpha n \log n)$, which notably is independent of k.

We now illustrate our discussion using the Toy Example from Section 4.2. We first follow a sequence of three groupings that selects an arbitrary locally optimal grouping in each round.

Round 1: $[0.9, 0.1, 0.2], [0.8, 0.3, 0.4], [0.7, 0.5, 0.6]$.

Updated Skills: $[0.9, 0.8, 0.7, 0.65, 0.6, 0.6, 0.55, 0.55, 0.5]$.

Round 2: $[0.9, 0.55, 0.5], [0.8, 0.6, 0.55], [0.7, 0.65, 0.6]$

Updated Skills: $[0.9, 0.8, 0.7, 0.675, 0.65, 0.7, 0.675, 0.725, 0.7]$

Round 3: $[0.9, 0.675, 0.65], [0.8, 0.7, 0.675], [0.725, 0.7, 0.7]$

Final: $[0.9, 0.8, 0.725, 0.7125, 0.7125, 0.75, 0.7375, 0.7875, 0.775]$

The total learning gain after 3 rounds is 2.4.

Let us now introduce how DyGroups-Star runs.

Round 1: $[0.9, 0.6, 0.5], [0.8, 0.4, 3], [0.7, 0.2, 0.1]$

Updated Skills: $[0.9, 0.8, 0.7, 0.75, 0.7, 0.6, 0.55, 0.45, 0.4]$

Round 2: $[0.9, 0.7, 0.7], [0.8, 0.6, 0.55], [0.75, 0.45, 0.4]$

Updated Skills:$[0.9, 0.8, 0.75, 0.8, 0.8, 0.7, 0.675, 0.6, 0.575]$

Round 3: $[0.9, 0.8, 0.75], [0.8, 0.7, 0.675], [0.8, 0.6, 0.575]$

Final: $[0.9, 0.8, 0.8, 0.85, 0.825, 0.75, 0.7375, 0.70, 0.6875]$

The total learning gain after 3 rounds is 2.55.

While we do not know if DyGroups-Star will in general produce the optimal grouping sequence, we do present a proof of this fact for $k = 2$, in Section 4.4.3 . We note that forming two groups is natural in applications, such as peer programming, where one group does the programming and the other peer reviews.

Finally we would like to shortly discuss the insight that leads to the proof for $k = 2$. A further inspection of the above example can reveal that the two different sequences of groupings produce the same learning gain after the first 2 rounds. However the variance maximization policy leads to a *higher* 3^{rd}*-order teacher* in round 3. This availability of better teachers earlier in the process is what leads to DyGroups-Star dominating other solutions.

4.3.2 DyGroups-Clique

Let us begin with Update-Skills-Clique. By design, in the Clique mode, every person learns from all the higher-skilled persons in the group and so there are $O(t^2)$ interactions, where $t = n/k$ is the size of the group. However it is possible to calculate

all the updated skills within the group in $O(t)$ time, leading to an $O(n)$ update algorithm. We prove this fact in Section 4.4, Theorem 14.

Similarly to DyGroups-Star, DyGroups-Clique finds a grouping that maximizes the gain for each round. This is formally stated in Theorem 14. The Clique mode definition leads us to a different grouping algorithm, which computes the *unique* grouping $\mathcal{G} = \{g_1, \ldots, g_k\}$ with the property that the j^{th}-ordered skill in g_i is greater than or equal to the j^{th}-ordered skill in g_{i+1} for each i, j. Algorithm 6 provides an implementation.

Algorithm 6 DyGroups-Clique-Local

input: Set S of n skills, where s_i is the skill of individual i, number of groups k.
output: Grouping $\mathcal{G}$ consisting of k groups of size n/k.
X = sort(S,descending) // sorted skill values
Let p_i be the participant with skill $X[i]$
$t = 1; s = n/k;$
for $j = 1 : s$ **do**
    for $i = 1 : k$ **do**
        Assign p_t to group g_i
        t = t+1
    end for
end for
return $\mathcal{G} = \{g_1, g_2, ..., g_k\}$

The **running time** of DyGroups-Star-Clique is dominated by $O(n \log n)$ for the sorting step. It is easy to see that the remaining lines take $O(n)$ time. Thus, the overall running time of DyGroups-Clique is $O(\alpha n \log n)$, which notably is independent of k.

Let us illustrate the algorithm with Toy Example.

Round 1: $[0.9, 0.6, 0.3], [0.8, 0.5, 0.2], [0.7, 0.4, 0.1]$

Updated: $[0.9, 0.8, 0.75, 0.7, 0.65, 0.55, 0.525, 0.425, 0.325]$

Round 2: $[0.9, 0.7, 0.525], [0.8, 0.65, 0.425], [0.75, 0.55, 0.325]$

Updated: $[0.9, 0.8, 0.8, 0.75, 0.725, 0.6625, 0.65, 0.575, 0.4875]$

Round 3: $[0.9, 0.75, 0.65], [0.8, 0.725, 0.575], [0.8, 0.6625, 0.4875]$

Final: $[0.9, 0.825, 0.8, 0.8, 0.7625, 0.7375, 0.73125, 0.66875, 0.609375]$ The total learning gain after 3 rounds is 2.334375.

4.4 Proofs

All formal statements in this Section refer back to the informal discussion in Section 4.3.

4.4.1 DyGroups-Star

Theorem 12. *Suppose $S = [s_1, \ldots, s_n]$ is a set of skills in descending order. Then: (a) An optimal grouping into k equi-sized groups, i.e. a grouping that maximizes the learning gain, must assign $s_1, \ldots, s_k$ to different groups. (b) Every grouping that assigns $s_1, \ldots, s_k$ is optimal.*

Proof. Given a group g let $s_{g,i}$ denote the i^{th} highest skill in g. Also, let $t = n/k$ denote the size of the groups. (a) For the sake of contradiction, suppose that $\mathcal{G}$ is optimal and that it does not assign $s_1, \ldots, s_k$ to the same group. That implies that there are two groups $g, g' \in \mathcal{G}$ such that $s_{g,2} > s_{g',1}$. Let C_g be the cumulative value of the $t-2$ smallest skills in g, and let C'_g be the cumulative value of the $t-1$ smallest skills in g'. The total gain A in the given grouping is given by:

$$r\left[(s_{g,1} - s_{g,2}) + ((t-2)s_{g,1} - C_g) + ((t-1)s_{g',1} - C_{g'})\right]$$

We can now swap the two skills g_2, g'_1 between g and g'. In this new grouping, the new total gain B is

$$r\left[(s_{g,1} - s_{g',1}) + ((t-2)s_{g,1} - C_g) + ((t-1)s_{g,2} - C_{g'})\right]$$

The statement is meaningful when $t > 2$. Given that $s_{g,2} > s_{g',1}$, $t > 1$, we get that $B > A$, which is a contradiction.

(b) Suppose now that $\mathcal{G}$ is an arbitrary optimal grouping. Let g, g' be two arbitrary groups in G, and let $s_{g,i}, s_{g',j}$ two skills from g, g' different than the top skills (i.e. $i, j \neq 1$). Swapping $s_{g,i}$ and $s_{g',j}$ between the two groups leaves the total gain invariant, as follows from a simple application of the definition. Now if $\mathcal{G}$ and $\mathcal{G}'$ are two different arbitrary optimal groupings, then they must agree on the leaders of the k groups, by part (a). One can then apply a sequence of swaps such as the one described above, to transform $\mathcal{G}$ into $\mathcal{G}'$, while leaving the gain invariant along the sequence. Therefore the total gain of $\mathcal{G}$ and $\mathcal{G}'$ must be the same. □

Theorem 13. *The output $\mathcal{G}$ of* DyGroups-Star-Local *is the grouping of highest variance among the groupings that maximize the learning gain on input set S.*

Sketch. For the sake of contradiction suppose $\mathcal{G}'$ is another optimal grouping. Let $\mathcal{G}' = \{g'_1, \ldots, g'_k\}$. By Theorem 12, since $\mathcal{G}'$ is optimal, the highest skills $s_1 \geq \ldots \geq s_k$ of its groups are fixed, and they are the same in the corresponding groups of $\mathcal{G}$. Since $\mathcal{G} \neq \mathcal{G}'$, there must be some $i > j$ such that g'_i contains skill s', and g'_j contains skill s, with $s < s'$.

Now let $\mathcal{G}''$ be the grouping after swapping s' and s in $\mathcal{G}'$. By Theorem 12, the mean skill μ is the same in $\mathcal{G}'$ and $\mathcal{G}''$ after the update. The variance after the update in $\mathcal{G}'$ is:

$$A = C + (s + r(s_i - s)) - \mu)^2 + (s' + r(s_j - s')) - \mu)^2$$

where C is a sum of squares, each coming from one of the other $n-2$ updated skill values. Similarly the variance after the update in $\mathcal{G}''$ is:

$$B = C + (s' + r(s_i - s')) - \mu)^2 + (s + r(s_j - s)) - \mu)^2$$

We can now calculate $B - A$, using the facts that $s_i > s_j$ and $s < s'$, and get $B > A$. This is a contradiction. □

4.4.2 DyGroups-Clique

Theorem 14. UPDATE-SKILLS-CLIQUE *can be implemented in* $O(n)$ *time.*

Proof. Let $t = n/k$ be the size of one group. We fix an arbitrary group, and assume that the initial set of descending-sorted skills is $s_1, \ldots, s_t$. Now let $c_i = \sum_{j=1}^{i} s_j$. Clearly each c_{i+1} can be computed from c_i with one addition. Hence computing all c_i's takes $O(t)$ time. Let $s'_1, \ldots, s'_t$ be the updated set of skills. Clearly, we have $s'_1 = s_1$ since the highest-skilled person does not learn from anyone in the group. Also, for $i > 1$, using our definitions from Section 4.2 we have

$$s'_{i+1} = s_i + r(c_i - is_{i+1})/i$$

Thus, having computed the c_i's, the calculation of all new skills requires $O(t)$ time. Doing that for k groups gives a total running time of $O(n)$. □

Theorem 15. DYGROUPS-LOCAL-CLIQUE *produces a grouping that maximizes the gain for the input set of skills.*

The proof follows a similar reasoning with that of the proof of Theorem 12. However the calculations are rather lengthy, and so we omit it due to space constraints.

4.4.3 DyGroups-Star for Two Groups

Terminology: when we say a grouping is *locally-optimal* we mean that maximizes the gain for that round only, but possibly not for the entire process. If it is not locally optimal, then we say it is *non-locally optimal*.

Theorem 16. DyGroups-Star *is optimal for the TDG problem when* $k = 2$.

The proof is rather non-trivial and is presented in several steps. We first present an equivalent yet alternative objective function of the TDG problem that is pivotal for the proof. We then present a set of helper lemmas that are crucial and finally prove Theorem 17.

An equivalent objective function: Let $s_1^0, ..., s_n^0$ be the input skill values in decreasing order. The TDG objective is:

$$\text{Maximize} \sum_{t=1}^{\alpha} \mathbf{LG}(\mathcal{G}_t)$$

Assume the skill value of a member i after α rounds is s_i^α, and s_i^0 is the initial skill value of i. Then, the objective function can be written as:

$$\text{Maximize} \sum_{i=1}^{n} s_i^\alpha - s_i^0$$

We thus convert the input to the distance to the highest-skilled member: $0, b_2^0, b_3^0..., b_n^0$, where $b_i^0 = s_1^0 - s_i^0$. In the Toy Example the $s_1, .., s_9$ are $[0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]$. Therefore, $b_1, ..., b_9$ are $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$.

Therefore, the equivalent objective function of the TDG problem is

$$\text{Minimize} \sum_{i=1}^{n} b_i^\alpha \tag{4.4}$$

Assume $D = \sum_{i=1}^{n} b_i^0$. Since the most skilled member will always lead a group, we can assume b_x^0 is the teacher in the other group. In each group, there are $\frac{n}{2} - 1$ members learning from the leader. Assume i is one of them and x is the leader in i's

group. The skill value s_i^1 after one round will be:

$a_i^0 + r(a_x^0 - a_i^0)$. So, the skill distance b_i^1 after this round will be:

$$\begin{aligned} s_1 - s_i^1 = & \\ & = s_1 - s_i^0 - r(s_x^0 - s_i^0) \\ & = b_i^0 - r(b_i^0 - b_x^0) \\ & = (1-r)b_i^0 + rb_x^0 \end{aligned}$$

Therefore, the aggregated skill distance after the first round will be:

$$\begin{aligned} & \sum_{\substack{i\in[2,\ldots,n] \\ i\neq x}} (1-r)b_i^0 + (\frac{n}{2} - 1)rb_x^0 + b_x^0 \\ = & \sum_{\substack{i\in[2,\ldots,n] \\ i\neq x}} (1-r)b_i^0 + (\frac{n}{2} - 1)rb_x^0 + (b_x^0 - rb_x^0) + rb_x^0 \\ = & \sum_{i\in[2,\ldots,n]} (1-r)b_i^0 + \frac{n}{2}rb_x^0 \\ = & (1-r)D + \frac{n}{2}rb_x^0 \end{aligned}$$

By this, we can assume b_x^i is the teacher of the second group for the round i ($i < \alpha$) and adapt the equation above recursively. According to this, we can re-write the objective function as:

$$\text{Minimize } \frac{n}{2}r\sum_{i=1}^{\alpha} b_x^i(1-r)^{\alpha-i} + D(1-r)^{\alpha} \tag{4.5}$$

Since the second part is constant, the problem becomes to find a series of groupings which maximize the skill value of the second teacher in the group. For the purpose of the proof, we will stick to this alternative objective function for the remainder of this subsection.

Lemma 3. *There exist* $2\binom{n-2}{\frac{n}{2}-1}$ *local optima in each round.*

Proof. According to the *Star* mode, the teacher of group-1 is the highest skilled individual (s_1) and the teacher of group-2 is the second highest skilled individual (with skill s_2). The assignment of the remaining individuals inside these two groups does not interfere with the learning gain function. Since there are such $2\binom{n-2}{\frac{n}{2}-1}$ possible assignments. Therefore, we have $2\binom{n-2}{\frac{n}{2}-1}$ local optimals in each round. □

Lemma 4. DyGroups-Star *in conjunction with* DyGroups-Star-Local *is not worse than any other solutions that produce local optima in each round.*

Proof. Assume X is the series of groupings that are produced by DyGroups-Star with DyGroups-Star-Local and there is another solution X' that also contains local maximum groupings but different from X. Therefore, in X', there must be at least one grouping that is not present in X. For a local maximum solution, the teacher of the second group is the second most skilled person. According to the objective function, if X' is a better solution than X, X' should have higher second skilled person at some rounds.

Let's assume that such a scenario occurs in round t.This entails individual i attains higher skill value in X' than in X and this skill value exceeds the second highest skill value in round $t-1$. It means either i achieves higher learning gain in X' than X, or i has higher skill value in X' before $t-1$.

In DyGroups-Star-Local, the 3rd to $\frac{n}{2}$ individuals are placed in the first group, which ensures the largest skill increase. Therefore, the aforementioned scenario can not happen. Thus, DyGroups-Star in conjunction with DyGroups-Star-Local will not produce any worse objective function than any other local maximum solution. □

Theorem 17. DyGroups-Star *is not worse than any other solution that contains non-local optima.*

Proof. The overall proof consists of multiple steps. First, we assume a case when the alternative solution with objective value X' contains only one non-local optima. Then, we extend the proof to the case when the alternative solution contains multiple (y) local solutions that are not *local optima*.

(Only one non-local optima:) First, let's assume there is a solution X' that contains only one *non-local optimum grouping at round* t $(t < \alpha)$. Other than that, DyGroups-Star and this other solution both produce local optima in the first $t-1$ rounds. Let $b_i^{t'}$ be the skill difference of the i-the member in the t-th round (recall Equation 4.4). Let $b_x^{t'}(x > 2)$ be the skill difference of the second teacher. So, the objective function value of X' will be:

$$D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=1}^{t-1} b_2^{i'}(1-r)^{\alpha-i} + b_x^{t'}(1-r)^{\alpha-t} + b_2^{t+1'}(1-r)^{\alpha-t-1} + \sum_{i=t+2}^{\alpha} b_2^{i'}(1-r)^{\alpha-i})$$

For our solution X, the objective value is:

$$D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=1}^{t-1} b_2^{i}(1-r)^{\alpha-i} + b_2^{t}(1-r)^{\alpha-t} + b_2^{t+1}(1-r)^{\alpha-t-1} + \sum_{i=t+2}^{\alpha} b_2^{i}(1-r)^{\alpha-i})$$

Since $b_2^{t'} \leq b_x^{t'}$, $b_2^{t'}$ is placed in the first group t. Therefore, $b_2^{t'}$ becomes to $(1-r)b_2^{t'}$ after round t. Noted that, $b_2^{t'} \leq b_i^{t'}, i \geq 3$, so $(1-r)b_2^{t-1'} \leq (1-r)b_i^{t-1'}, i \geq 3$. That means $(1-r)b_2^{t-1'}$ is the teacher of the second group$(b_2^{t+1'})$ of X' at round $t+1$.

The minimum skill difference (that is the b value where smaller is better) that the second teacher can attain is $b_2^{i'}, i < t$ of X' is b_2^{i}. So, we can rewrite the objective

value of X' as:

$$D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=0}^{t-1} b_2^i(1-r)^{\alpha-i} + b_x^t(1-r)^{\alpha-t} + b_2^t(1-r)^{\alpha-t} + \sum_{i=t+2}^{\alpha} b_2^{i'}(1-r)^{\alpha-i})$$

Then, there are two possible cases: **(Case 1:)** b_2^t stays as the second teacher in X from t to α; **(Case 2:)** A new second teacher shows up in round u ($u > t$) with $b_3^t(1-r)^{u-t}$.

(Case 1:) The objective value of X is:

$$D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=1}^{t-1} b_2^i(1-r)^{\alpha-i} + b_2^t(1-r)^{\alpha-t} + b_2^t(1-r)^{\alpha-t-1} + \sum_{i=t+2}^{\alpha} b_2^{t-1}(1-r)^{\alpha-i}) \tag{4.6}$$

The objective value for X' is:

$$D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=1}^{t-1} b_2^i(1-r)^{\alpha-i} + b_x^t(1-r)^{\alpha-t} + b_2^t(1-r)^{\alpha-t} + \sum_{i=t+2}^{\alpha} b_2^t(1-r)^{\alpha-i+1}) \tag{4.7}$$

So, Equation 4.6 - Equation 4.7 is:

$$r\frac{n}{2}(b_2^t - b_x^t(1-r)^{\alpha-t})$$

Since there is no position change in the second teacher, $b_2^t \leq b_x^t(1-r)^{\alpha-t}$. Therefore, Equation 4.6 - Equation 4.7 ≤ 0 and proved.

(Case 2:): In this case, $(1-r)^{u-t}b_3^t \leq b_2^t$. In addition, since there is no change in the second teacher before round u, $b_2^t \leq (1-r)^{u-t-1}b_i^t, i \geq 3$.

Therefore, $(1-r)b_2^t \leq (1-r)^{u-t}b_i^t, i \geq 3$. Overall, the objective function value of X is:

$$
\begin{aligned}
&D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=0}^{t-1} b_2^i(1-r)^{\alpha-i} + b_2^t(1-r)^{\alpha-t} \\
&+ \ldots + b_2^t(1-r)^{\alpha-u+1} + (1-r)^{u-t}b_3^t(1-r)^{\alpha-u} \\
&+ \min(b_4^t(1-r)^{u-t+1}, b_2^t(1-r))(1-r)^{\alpha-u-1} \\
&+ \sum_{i=u+2}^{\alpha} b_2^i(1-r)^{(\alpha-i)})
\end{aligned}
$$

Similarly, the objective value of X' needs to be discussed considering two cases:

The third highest skilled individual is the teacher of the second group:

$$
\begin{aligned}
&D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=0}^{t-1} b_2^i(1-r)^{\alpha-i} + b_3^t(1-r)^{\alpha-t} \\
&+ b_2^t(1-r)^{\alpha-t} + \ldots + b_2^t(1-r)^{\alpha-u+1} \\
&+ \min(b_4^t(1-r)^{u-t+1}, b_2^t(1-r))(1-r)^{\alpha-u-1} \\
&+ \sum_{i=u+2}^{\alpha} b_2^{i'}(1-r)^{\alpha-i})
\end{aligned}
$$

We can observe that since X' has the same second skill value as X at the round $u+1$, and X' adopts the local maximum groupings after round t, so X and X' has the same objective value in this case.

Any other individual with b_x^t is the teacher of the second group:

$$
\begin{aligned}
&D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=0}^{t-1} b_2^i(1-r)^{\alpha-i} + b_x^t(1-r)^{\alpha-t} \\
&+ b_2^t(1-r)^{\alpha-t} + ... + b_2^t(1-r)^{\alpha-u+1} \\
&+ min(b_3^t(1-r)^{u-t+1}, b_2^t(1-r))(1-r)^{\alpha-u-1} \\
&+ \sum_{i=u+2}^{\alpha} b_2^{i'}(1-r)^{\alpha-i})
\end{aligned}
$$

As before, X' is not better than X after u. In fact, it could be easily shown, X' gets worse (with higher value of the objective function in Equation 4.4), as we consider any other individuals with larger b_x^t as the teacher.

Overall, the optimal solution that only contains one round of non-local maximum grouping cannot achieve better objective value than ours. We can also observe two facts, for round $u > t$:

- When $b_2^{u'} > b_2^u$, the objective value of X' is the same or worse than X (Case 1, no swap after the round t).
- When $b_2^{u'} = b_2^u$, the objective values of two methods are equal (Case 2 after the swap round).

Multiple y one non-local optimas: Next we prove the case when there exists y non-local optimas in X'. Before the second non-local maximum grouping and after the the first non-local maximum grouping, the aforementioned two facts hold.

Assume the i-th non-local maximum grouping is adopted by X' at the round t_i. When $b_2^{v'} = b_2^v (t_1 < v < t_2)$, it is the exact same as the previous discussions: X' is not better than X before the round u_2. According to the algorithm, it always put the $3rd$ to $\frac{n}{2}$ members into the first group. Therefore, according to the lemma 2, this property will hold until the next non-local maximum grouping happens in X'.

Assume the second leader of X' at the round t_i is $b_{x_i}^{t_i}$, and X changes the second leader at the round z, so $b_2^{v'} > b_2^v (t_1 \leq v \leq z)$. There are y non-local maximum groupings in X' before the round z. Then, the objective value of X' will be:

$$
\begin{aligned}
& D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=0}^{t_1-1} b_2^i (1-r)^{\alpha-i} \\
& + b_{x_1}^{t_1}(1-r)^{\alpha-t_1} + (1-r)b_2^{t_1}(1-r)^{\alpha-t_1-1} \\
& + \ldots + (1-r)b_2^{t_1}(1-r)^{\alpha-t_2+1} \\
& + b_{x_2}^{t_2}(1-r)^{\alpha-t_2} + \cdots + b_2^{\alpha'})
\end{aligned}
\tag{4.8}
$$

And the objective value of X is:

$$\begin{aligned} & D(1-r)^{\alpha} + r\frac{n}{2}(\sum_{i=0}^{t_1-1} b_2^i(1-r)^{\alpha-i} \\ & + b_2^{t_1}(1-r)^{\alpha-t_1} + b_2^{t_1}(1-r)^{\alpha-t_1-1} \\ & + ... + b_2^{t_1}(1-r)^{\alpha-t_2+1} \\ & + b_2^{t_2}(1-r)^{\alpha-t_2} + \cdots + b_2^{\alpha}) \end{aligned} \tag{4.9}$$

Therefore, Equation 4.9 - Equation 4.8 is:

$$\sum_{i=0}^{y-1} b_2^{t_1}(1-r)^{\alpha-z+i} - \sum_{i=1}^{y} b_{x_i}^{t_i}(1-r)^{\alpha-t_i} \tag{4.10}$$

Since our algorithm always pick the second skilled person as the leader of the second group, $b_2^{t_1} \leq b_{x_1}^{t_1}(1-r)^{t_2-t_1-1}$, and $b_2^{t_1} \leq b_{x_2}^{t_2}$, and so on.. Therefore, Equation 4.10 is smaller or equal to zero, and X is not worse than X'. Overall, X is not worse than any optimal solution that contains any non-local maximum groupings. Hence, the proof. □

Proof. (of Thorem 16) With Lemmas 3 and 4, we prove that although there exists many local optima, DyGroups-Star in conjunction with DyGroups-Star-Local is not worse than any other local optima solutions considering the objective function in Equation 4.5. From Theorem 17, we prove that DyGroups-Star is not worse than any solution that does not produce local optima. Combining these three, we therefore prove that DyGroups-Star produces global optima for the TDG problem. □

4.5 Experimental Evaluation

We now present our experimental evaluation of DyGroups. We perform two experiments with human subjects and then we demonstrate the quantitative and runtime performance of DyGroups on synthetic data.

4.5.1 Human Subjects Experiments

The main purpose of this study is to experimentally examine: a. The effectiveness of peer learning, i.e., whether individual skills improve through interactions with peers. b. The effectiveness of DyGroups, relative to baseline solutions.

We consider an application on learning facts from peers through targeted dynamic groups formation.We present two experiments with human subjects. They are similar but they have been conducted independently; also the second experiment is more extensive. The experiments employ real workers hired on Amazon Mechanical Turk (AMT) to learn facts related to COVID-19 from their peers.

AMT Setup: We deploy multiple-choice questions as a Human Intelligent Task (HIT) consisting of facts and rumors about COVID-19. Those are shown to the workers as tasks[7]. Workers are paid $5 if they stick with the entire learning process. Each deployment was accessible for 24 hours and 1 hour is allotted to each worker.

Skill Assessment: Each HIT consists of 10 questions. The HIT questions also comprise a skill assessment test; the skill of each participant is set to be equal to the number of their correct answers, divided by 10.

---

[7]

Sample questions:
• What is the longest incubation time of COVID-19 in the record?
A. 14 days, B. 19 days, C. 20 days, D. More than 20 days

• Which action will help to prevent COVID-19?
A. Wash your hands regularly and thoroughly
B. Taking a hot bath, C. Drinking alcohol, D. None of the above

Experiment-1. We recruit $N = 64$ individuals. They first undergo PRE-QUALIFICATION: Workers are assigned individual assessment tests to estimate their skill level. Based on the qualification outcome, we split them into two Populations A,B each containing $n = 32$ participants. The split is random, under the constraint that the two populations have very similar skill distributions, and in particular the same average skill. Population A follows DYGROUPS with a learning rate of $r = 0.5$ and $k = 4$, while Population B follows a baseline solution KMEANS (details in Section 4.5.2). During the learning process, each Population alternates between these two steps: [GROUP-FORMATION]: Worker groups are formed, following the respective policy. The workers are asked to answer the questions collaboratively, by consulting with the rest of their peers in their group. [POST-ASSESSMENT]: Another test, akin to PRE-QUALIFICATION is performed to estimate the new skill of the individuals - and compute the learning gain after each round. The experiment consists of $\alpha = 3$ rounds.

Experiment-2. This is identical to Experiment-1 except it is conducted using N=128, that are split into four Populations of size $n = 32$ following DYGROUPS and the baselines KMEANS, LPA and PERCENTILE-PARTITIONS, further discussed in Section 4.5.2. The experiment consists of $\alpha = 2$ rounds.

Parameter justification: The choice of parameters r=0.5 and k=4 in our experiments is not arbitrary. Before running the actual experiment, we have made several initial deployments, where we hired workers of varying expertise from AMT and formed random groups of different size: small groups of size $2, 3, 4, 5$, and large groups of size $10, 12, 15$, and let them interact across multiple rounds. We have conducted pre-assessment and post-assessment tests on these deployments that steered us in the choice of parameters. From these initial deployments we have learned that for these assigned fact checking tasks, the new skill that individuals acquire after interaction with another higher skilled peer *is on average half of the difference*

of skills between between them prior to interaction. We also found that groups are most interactive and manageable when they contain $4-5$ people, and that worker engagement tends to dissipate if works are asked to participate in too many rounds. This leads us to set $r = 0.5$, k=4, and α=2 or 3 for the actual experiments. We also have observed that the one day time window is good enough for each round, and the workers do not need to spend more than one hour overall.

Summary of Results.

- *Observation I.* The aggregated skill improves with peer interaction (75% confidence interval), i.e. peer learning is effective. This can be seen in Figures 4.1 and 4.4(a) that show that the learning gain of Pre-Qualification and Post-Assessment scores after each round.
- *Observation II.* DyGroups outperforms the baselines with statistical significance (Figures 4.4(a) and 4.1). Interestingly, DyGroups outperforms even after the first round, which shows that it is very competitive even as a single-shot group formation algorithm.

  We also note two serendipitous features of DyGroups that are worth of further investigation:

- *Observation III.* DyGroups has higher worker retention than other baselines (Figures 4.3, 4.4(b)). This anecdotally indicates that under the same monetary rewards, the rate of skill improvement may be an important factor towards retaining participants in the process.
- *Observation IV.* As the amount of total skill left to be learned decreases with α, we expect that the aggregate learning must have a negative second derivative. However, in Figure 4.2, aggregated learning gain appears to increase linearly in the first rounds of DyGroups. This indicates that the learning rate may accelerate during the first rounds.

4.5.2 Synthetic Data Experiments

Experimental Setup Experiments are implemented in C++ and performed on a machine with Intel i5 CPU and 4GB Memory. In experiments involving randomness, we average over 10 different runs. Due to space constraints, we present a representative subset of our results.

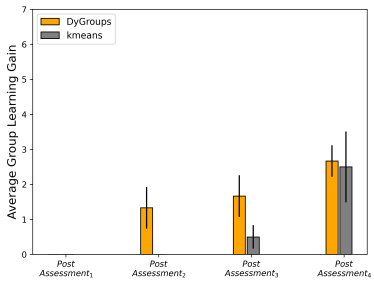


Figure 4.1 Experiment-1: Learning gain across rounds.

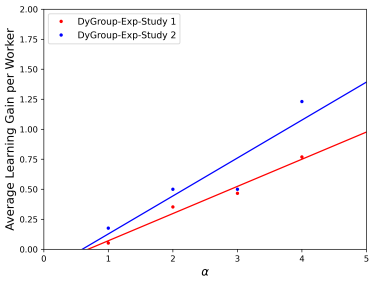


Figure 4.2 Linear fit to learning gain.

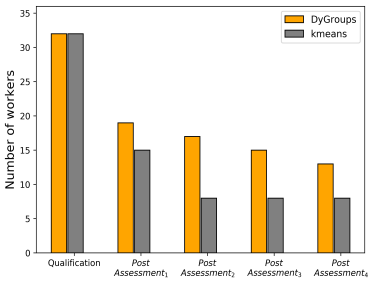


Figure 4.3 Experiment-1: Worker retention.

Baseline Algorithms. We note that there are no prior works on the dynamic groups formation problem. The closest related works are [5, 61], both of which focus on the one-shot grouping problem. We thus design a range of baseline algorithms, each employing a different grouping scheme applied for α rounds:

- RANDOM-ASSIGNMENT. Groups are selected randomly.
- PERCENTILE-PARTITIONS. Groups are computed using an algorithm from [5]. The algorithm involves a parameter p, which is set to 0.75, following the discussion in [5].
- LPA. This uses an algorithm from [61].
- K-MEANS. This is an alternative grouping heuristic that we devise as a baseline. The algorithm picks k random participants as group 'centers' and assigns the rest to their nearest group, that is not completely full.

We also implement BRUTE-FORCE, an exponential-time algorithm that solves the TGD problem optimally. Naturally, the algorithm can be run only for very small values of n and k, and α.

Parameters. We vary the following seven parameters: number of partipants:(n), number of groups:(k), number of rounds:(α), interaction mode:(star/clique), distribution of the initial skill values, learning rate:(r).

Distribution. We generate the initial skill values of people using the log-normal and Zipf distributions. Both are guaranteed to produce positive skill values (unlike

the normal distribution). We set the mean $\mu = e$ and the standard deviation $\sigma = \sqrt{e}$ for the log-normal distribution. On the other hand, the shape parameters of Zipf distribution are set to 2.3 and 10.

Summary of results. In the immediately subsequent sections, we can see the following.

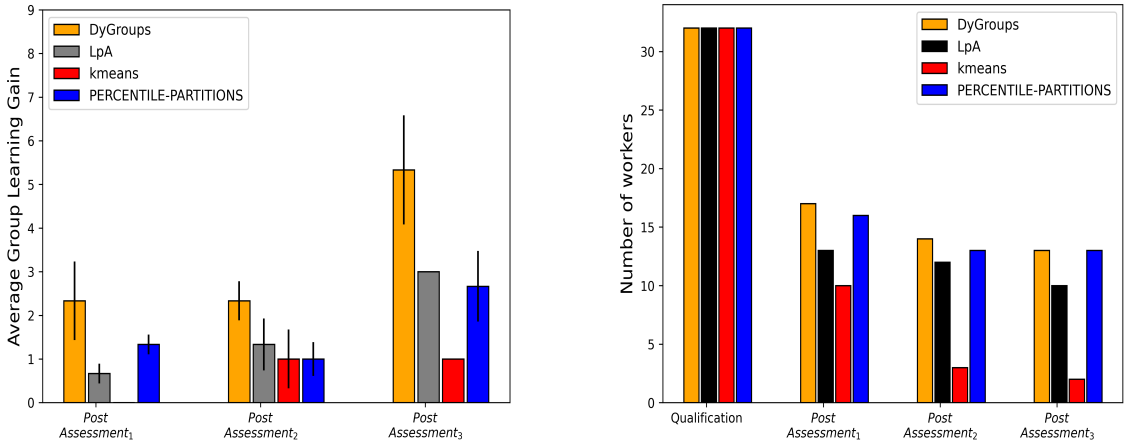
- *Section 4.5.2.* DyGroups is superior compared to other baselines in terms of improving the aggregated learning gain, under a range of parameter settings.
- *Section 4.5.2.* Brute-Force matches DyGroups-Star for $k = 2$ and small values of α, n as predicted by our theoretical results.
- *Section 4.5.2.* DyGroups methods induces significant learning gains relative to Random-Assignment.
- *Section 4.5.2.* DyGroups allows higher 'inequality' among participants relative to Random-Assignment.
- *Section 4.5.2.* DyGroups is highly scalable and therefore suitable for large scale real world applications.

Effectiveness Experiments We review experiments on the effectiveness of DyGroups.

Default Parameters. Unless otherwise noted, $k = 5$, $n = 10000$, $\epsilon = 0.05$, $r = 0.5$, $\alpha = 5$, *star* mode, with log-normally distributed initial skills. Deciding appropriate values for these parameters is often times application dependent. In our synthetic data experiments, these default values are decided based on the outcome of our real data experiments.

Varying n-[Figures 4.5(a,b)]. We record the aggregate learning gain **LG** as a function of n, for both initial skill distributions. The results demonstrate that aggregate learning gain increases with increasing n. DyGroups convincingly outperforms all other baselines.

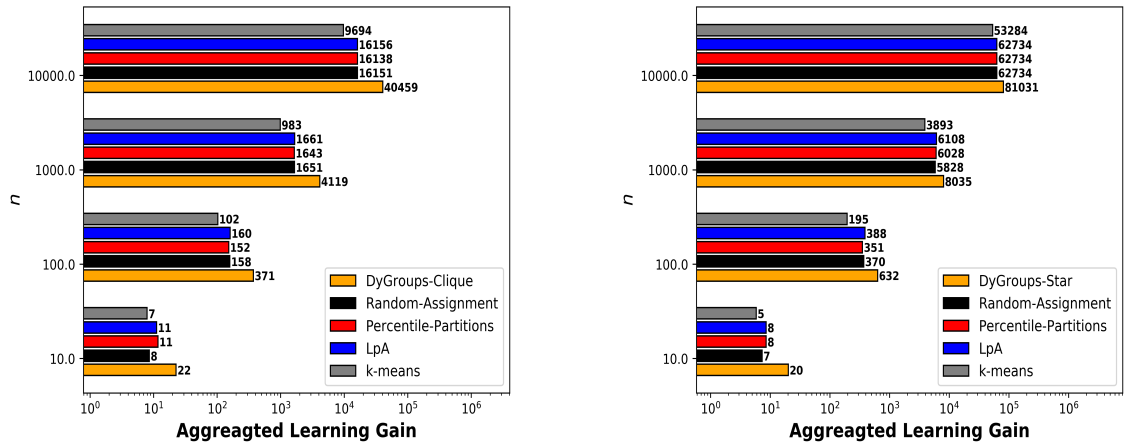
Varying k-[Figures 4.6(a,b)]. We record the learning gain as a function of the group size k. DyGroups outperforms other baselines. We also notice that **LG**



(a) Learning gain across rounds

(b) Worker retention

Figure 4.4 Results of Experiment-2.

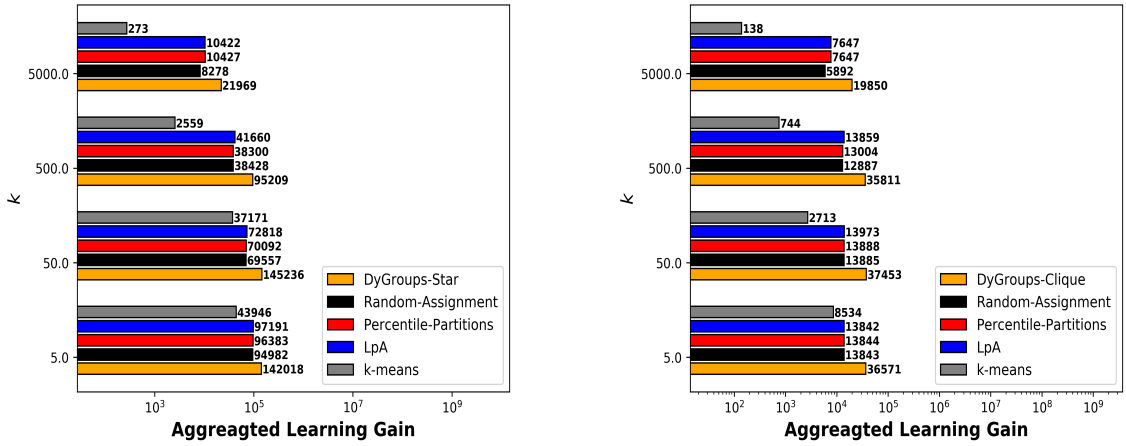


(a) Clique, log-Normal

(b) Star, Zipf

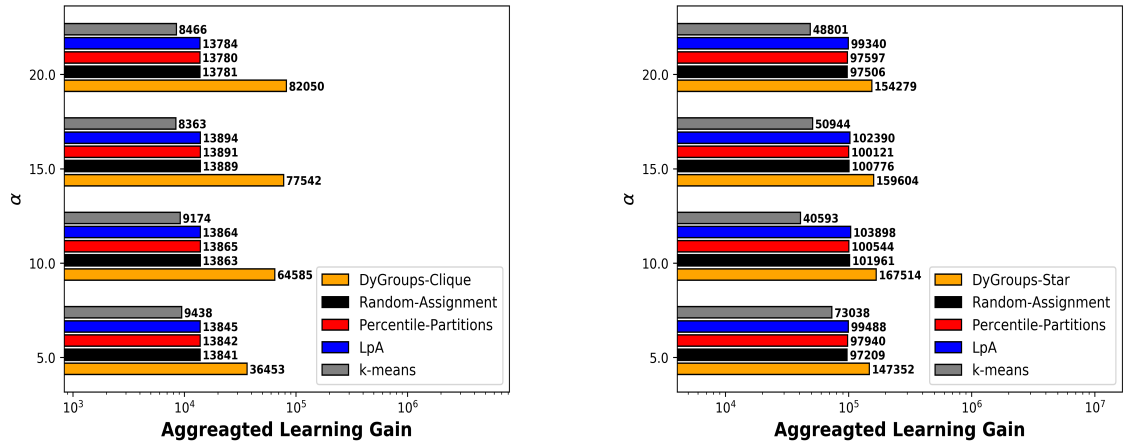
Figure 4.5 Aggregate Learning gain - varying n.

decreases with increasing k. This is expected since with a higher number of groups, not all groups get to have expert peers and therefore the learning gain decreases.



(a) Star, log-Normal (b) Clique, Zipf

Figure 4.6 Aggregate Learning gain - varying k.



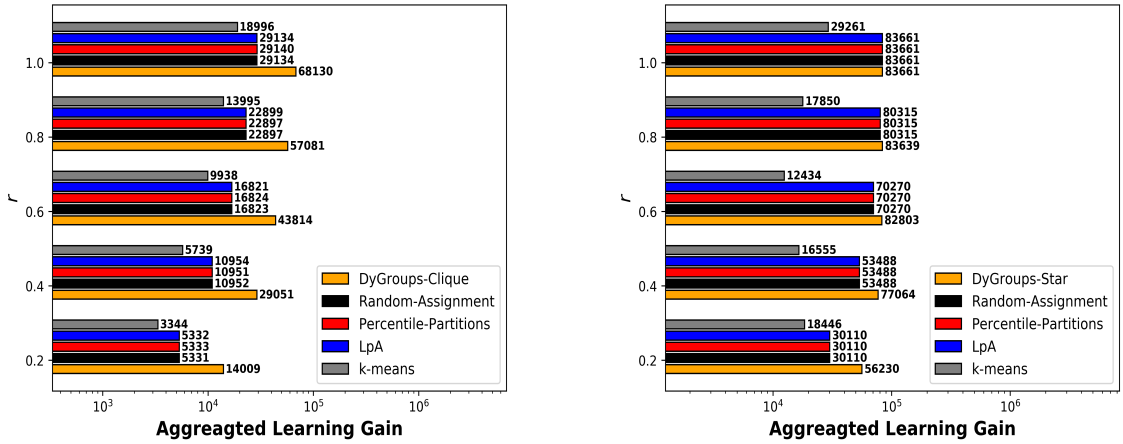
(a) Clique, Zipf (b) Star, log-Normal

Figure 4.7 Aggregate Learning gain - varying α.

Varying α-[Figures 4.7(a,b)]. As before, DyGroups convincingly wins. As expected, a higher α induces a higher aggregate learning gain.

Varying r-[Figures 4.8-4.9]. We record the aggregate learning gain as a function of the learning rate r. We can observe that DyGroups outperforms in the clique

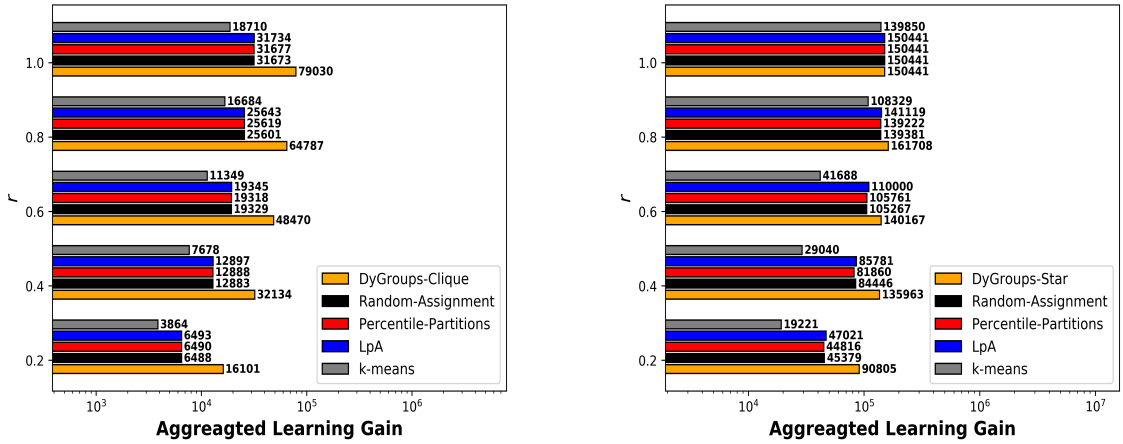
model for all of r values. In the special case of $r = 1$, by definition of the star mode, it takes $\log_{n/k}(n)$ rounds to make everyone reach the highest skill value for DyGroups and LpA.



(a) Clique, Zipf

(b) Star, Zipf

Figure 4.8 Aggregate Learning gain - varying r.



(a) Clique, log-Normal

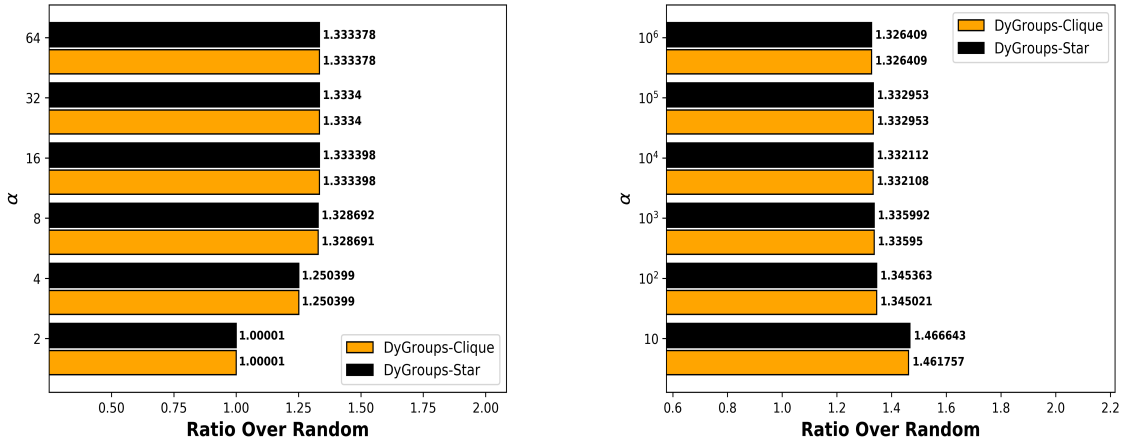
(b) Star, log-Normal

Figure 4.9 Aggregate Learning gain - varying r.

Star Interaction Mode with $k = 2$ We experimentally validate our theoretical claim presented in Section 4.4.3. We compare Brute Force with DyGroups-Star,

where we set $\alpha \in [1, 4]$, $n \in \{4, 6, 8\}$ and the skill values are picked from $[0, 1]$ uniformly. We run 1000 different experiments with the parameters picked at random. In all of them DyGroups-Star agrees with Brute-Force, i.e. it maximizes the aggregate learning gain.

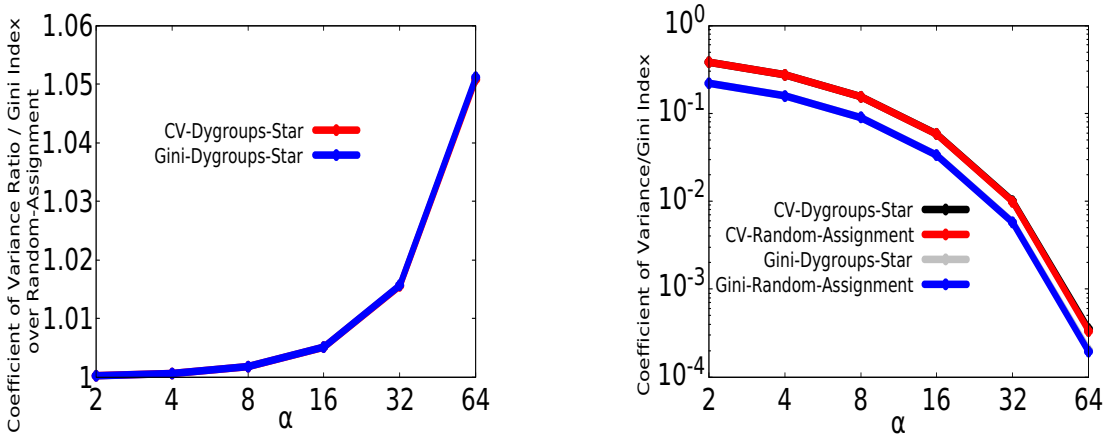
Learning Gain Relative to Random Groupings In Figure 4.10 we plot the ratio of the learning gain of DyGroups methods vs that of the Random-Assignment, as a function of α and n. Specifically, for a fixed $n = 10000$ we let α range over $\{2, 4, 6, 8, 16, 32, 64\}$. For a fixed $\alpha = 10$ we let n range over $\{10, 10^2, 10^3, 10^4, 10^5, 10^6\}$. It can be seen that DyGroups achieve up to 30% higher learning gain relative to random groupings over a small number of rounds. Interestingly, DyGroups-Star is comparable to DyGroups-Clique under this special case, and thus the simpler star mode may be a good proxy for the clique mode.



(a) Varying α (b) Varying n

Figure 4.10 Learning gain relative to Random-Assignment.

Fairness We measure the *inequality* of the distribution of skills in DyGroups vs that in Random-Assignment. For this experiment we use $r = 0.1$. We use



(a) Inequality Ratios

(b) Inequality Measures

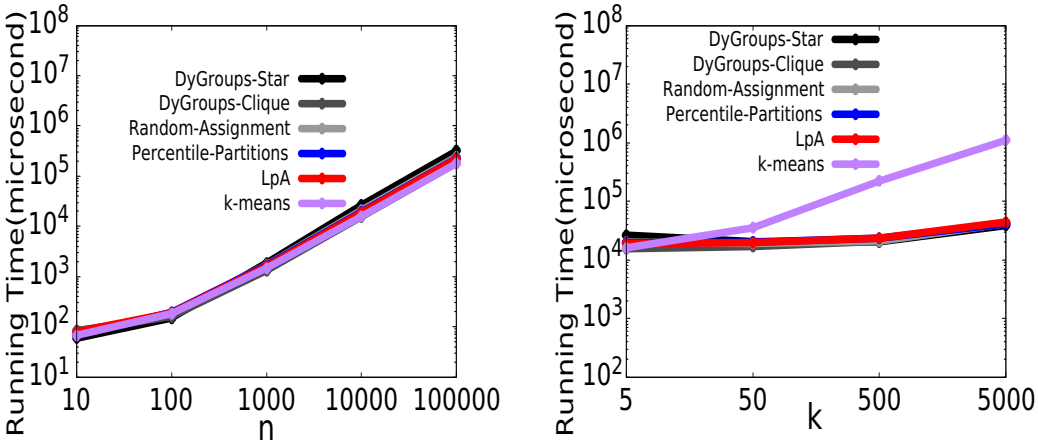
Figure 4.11 Inequality relative to RANDOM-ASSIGNMENT.

two metrics: the CV-coefficient of variation [8], and the well-known **Gini** coefficient [9]. Inequality **drops** with both methods (Figure 4.11(b)), something which may be expected due to the fact that there is an upper bound in the skill level. However, in Figure 4.11(a) we plot the **ratio** of the CV and the Gini index in DYGROUPS-STAR and RANDOM-ASSIGNMENT. We observe that DYGROUPS-STAR allow a higher inequality relative to RANDOM-ASSIGNMENT in all rounds, and that the gap between the two methods appears to be *widening* over time.

Running Time Experiments The running time of both DYGROUPS variants is dominated by the time to sort the skill values. This leads to excellent scaling behavior, shown in Figures 4.12 and 4.13. In practice, the time to run DYGROUPS is negligible. For instance, performing 5 rounds on $n = 10^5$ participants takes 0.18 sec.

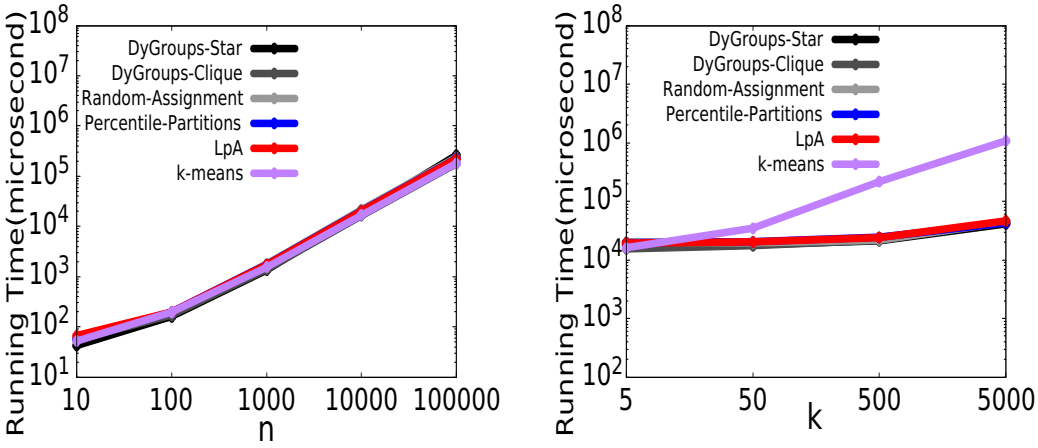
[8] CV is the ratio of the average by the standard deviation of skills.

[9] The Gini coefficient is $G = \frac{\sum_{i>j} |s_i - s_j|}{n \sum_i |s_i|}$, where s_i are skill levels.



(a) Varying n (b) Varying k

Figure 4.12 Running time (in micro sec), Star, log-Normal.



(a) Varying n (b) Varying k

Figure 4.13 Running time (in micro sec), Clique log-normal.

4.6 Related Work

We are unaware of any existing works that present models and algorithms for targeted groups formation in multiple rounds. In this section, we present the relevant existing works.

Groups formation: Designing quantitative models and algorithms for groups formation to optimize learning through peer interaction is first studied by Agrawal

et al. [3]. The authors propose learning gain models and algorithms for the one-shot groups formation problem. Esfandiari et al. [61], adapt the learning gain functions of [3] but also add the additional dimension of affinity to optimize peer learning. Besides [3, 61], another of our prior works studies group formation to optimize recommendation [123].

Related problems have been considered by the operation research community; the problem is always formalized as an Integer Programming Problem (ILP) and often solved using simulated annealing [29], branch-and-cut [150] or genetic algorithms [142]. From an algorithmic standpoint, Anagnostopoulos et al. [14] present a general framework for a task-assignment problem and a series of approximation algorithms with theoretical guarantees. In addition, Anagnostopoulos et al. [15], study how to form teams for a series of arriving tasks without overwhelming any expert, and there is some communication between teams. Lappas et al. [88], introduce a team formation problem that also involves skill requirements and communication costs. Rangapuram et al. [119] propose approximation algorithms for solving a constrained matching problem via the densest subgraph problem. Sanaz et al. [21, 22] solve the non-overlapping teams formation problems. Especially in [21], the authors aim to maximize the potential of students' learning in online classes.

Unlike these existing works, we study peer learning in a dynamic setting where group composition changes over time. While we adapt learning gain models of [3, 61], we note that the existing solutions do not extend to TDG.

Information Diffusion/ Gossip Propagation: TDG is in some sense a *diffusion* problem: knowledge is diffused via pair-wise interactions, and the objective is to attain the maximum possible diffusion (as measured by the total skill) in a specified number of rounds. Diffusion problems are very well studied in various contexts, including information diffusion maximization in social networks (e.g.

[73,74,125,135,138]), and gossip propagation (e.g. [92,130]). A tangential topic is the problem of influence maximization, introduced by Kempe, Kleinberg, and Tardos [83].

However, all these works assume the presence of a graph topology or network. Conversely, TDG *assumes a fully connected underlying network, and instead asks for a controlled utilization of its resources over time in a group-like manner.*

4.7 Discussion and Future Work

We believe that our study can stimulate further research in this space, along the following directions.

Alternative formulations. Our formulation assumes equal-size groups and linear learning gain. These settings are directly adopted from related works [3, 5, 76, 79], including education literature. We note that DyGroups can be adapted for the case when groups have varying sizes. Of course, more complicated models are conceivable. A particularly interesting problem is to study settings where the learning gain depends on additional factors that capture "intrinsic learning ability", e.g. a time-evolving affinity among individuals [5] that impact learning, or different learning rates for the participants. One possible way to model the former problem is to solve a bi-criteria optimization problem, with the goal of forming dynamic groups where both affinity and skill evolves across rounds. Another interesting question, motivated by Observation III in Section 4.5.1 is the impact of *retention* on the aggregate learning gain. A faster overall learning gain may still higher satisfaction among participants, and thus create a positive feedback loop.

DyGroups for more groups. Recall that we proved the optimality of DyGroups-Star for the case k=2. Clearly, DyGroups can still be used to solve the problem when $k > 2$ and we conjecture that DyGroups-Star is still optimal. That said, the computational complexity of the problems for different interaction modes is an open problem for larger values of k, possibly of independent theoretical interest.

Other learning gain functions. As long as the learning gain function is concave, DyGroups can be adapted to solve TDG. Our initial research suggests that for non-linear concave learning gain functions, DyGroups is not optimal, thus raising questions on the approximability of the optimal, or other theoretical guarantees.

Fairness. We have only scratched the surface with respect to fairness. We believe that studying bi-criteria optimization problems with respect to fairness and learning gain is an extremely interesting theoretical and practical issue, even within the scope of relatively limited peer learning models.

4.8 Conclusion

We initiate the study of peer-learning processes in *rounds*. The problem is motivated by practical considerations of groups in online social networks or offline classroom learning. In each round, the participants are split into groups and learn from interactions within their group. The objective is to find a *sequence* of groupings that will maximize the total knowledge at the end of the process. We introduce a model and an associated algorithmic framework DyGroups for this problem. Using the insights we gain from our theoretical study, we design experiments with human subjects that provide evidence corroborating our hypothesis that the choice of groupings can indeed significantly impact the total amount of learning.

CHAPTER 5

RECOMMENDING DEPLOYMENT STRATEGIES FOR COLLABORATIVE TASKS

5.1 Introduction

Despite becoming a popular mean of deploying tasks, crowdsourcing offers very little help to requesters. In particular, task deployment requires that requesters identify *appropriate deployment strategies*. A strategy involves the interplay of multiple dimensions: *Structure* (whether to solicit the workforce sequentially or simultaneously), *Organization* (to organize it collaboratively or independently), and *Style* (to rely on the crowd alone or on a combination of crowd and machine algorithms). A strategy needs to be commensurate to *deployment parameters* desired by a requester, namely, a lower-bound on quality, an upper-bound on latency, and an upper-bound on cost. For example, for a sentence translation task, a requester wants the translated sentences to be at least 80% as good as the work of a domain expert, in a span of at most 2 days, and at a maximum cost of $100. Till date, the burden is entirely on requesters to design deployment strategies that satisfy desired parameters. Such parameters are to be estimated for each strategy. If their estimation satisfies desired values, the strategy is deemed suitable for a task. Otherwise, the parameters need to be revisited. Our effort in this work is to present a formalism and computationally efficient algorithms to recommend multiple strategies (namely k) to the requester that are commensurate to her deployment parameters, primarily for collaborative tasks.

A recent work [35] investigated empirically the deployment of text creation tasks in Amazon Mechanical Turk (AMT). The authors validated the effectiveness of different strategies for different collaborative tasks, such as text summarization and text translation, and provided evidence for the need to guide requesters in choosing

the right strategy. In this chapter, we propose *to automate strategy recommendation*. This is particularly challenging because the estimation of the cost, quality and latency of a strategy for a given deployment request must account for many factors. *We conjecture that worker availability on the platform is a major factor to account for in task deployment. Our contributions are: we express deployment strategies of collaborative tasks as a function of the availability of qualified workers, we empirically verify that conjecture, we formalize a set of problems that are related to recommending deployment strategies, present principled algorithms, and validate them with extensive experiments.*

To realize our contributions, we develop StratRec (refer to Figure 5.1), an optimization-driven middle layer that sits between requesters, workers, and platforms. StratRec has two main modules: *Aggregator* and *Alternative Parameter Recommendation* (ADPaR in short). *Aggregator* is responsible for recommending k strategies to a batch of such incoming deployment requests, considering worker availability. If the platform does not have enough qualified workers to satisfy all requests, *Aggregator triages them by optimizing platform-centric goals, i.e., to maximize throughput or pay-off (details in Section 5.2.2).* Unsatisfied requests are sent to the *Alternative Parameter Recommendation* module (ADPaR), that recommends different deployment parameters for which k strategies are available.

In principle, *recommending deployment strategies involves modeling worker availability considering their skills for the tasks that require deployment.* This gives rise to a complex function that estimates parameters (quality, latency, and cost) of a strategy considering worker skills, task types, and worker availability. As the first ever principled investigation of strategy recommendation in crowdsourcing, we first make a binary match between workers' skills and task types and then estimate strategy parameters considering those workers' availability. Worker availability is captured as a probability distribution function (pdf) by leveraging historical data on a platform.

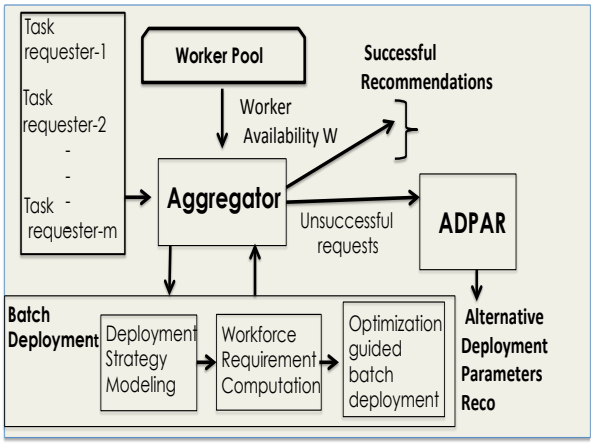


Figure 5.1 StratRec framework.

For example, the pdf can capture that there is a 70% chance of having 7% of the workers and a 30% chance of having 2% of the workers available who are suitable to undertake a certain type of task. In expectation, this gives rise to 5.5% of available workers. If a platform has 4000 total workers available to undertake a certain type of task, that gives rise to a total of 220 available workers in an expected sense. StratRec works with such *expected values*.

Contribution 1. Modeling and Formalism: We present a general framework StratRec for modeling quality, cost, and latency of a set of collaborative tasks, when deployed based on a strategy considering worker availability (Section 5.3.1). The first problem we study is *Batch Deployment Recommendation* inside to deploy a batch of tasks to maximize two different platform-centric criteria: task throughput and pay-off. After that, unsatisfied requests are sent one by one to the *Alternative Parameter Recommendation* module (ADPaR). ADPaR solves an optimization problem that recommends alternative parameters for which k deployment strategies exist. For instance, if a request has a very small latency threshold that cannot be attained based on worker availability, ADPaR may recommend to increase the latency and cost thresholds to find k legitimate strategies. ADPaR does not arbitrarily choose the alternative deployment parameters. It recommends those

alternative parameters that are *closest*, i.e., minimizing the ℓ_2 distance to the ones specified.

Contribution 2. Algorithms: In Section 5.3, we design BatchStrat, a unified algorithmic framework to solve the *Batch Deployment Recommendation* problem. BatchStrat is greedy in nature and provides exact results for the throughput maximization problem, and a 1/2-approximation factor for the pay-off maximization problem (which is NP-hard). In Section 5.4, we develop ADPaR-Exact to solve ADPaR that is geometric and exploits the fact that our objective function is monotone (Equation 5.3). Even though the original problem is defined in a continuous space, we present a discretized technique that is exact. ADPaR-Exact employs a sweep-line technique [31] that gradually relaxes quality, cost, and latency, and is guaranteed to produce the tightest alternative parameters for which k deployment strategies exist.

Contribution 3. Experiments: We conduct comprehensive real-world deployments for text editing applications with real workers and rigorous synthetic data experiments (Section 5.5). The former validate that worker availability *varies over time, and could be reasonably estimated through multiple real world deployments.* It also shows *with statistical significance that cost, quality, latency have a linear relationship with worker availability for text editing tasks.* In Section 5.7, we discuss how StratRec could be adapted for tasks that do not exhibit such linear relationships. Our synthetic experiments compare our solutions with realistic baselines and validate scalability and theoretical bounds.

5.2 Framework and Problem

Example 3. *Assume there are three ($m = 3$) task deployment requests for different types of collaborative text editing tasks. The first requester d_1 is interested in deploying sentence translation tasks for two days (out of seven days), at a cost up to \$*100 *(out of* \$600 *max), and expects the quality of the translation to reach at least* 40% *of domain*

expert quality. Table 5.1 presents all three deployment requests after normalization between $[0-1]$*. For the purpose of this example, we assume worker availability is a pdf with a* 50% *probability of having* 700 *workers and a* 50% *probability of having* 900 *workers out of* 1000 *suitable workers for text editing tasks available for the next seven days. Thus, expected worker availability* W *is* 0.8 *and we set* $k = 3$*.*

For the purpose of illustration, $\mathcal{S}$ *consists of the set of four deployment strategies, as shown in Figure 5.2:* SIM-COL-CRO, SEQ-IND-CRO, SIM-IND-CRO, SIM-IND-HYB. *To ease understanding, we name them as* s_1*,* s_2*,* s_3*,* s_4*, respectively.* s_1 *costs* \$150 *and takes* 2 *days (out of seven days) and ensures at least a* 50% *quality.* s_2*,* s_3*,* s_4 *have corresponding parameters. Strategy parameters are normalized and presented in the last column of Table 5.1.* StratRec *intends to recommend strategies from* $\mathcal{S}$ *to the requesters. If* StratRec *cannot produce* k *strategies for all three requests because of limited worker availability, it selects a subset that maximizes a platform-centric goal (throughput or pay-off) and sends the rest to* ADPaR*.*

5.2.1 Data Model

Crowdsourcing Platforms & Tasks: A platform is designed to crowdsource tasks, deployed by a set of requesters and undertaken by crowd workers. We consider collaborative tasks such as sentence translation, text summarization, and puzzle solving [114, 117]. Conceptually speaking, we do not restrict ourselves to specific task types. In our experiments, we illustrate text editing tasks.

Tasks of the same type are usually deployed in batches (referred to as Human Intelligence Tasks). For the rest of the work, a task refers to a batch of tasks of the same type.

Deployment Strategies: A deployment strategy [81] instantiates three dimensions: *Structure* (sequential or simultaneous), *Organization* (collaborative or independent), and *Style* (crowd-only or crowd and algorithms). We rely on common

deployment strategies [35,81] and refer to them as $\mathcal{S}$. Figure 5.2 enlists some strategies that are suitable for text translation tasks (from English to French in this example). For instance, *SEQ-IND-CRO* in Figure 5.2(a) dictates that workers complete tasks sequentially (*SEQ*), independently (*IND*) and with no help from algorithms (*CRO*). In *SIM-COL-CRO* (Figure 5.2(b)), workers are solicited in parallel (*SIM*) to complete a task collaboratively (*COL*) and with no help from algorithms (*CRO*). The last strategy *SIM-IND-HYB* dictates a hybrid work style (*HYB*) where workers are combined with algorithms, for instance with Google Translate.

Table 5.1 Deployment Requests and Strategies

	Quality	Cost	Latency
d_1	0.4	0.17	0.28
d_2	0.8	0.2	0.28
d_3	0.7	0.83	0.28
s_1	0.5	0.25	0.28
s_2	0.75	0.33	0.28
s_3	0.8	0.5	0.14
s_4	0.88	0.58	0.14

In principle, the number of possible strategies is infinite since a strategy can combine structure, organization and style in any order and any number of times. All strategies are applicable regardless of the type of collaborative task. A platform could provide the ability to implement some strategies. For instance, communication between workers enables `SEQ` while collaboration enables `COL`. Additionally, coordination between machines and humans may enable `HYB`. Therefore, strategies could be implemented inside or outside platforms. In the latter, a platform could be used solely for hiring workers who are then redirected to an environment where strategies

are implemented. In all cases, we will assume that for a given platform, a set of strategies $\mathcal{S}$.

Task Requests and Deployment Parameters: A requester intends to find one or more strategies (notationally k, a small integer) for a deployment d with parameters on quality, cost, and latency ($d.quality$, $d.cost$, $d.latency$) such that, when a task in d is deployed using strategy $s \in S$, it is estimated to achieve a crowd contribution quality $s.quality$, by spending at most $s.cost$, and the deployment will last at most $s.latency$. Using Example 3, the deployment parameters are 40% on quality, 2 days on latency, and $100 in cost. For simplicity, these parameters are normalized in $[0 - 1]$.

s is suitable to be recommended to d, if $s.quality \geq d.quality$ & $s.cost \leq d.cost$ & $s.latency \leq d.latency$. Therefore, we must estimate the parameters $s.quality$, $s.cost$, $s.latency$ for each s, for a deployment d. Estimation of these parameters require accounting for the worker pool and their skills who are available to undertake the tasks in d. *A simple yet reasonable approach to that is to first match task types in a deployment request with workers' skills to select a pool of workers. Following that, we account for* worker availability *from this selected pool, since the deployed tasks are to be done by those workers. Thus, the (estimated) quality, cost and latency of a strategy for a task is a function of worker availability, considering a selected pool of workers who are suitable for the tasks.* For simplicity of illustration, we will use the terms estimated quality (resp., estimated cost, estimated latency) and quality (resp., cost, latency) interchangeably.

Worker Availability: Given a filtered pool of workers who are suitable for a deployment, worker availability is a discrete random variable and is represented by its corresponding distribution function (pdf), which gives the probability of the proportion of workers who are suitable and available to undertake tasks of a certain type within a specified time $d.latency$ (refer to Example 3). This pdf is computed from

historical data on workers' arrival and departure on a platform. StratRec computes the expected value of this pdf to represent the available workforce W, as a normalized value in $[0, 1]$. In the remainder of the work, worker availability stands for worker availability in expectation, unless otherwise specified. How to accurately estimate worker availability is an interesting yet orthogonal problem and not our focus here. In our real data experiments (Section 5.5.1), we describe how to compute worker availability through multiple deployment efforts.

5.2.2 Proposed Framework

StratRec is an optimization-driven middle layer that sits between requesters, workers, and platforms. At any time, a crowdsourcing platform has a batch of m deployment requests each with its own parameters as defined above, coming from different requesters. StratRec is composed of two main modules - *Aggregator* and *Alternative Parameter Recommendation* (or ADPaR). Given a batch of m deployment requests, let the i-th task deployment d_i be associated with parameters $d_i.quality$, $d_i.cost$ and $d_i.latency$. These requests, along with worker availability estimation W, once received, are sent to the *Aggregator*. *Aggregator* is responsible to recommend strategies to these batch of requests to optimize different goal. For that, it first consults with the *Deployment Strategy Modeling Module* to obtain model parameters of different candidate strategies. Then, it consults with the *Workforce Requirement Computation* module to estimate workforce requirement to satisfy all m requests.

If the platform does not have enough available workers to satisfy all m requests (from a filtered pool of workers who are qualified for the tasks in the requests), *Aggregator* triages them to optimize platform-centric goals, namely task throughput or pay-off[1]. Based on the goal, the *Optimization guided batch deployment* module is then invoked to select a subset of these requests that optimizes the underlying goal

[1]Although, by design StratRec, is extensible to optimize other types of goals, such as worker-centric goals, as long as the required inputs are available.

and recommends k strategies for those requests. Each unsatisfied request d_i is sent to ADPaR that recommends an alternative deployment d_i' to the requester for which there exists k deployment strategies.

5.2.3 Problem Definitions

We are now ready to formalize our two problems: batch deployment recommendation (*Aggregator*) and alternative parameter recommendation (ADPaR).

Problem 3. Batch Deployment Recommendation: *Given an optimization goal F, a set $\mathcal{S}$ of strategies, a batch of m deployment requests from different requesters, where the i-th task deployment d_i is associated with parameters $d_i.quality$, $d_i.cost$ and $d_i.latency$, and worker availability W, distribute W among these requests by recommending k strategies for each request, such that F is optimized.*

The high level problem optimization problem could be formalized as:

$$\begin{aligned} &Maximize\ F = \sum f_i \\ &s.t. \sum \vec{w_i} \leq W\ \ AND \\ &d_i\ is\ successful \end{aligned} \tag{5.1}$$

where f_i is the optimization value of deployment d_i and $\vec{w_i}$ is the workforce required to successfully recommend k strategies it. A deployment request d_i is successful, if for each of the k strategies in the recommended set of strategies S_d^i, the following three criteria are met: $s.cost \leq d_i.cost$, $s.latency \leq d_i.latency$ and $s.quality \geq d_i.quality$.

Using Example 3, d_3 is successful, as it will return $S_d^3 = \{s_2, s_3, s_4\}$, such that $d_3.cost \geq s_4.cost \geq s_3.cost \geq s_2.cost$ & $d_3.latency \geq s_4.latency \geq s_3.latency \geq s_2.latency$ & $d_3.quality \leq s_4.quality \leq s_3.quality \leq s_2.quality$, and it could be deployed with the available workforce $W = 0.8$.

In this work, F is designed to maximize one of two different platform centric-goals: task throughput and pay-off.

Throughput maximizes the total number of successful strategy recommendations without exceeding W. Formally speaking,

$$\begin{aligned}
&\text{Maximize} \sum_{i=1}^{m} x_i \\
&\text{s.t.} \sum x_i \times \vec{w_i} \leq W \\
&x_i = \begin{cases} 1 & d_i.cost \leq s_j.cost \text{ AND} \\ & d_i.latency \leq s_j.latency \text{ AND} \\ & d_i.quality \geq s_j.quality \text{ AND} \\ & |S_d^i| = k, \forall i = 1, \ldots, m; j = 1, \ldots, |\mathcal{S}| \\ 0 & otherwise \end{cases}
\end{aligned} \tag{5.2}$$

Pay-off maximizes $d_i.cost$, if d_i is a successful deployment request without exceeding W. The rest of the formulation is akin to Equation 5.2.

Problem 4. Alternative Parameter Recommendation: *Given a deployment d, worker availability W, a set of deployment strategies $\mathcal{S}$, and a cardinality constraint k, ADPaR recommends an alternative deployment d' and associated k strategies, such that, the Euclidean distance (ℓ_2) between d and d' is minimized.*

Formally, our problem could be stated as a constrained optimization problem:

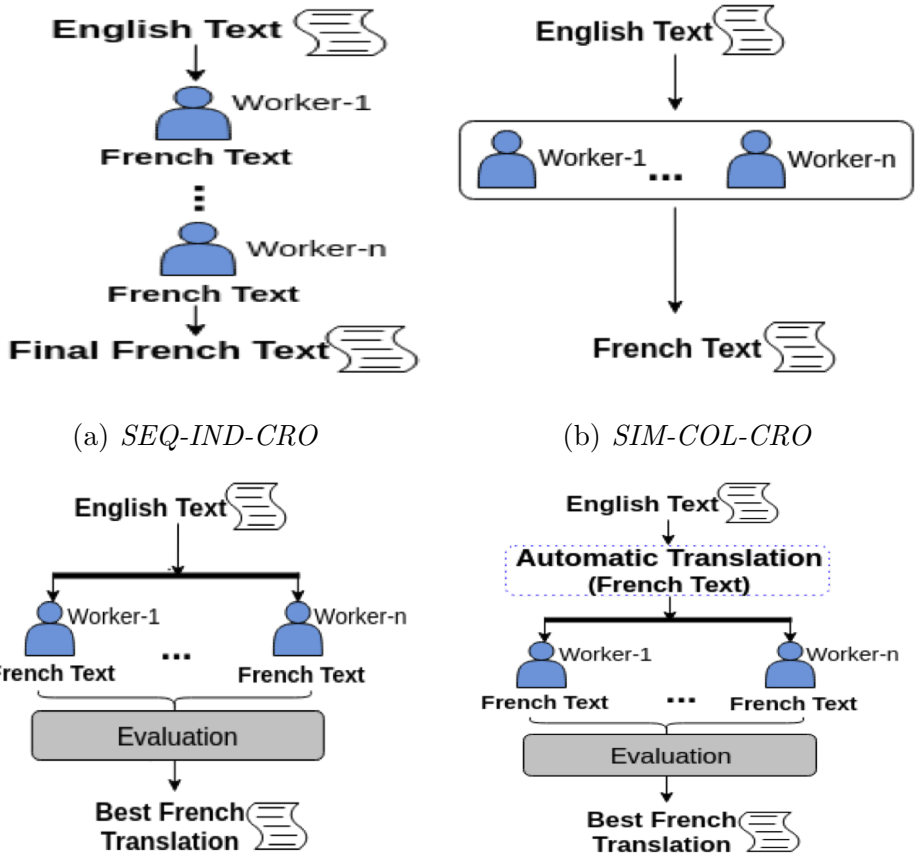
$$
\begin{aligned}
min \quad & (d'.cost - d.cost)^2 + (d'.latency - d.latency)^2 \\
& + (d'.quality - d.quality)^2 \\
s.t. \quad & \sum_{j=1}^{|\mathcal{S}|} x_j = k \\
& x_j = \begin{cases} 1 & d'.cost \leq s_j.cost \ AND \\ & d'.latency \leq s_j.latency \ AND \\ & d'.quality \geq s_j.quality \\ 0 & otherwise \end{cases}
\end{aligned} \tag{5.3}
$$

Based on Example 3, if ADPaR takes the following input values $d_1 : (0.4, 0.17, 0.28)$ and $\mathcal{S}$. For d_1, the alternative recommendation should be $(0.4, 0.5, 0.28)$ with three strategies s_1, s_2, s_3.

5.3 Deployment Recommendation

We describe our proposed solution for *Batch Deployment Recommendation* (Problem 3). Given m requests and W, the *Aggregator* invokes BatchStrat, our unified solution to solve the batch deployment recommendation problem. There are three major steps involved. BatchStrat first obtains model parameters of a set of candidate strategies (Section 5.3.1), then computes workforce requirement to satisfy these requests (Section 5.3.2), and finally performs optimization to select a subset of m deployment requests, such that different platform-centric optimization goals could be achieved (Section 5.3.3).

We first provide an abstraction which serves the purpose of designing BatchStrat. Given m deployment requests and W workforce availability, we intend to compute a two dimensional matrix $\mathcal{W}$, where there are $|\mathcal{S}|$ columns that map to available deployment strategies and m rows of different deployment requests. Figure 5.3(a)



(a) *SEQ-IND-CRO* (b) *SIM-COL-CRO*

(c) *SIM-IND-CRO* (d) *SIM-IND-HYB*

Figure 5.2 Deployment strategies.

shows the matrix built for Example 3. A cell w_{ij} in this matrix estimates the workforce required to deploy i-th request using j-th strategy. This matrix $\mathcal{W}$ is crucial to enable platform centric optimization for batch deployment.

5.3.1 Deployment Strategy Modeling

BatchStrat first performs deployment strategy modeling to estimate quality, cost, latency of a strategy s for a given deployment request d. As the first principled solution, it models these parameters as a linear function of worker availability, from the filtered pool of workers whose profiles match tasks in the deployment request. However, in Section 5.7, we describe how StratRec could be adapted for tasks that do not exhibit such linear relationships. Therefore, if d is deployed using strategy s, the quality parameter of this deployment is modeled as:

$$s_d.quality = \alpha_{qds}.(w_{qds}) + \beta_{qds} \tag{5.4}$$

Our experimental evaluation (Table 5.6) in Section 5.5.1, performed on AMT validates this linearity assumption with 90% statistical significance for two text editing tasks.

Model parameters α and β are obtained for every s, d, and parameter (quality, cost, latency) combination, by fitting historical data to this linear model. Once these parameters are known, BatchStrat uses Equation 5.4 again to estimate workforce requirement w_{qds} to satisfy quality threshold (cost and latency like-wise) for deployment d using strategy s. We repeat this exercise for each $s \in \mathcal{S}$, which comprises our set of candidate strategies for a deployment d.

5.3.2 Workforce Requirement Computation

The goal of the *Workforce Requirement Computation* is to estimate workforce requirement per (deployment, strategy) pair. It performs that in two sub-steps, as

described below.

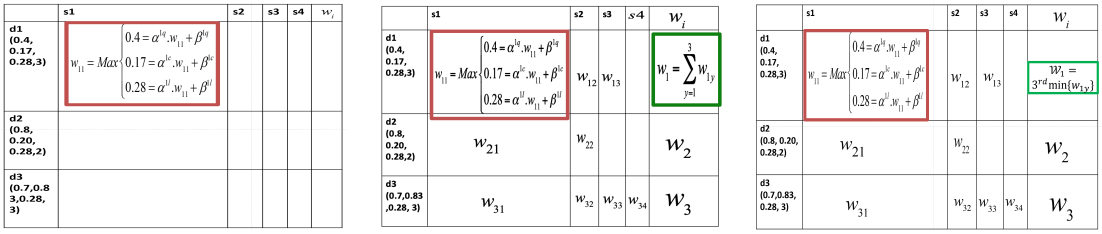
(1) Computing Matrix $\mathcal{W}$: The first step is to compute $\mathcal{W}$, where $w_{i,j}$ represents the workforce requirement of deploying d_i with strategy s_j. Recall that in Equation 5.4, as long as for a deployment d_i, the deployment parameters on quality, cost, and latency, i.e., $d_i.quality$, $d_i.cost$ and $d_i.latency$ are known, for a strategy, s_j, we can compute $w_{i,j}$, i.e., that is the minimum workforce needed to achieve those thresholds, by considering the equality condition, i.e., $s_j.quality = d_i.quality$ (similarly for cost and latency), and solving Equation 5.4 for w, with known (α, β) values. Using Example 3, the table in Figure 5.3(a) shows the rows and columns of matrix $\mathcal{W}$ and how a workforce requirement could be calculated for w_{11}. Basically, once we solve the workforce requirement of quality, cost, and latency(w_{qij}, w_{cij}, w_{lij}), the overall workforce requirement of deploying d_i using s_j is the maximum over these three requirements. Formally, they could be stated as follows:

$$w_{ij} = Max \begin{cases} d_i.quality = \alpha_{qij} w_{qij} + \beta_{qij} \\ d_i.cost = \alpha_{cij} w_{cij} + \beta_{cij} \\ d_i.latency = \alpha_{lij} w_{lij} + \beta_{lij} \end{cases}$$

Using Example 3, w_{11} is the maximum over $\{w_{q11}, w_{c11}, w_{l11}\}$. Figure 5.3(a) shows how w_{11} needs to be computed for deployment d_1 and strategy s_1 for the running example.

Running Time: Running time of computing $\mathcal{W}$ is $O(m|\mathcal{S}|)$, since computing each cell w_{ij} takes constant time.

(2) Computing Workforce Requirement per Deployment: For a deployment request d_i to be successful, BatchStrat has to find k strategies, such that each satisfies the deployment parameters. In step (2), we investigate how to make compute workforce requirement for all k strategies, for each d_i. The output of this



(a) Requirement for (d_1, s_1) (b) Aggregated requirement per request (Sum) (c) Aggregated requirement per request (Max)

Figure 5.3 Computing workforce requirement.

step produces a vector $\vec{W}$ of length m, where the i-th value represents the aggregated workforce requirement for request d_i. Computing $\vec{W}$ requires understanding of two cases:

- **Sum-case:** It is possible that the task designer intends to perform the deployment using all k strategies. Therefore, the minimum workforce (w_i) needed to satisfy cardinality constraint k_i is $\Sigma_{y=1}^{k} w_{iy}$ (where w_{iy} is the y-th smallest workforce value in row i of matrix $\mathcal{W}$.
- **Max-case:** The task designer intends to only deploy one of the k recommended strategies - in that case, $w_i = w_{iy}$, (where w_{iy} is the k-th smallest workforce value in row i of matrix $\mathcal{W}$).

Figures 5.3(b) and 5.3(c) represent how $\vec{W}$ is calculated considering sum-case and max-case, respectively.

Running Time: The running time of computing the aggregated workforce requirement of the i-th deployment request is $O(|\mathcal{S}|klog|\mathcal{S}|)$, if we use min-heaps to retrieve the k smallest numbers. The overall running time is again $O(mk\ log|\mathcal{S}|)$.

5.3.3 Optimization-Guided Batch Deployment

Finally, we focus on the optimization step of BatchStrat, where, given $\vec{W}$, the objective is to distribute the available workforce W among m deployment requests such that it optimizes a platform-centric goal F. Since W can be limited, it may not be

possible to successfully satisfy all deployment requests in a single batch. This requires distributing W judiciously among competing deployment requests and satisfying the ones that maximize platform-centric optimization goals, i.e., throughput or pay-off.

At this point, a keen reader may notice that the batch deployment problem bears resemblance to a well-known discrete optimization problem that falls into the general category of assignment problems, specifically, Knapsack-type of problems [70]. The objective is to maximize a goal (in this case, throughput or pay-off), subject to the capacity constraint of worker availability W. In fact, depending on the nature of the problem, the optimization-guided batch deployment problem could become intractable.

Intuitively, when the objective is only to maximize throughput (i.e., the number of satisfied deployment requests), the problem is polynomial-time solvable. However, when there is an additional dimension, such as pay-off, the problem becomes NP-hard problem, as we shall prove next.

Theorem 18. *The Pay-Off maximization problem is NP-hard.*

Proof. (sketch): To prove NP-hardness, we reduce an instance of the known NP-hard problem 0/1 Knapsack problem [70] problem to an instance of the Pay-off Maximization problem. Given an instance of the 0/1 Knapsack problem, an instance of our problem could be created as follows: an item i of weight w_i and value v_i represents a deployment request d_i with minimum workforce requirement w_i and pay-off v_i. Clearly this transformation is performed in polynomial time. After that, it is easy to notice that a solution exists for the 0/1 Knapsack problem iff the same solution solves our Pay-Off maximization problem. □

Our proposed solution bears similarity to the greedy approximation algorithm of the Knapsack problem [78]. The objective is to sort the deployment strategies in non-increasing order of $\frac{f_i}{\vec{w_i}}$. The algorithm greedily adds deployments based on this

sorted order until it hits a deployment d_i that can no longer be satisfied by W, that is, $\Sigma_{x=1..i}\ d_i > W$. At that step, it chooses the better of $\{d_1, d_2, d_{i-1}\}$ and d_i and the process continues until no further deployment requests could be satisfied based on W. Lines $4-8$ in Algorithm BatchStrat describe those steps.

Running Time: The running time of this step is dominated by the sorting time of the deployment requests, which is $O(m\ logm)$.

Algorithm 7 Algorithm BatchStrat

1: **Input:** m deployment requests, $\mathcal{S}$, objective function F, available workforce W
2: **Output:** recommendations for a subset of deployment requests.
3: Estimate model parameters for each (strategy, deployment) pair.
4: Compute Workforce Requirement Matrix $\mathcal{W}$
5: Compute Workforce Requirement per Deployment Vector $\vec{W}$
6: Compute the objective function value f_i of each deployment request d_i
7: Sort the deployment strategies in non-increasing order of $\frac{f_i}{\vec{w_i}}$
8: Greedily add deployments until we hit d_i, such that $\Sigma_{x=1..i}\ d_i > W$
9: Pick the better of $\{d_1, d_2, d_{i-1}\}$ and d_i

Maximizing Throughput When task throughput is maximized, the objective function F is computed simply by counting the number of deployment requests that are satisfied by the `Aggregator`. Therefore, f_i, the objective function value of deployment d_i is the same for all the deployment requests and is 1. Our solution, BatchStrat-ThroughPut, sorts the deployment requests in increasing order of workforce requirement $\vec{w_i}$ to make $\frac{1}{\vec{w_j}}$ non-increasing. Other than that, the rest of the algorithm remains unchanged.

Theorem 19. *Algorithm BatchStrat-ThroughPut gives an exact solution to the problem.*

Proof. Algorithm BatchStrat-ThroughPut produces a solution by selecting a subset of deployment requests. According to the algorithm, the minimum workforce requirement of any request in this selected set is less or equal to any other requests that are not selected by BatchStrat-ThroughPut. Hence, there is no way to add a deployment request from the un-selected set to the selected one without exceeding W. This means BatchStrat-ThroughPut produces an exact solution. □

Maximizing Pay-Off Unlike throughput, when pay-off is maximized, there is an additional dimension involved that is different potentially for each deployment request. f_i for deployment request d_i is computed using $d^i.cost$, the amount of payment deployment d_i is willing to expend. Other than that, the rest of the algorithm remains unchanged.

Theorem 20. *Algorithm BatchStrat-PayOff has a 1/2-approximation factor.*

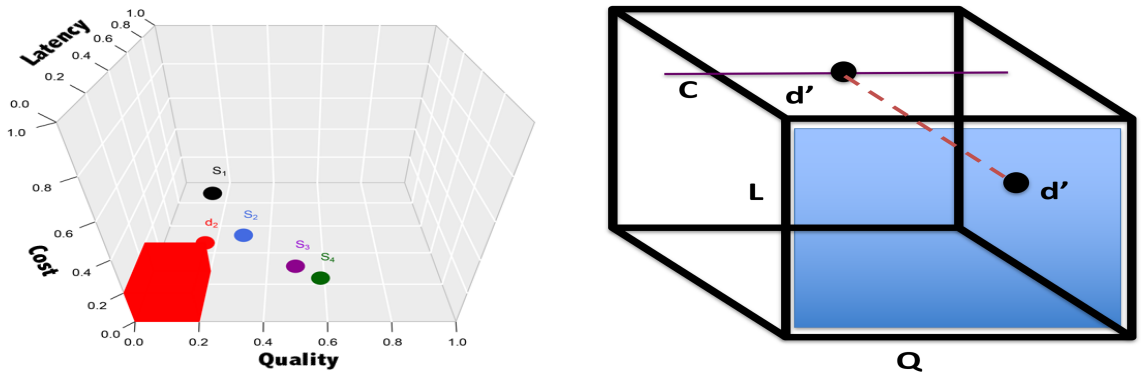
Proof. (sketch): The proof directly follows from [91]. □

5.4 ADPaR

We discuss our solution to the ADPaR problem, that takes a deployment d and strategy set $\mathcal{S}$ as inputs, and is designed to recommend alternative deployment parameters d' to optimize the goal stated in Equation 5.3 (Section 5.2.3), such that d' satisfies the cardinality constraint of d.

Going back to Example 3 with the request d_2, StratRec there is no strategy that satisfies d_2 (refer to Figure 5.4(a)).

At a high level, ADPaR bears resemblance to *Skyline and Skyband queries* [49, 80, 111] - but as we describe in Section 5.6, there are significant differences between these two problems - thus the former solutions do not adapt to solve ADPaR. Similarly, ADPaR is significantly different from existing works on query refinement [13, 71, 106, 109], that we further delineate in Section 5.6.



(a) Deployment parameters in 3-D space

(b) Projection of d' on (L, Q) plane

Figure 5.4 ADPaR.

5.4.1 Algorithm ADPaR-Exact

Our treatment is geometric and exploits the monotonicity of our objective function (Equation 5.1 in Section 5.2.3). Even though the original problem is defined in a continuous space, we present a discretized technique that is exact. ADPaR-Exact, employs three sweep-lines [31], one for each parameter, quality, cost, and latency and gradually relaxes the parameters to produce the tightest alternative parameters that admit k strategies. By its unique design choice, ADPaR-Exact is empowered to select the parameter that is most suitable to optimize the objective function, and hence, produces exact solutions to ADPaR.

ADPaR-Exact has four main steps. Before getting into those details, we present a few simplifications to the problem for the purpose of elucidation. As we have described before, we normalize quality, cost, latency thresholds of a deployment or of a strategy in $[0, 1]$, and inverse quality to $(1-quality)$. This step is just for unification, making our treatment for all three parameters uniform inside ADPaR, where smaller is better, and the deployment thresholds are considered as upper-bounds. With this, each strategy is a point in a three dimensional space and a deployment parameter (modulo its cardinality constraint) is an axis-parallel hyper-rectangle [31] in that

space. Consider Figure 5.4(a) that shows the four strategies in Example 3 and d_2 as a hyper-rectangle.

Step-1 of ADPaR-Exact computes the relaxation (increment) that a deployment requires to satisfy a strategy among each deployment parameter. This is akin to computing $s_i.cost - d_2.cost$ (likewise for quality and latency) and when the strategy cost is smaller than the deployment threshold, it shows no relaxation is needed - hence we transform that to 0. The problem is studied for quality, cost, and latency (referred to as `Q`, `C`, `L`) (Table 5.3). It also initializes $d' = \{1, 1, 1\}$, the worst possible relaxation.

Step-2 of ADPaR-Exact involves sorting the strategies based on the computed relaxation values from step-1 in an increasing order across all parameters, as well as keeping track of the index of the strategies and the parameters of the relaxation values. The sorted relaxation scores are stored in list R, the corresponding I data structure provides the strategy index, and D provides the parameter value. In other words, $R[j]$ represents the j-th smallest relaxation value, where $I[j]$ represents the index of the strategy and $D[j]$ represents the parameter value corresponding to that. A cursor r is initialized to the first position in R (Table 5.4). Another data structure, a boolean matrix M of size $|\mathcal{S}| \times 3$ (Table 5.2) is used that keeps track of the number of strategies that are covered by the current movement of cursor r in list R. This matrix is initialized to 0 and the entries are updated to 1, as r advances.

Step-3 involves designing three sweep-lines along `Q`, `C`, and `L` (Table 5.5). A sweep line is an imaginary vertical line which is swept across the plane rightwards. The `Q` sweep-line sorts the $\mathcal{S}$ in `C` `L` plane in increasing order of `Q` (the other two works in a similar fashion). ADPaR-Exact sweeps the line as it encounters strategies, in order to discretize the sweep. At the beginning, each sweep-line points the k-th strategy along `Q`, `C`, `L`, respectively. d' is updated and contains the current `Q`, `C`, `L` value i..e, $d'.quality$ = `Q`, $d'.cost$ = `C`, and $d'.latency$ = `L`. Cursor r points to the smallest

of these three values in R. Matrix M is updated to see what parameters of which strategies are covered so far.

Now, at step-4, ADPaR-Exact checks if the current d' covers k strategies or not. This involves reading through I and checking if there exists k strategies such that for each strategy $s.quality \leq d'.quality$ and $s.cost \leq d'.cost$ and $s.latency \leq d'.latency$. If there are not k such strategies, it advances r to the next position and resets $d' = \{1, 1, 1\}$ again.

If there are more than k strategies, the new d', however, does not ensure that it is the tightest one to optimize Equation 5.3. Therefore, ADPaR-Exact cannot halt. ADPaR-Exact needs to check if there exists another d'' that still covers k strategies better than d'. This can indeed happen as we are dealing with a 3-dimensional problem and these three values in combination determine the objective function.

Now, ADPaR-Exact takes turn in considering the current values of each parameter based on d', and creates a projection on the corresponding 2-D plane, for the fixed value of the third parameter. Figure 5.4(b) shows one such example in (Q, L) plane for a fixed cost value. It then considers all strategies whose $s.cost \leq d'.cost$. After that, it finds the largest expansion among the two parameters such that this new d'' covers k strategies. This gives rise to three new deployment parameters, d''_C, d''_Q, d''_L. It chooses the best of these three and updates d'. At this point it checks if M has k strategies covered. If it does, it stops processing and returns the new d' and the k strategies. If it does not, it advances the cursor r to the right.

Using Example 3, the alternative parameters are $(0.75, 0.5, 0.28)$ for d_2 and s_1, s_2, s_3 are returned.

Lemma 5. *To cover k strategies, d' needs to be initialized at least to the k^{th} smallest values on each parameter.*

Proof. (sketch) If s returned, the value of the alternative deployment parameters d' should dominate this s's parameters. Thus, to cover k strategies, d' needs to be initialized to the k^{th} value for each parameter. □

Table 5.2 Matrix M in ADPaR-Exact

	Cost	Quality	Latency
s_1	0	0	1
s_2	0	0	1
s_3	0	0	0
s_4	0	0	0

Table 5.3 Step 1 of ADPaR-Exact

	Cost	Quality	Latency
s_1	0.3	0.05	0
s_2	0.05	0.13	0
s_3	0	0.3	0
s_4	0	0.38	0

Table 5.4 Step 2 of ADPaR-Exact

Relaxation R	0	$\underline{0}$	0	0	0	0
Strategy Index I	1	$\underline{2}$	3	4	3	4
Parameter D	L	$\underline{\text{L}}$	L	L	C	C
Relaxation R	0.05	0.05	0.13	0.3	0.3	0.38
Strategy Index I	1	2	2	1	3	4
Parameter D	Q	C	Q	C	Q	Q

Lemma 6. *Going by the relaxation value and parameter order of R and D, it ensures the tightest increase in the objective function in ADPaR-Exact.*

Proof. (sketch) We omit the details for brevity but state that the aforementioned lemma could be proved through the monotonicity of the objective function in ADPaR. □

Theorem 21. *ADPaR-Exact produces an exact solution to the ADPaR problem.*

Table 5.5 Step 3 of ADPaR-Exact

sweep-line(Q)	C, L plane	0.05	0.13	0.3	0.38
	s.cost	0.3	0.05	0	0
	s.latency	0	0	0	0
sweep-line(C)	Q, L plane	0	0	0.05	0.3
	s.quality	0.38	0.3	0.13	0.05
	s.latency	0	0	0	0
sweep-line(L)	C, Q plane	0	0	0	0
	s.cost	0.3	0.05	0	0
	s.quality	0.05	0.13	0.3	0.38

Proof. (sketch) For this, we need to prove that there is no alternative deployment parameters d'' that has a smaller l_2 distance than d', returned by ADPaR-Exact, and covers at least k strategies. We prove this by contradiction.

Assume that is indeed the case. Then there is a d'' whose Euclidean distance from d is smaller than the distance between d and d'. Based on Lemma 5, it is obvious that d' must be greater or equal to the k^{th} smallest values on each parameter. When a parameter d' needs to be updated, it must be increased to that corresponding value of its closest strategy, i.e., update d' based of R and D, as described in Algorithm 8. If that was the case, then ADPaR-Exact does not make the decision to relax the next parameter optimally. However, that is not the case, based on Lemma 6. Hence the contradiction. □

Running Time: Step-1 of Algorithm ADPaR-Exact takes $O(|\mathcal{S}|)$. Step-2 and 3 are dominated by sorting time, which takes $O(|\mathcal{S}|\ log|\mathcal{S}|)$. Step-4 is the most time-consuming and takes $O(|\mathcal{S}^3|)$. Therefore, the overall running time of the algorithm is cubic to the number of strategies.

5.5 Experimental Evaluation

Our efforts in the experiments are two-fold: we perform multiple real-world deployments to estimate worker availability and demonstrate that it varies over time.

Algorithm 8 Algorithm ADPaR-Exact for alternative deployment parameter recommendation

Require: $\mathcal{S}$, k, W, d, k.

1: Compute relaxation values $s.quality - d.quality$, $s.cost - d.cost$, $s.latency - d.latency$, $\forall s \in \mathcal{S}$.
2: Compute R by sorting $3|\mathcal{S}|$ numbers in increasing order.
3: Compute I and D accordingly.
4: Initialize M to all 0's and $d' = \{1, 1, 1\}$
5: Initialize Cursor $r = R[0]$
6: Sort (`C L`) , (`Q L`), and (`Q C`) planes based on the `Q`, `C`, `L` sweep-lines respectively.
7: x= k-th value in (`C L`), y= k-th value in (`Q L`), z= k-th value in (`Q C`) plane
8: Update d'= $\{x, y, z\}$
9: r = minimum $\{x, y, z\}$
10: Update matrix M
11: **if** d' covers $\geq k$ strategies **then**
12: Compute the best d'' better than d' that covers k strategies
13: **if** M covers k strategies **then**
14: $d' = d''$ and return
15: **end if**
16: **if** M covers $< k$ strategies **then**
17: move r to the right
18: **end if**
19: **end if**
20: **if** d' covers $< k$ strategies **then**
21: Move r to the right
22: Update d''s one of the parameters by consulting R and D
23: **end if**
24: go back to line 10

In our synthetic data experiments, we present results to validate the qualitative and scalability aspects of our algorithms.

5.5.1 Real Data Experiments

We design real data experiments that involve workers from AMT focusing on text editing tasks. In particular, we consider two different types of tasks: a) sentence translation (translating from English to Hindi) and text creation (writing 4 to 5 sentences on some topic). The objective of these experiments is to validate the following questions:

1. Can worker availability be estimated and does it vary over time? To answer this question, we performed 3 different deployments for each task. The first deployment was done on the weekend (Friday 12am to Monday 12am), the second deployment was done at the beginning to the middle of the week (Monday to Thursday), the last one is from the middle of the week until the week-end (Thursday to Sunday). We design the HITs (Human Intelligence Tasks) in AMT such that each task needs to be undertaken by a maximum number of workers x. Worker availability is computed as the ratio of $\frac{x'}{x}$, where x' is the actual number of workers who undertook the task during the deployment time.

2. How does worker availability impact deployment parameters? To answer this question, we need to be able to calculate the quality, cost, and latency, along with worker availability. Latency and cost are easier to calculate, basically, it is the total amount of money that was paid to workers and the total amount of time the workers used to make edits in the document. Since text editing tasks are knowledge-intensive, to compute the quality of the crowd contributions, we ask a domain expert to judge the quality completed tasks as a percentage. Once worker availability, quality, cost, and latency are computed, we perform curve fitting that has the best fit to the series of data points.

3. How do deployment strategies impact different task types? To answer that question, we deployed both types of text editing tasks using two different deployment strategies *SEQ-IND-CRO* and *SIM-COL-CRO* that were shown to be effective with more than 70% of quality score for short texts [35]. Since our effort here was to evaluate the effectiveness of these two strategies considering quality, cost, and latency, we did not set values for deployment parameters and we simply observed them through experimentation.

Tasks and Deployment Design: We chose three popular English nursery rhymes for sentence translation. Each rhyme consisted of 4-5 lines that were to be

Original Text	Mary had a little lamb, little lamb, little lamb, Mary had a little lamb, its fleece was white as snow. Everywhere that Mary went, Mary went, Mary went, Everywhere that Mary went, the lamb was sure to go.	*Lavender's blue, dilly dilly,* *Lavender's green* *When you are king, dilly dilly,* *I shall be queen*	Rock-a-bye, baby, in the treetop When the wind blows, the cradle will rock When the bough breaks, the cradle will fall And down will come baby, cradle and all
Sequential -independent- crowd	मेरी ने एक भेड पाली, भेड पाली, भेड पाली, मेरी ने एक भेड पाली सफेद बालौ वाली. जहाँ भी मेरी जाती थी, जाती थी, जाती थी, जहाँ भी मेरी जाती थी, वो पीछे आती थी.	लैवेंडर की नीली, गहरी नीली, लैवेंडर का हरा जब आप राजा होते हैं, तो आप बहुत खुश होते हैं। मैं रानी बनूंगी	रॉक-ए-बाय, बेबी, ट्रीटोप में जब हवा चलेगी, तो खड़खड़ाहट उठेगी जब कड़ा फटेगा तो खटिया गिर जाएगी और नीचे आएगा बच्चा, पालना और सब
simultaneous - collaborative- crowd	मैरी के पास एक छोटा सा मेमना था, थोड़ा सा मेमना, थोड़ा सा भेड़ का बच्चा, मैरी के पास थोड़ा सा मेमना था, उसका ऊन बर्फ की तरह सफेद था। हर जगह मैरी चली गई, मैरी चली गई, मैरी चली गई, हर जगह मैरी चली गई, मेमने का जाना निश्चित था	लैवेंडर की नीली, गहरी नीली, लैवेंडर का हरा जब आप राजा होते हैं, तो आप बहुत खुश होते हैं। मैं रानी बनूंगी	रॉक-ए-बाय, बेबी, ट्रीटोप में जब हवा चलेगी, तो खड़खड़ाहट उठेगी जब कड़ा फटेगा तो खटिया गिर जाएगी और नीचे आएगा बच्चा, पालना और सब

Figure 5.5 Translation: original texts and translation.

translated from English to Hindi (one such sample rhyme is shown in Figure 5.5). For text creation, we considered three popular topics, *Robert Mueller Report*, *Notre Dame Cathedral*, and *2019 Pulitzer prizes*. One sample text creation is shown in Figure 5.6.

As mentioned above, we designed three deployment windows at different days of the week. We note that unlike typical micro-tasks in AMT, text editing tasks are more complex, required significantly more time to complete (as described later, we allocated 2 hours per HIT) than a typical microtask involving only a few clicks. Therefore, a HIT in this experiment contained either three sentence translation tasks or three text creation tasks. As opposed to micro-tasks, where a HIT may contain tens of tasks, designing HITS with a smaller number of tasks (e.g., 3) is a common practice for complex tasks. For each task type, we validated 2 deployment strategies - in *SEQ-IND-CRO*, workers were to work in sequence and independently, whereas, in *SIM-COL-CRO*, workers were asked to work simultaneously and collaboratively. We created two different samples of the same study resulting in a total of eight HITs

Strategy	*TOPIC*	*TEXT*
Sequential - independent-crowd	*Robert Mueller report*	The Mueller Report, formally titled the Report on the Investigation into Russian Interference in the 2016 Presidential Election, i*s the official report documenting the findings of the Special Counsel investigation, led by Robert Mueller, into Russian efforts to interfere in the 2016 United States presidential election, allegations of conspiracy or coordination between Donald Trump's presidential campaign and Russia, and allegations of obstruction of justice. The report was submitted to Attorney General William Barr on March 22, 2019. This report addressed obstruction of justice, stating it "does not conclude that the President committed a crime, [and] it also does not exonerate him".*
simultaneous - collaborative-crowd	*Robert Mueller report*	*It was a report related to United States counterintelligence investigation of the Russian government's efforts to interfere in the 2016 presidential election. As of April 2019, thirty-four individuals were indicted by Special Counsel investigators. Eight have pled guilty to or been convicted of felonies, including at least five Trump associates and campaign officials. The report concluded that Russian interference in the 2016 presidential election did occur and "violated U.S. criminal law."*

Figure 5.6 Text Creation: two samples on Robert Mueller report.

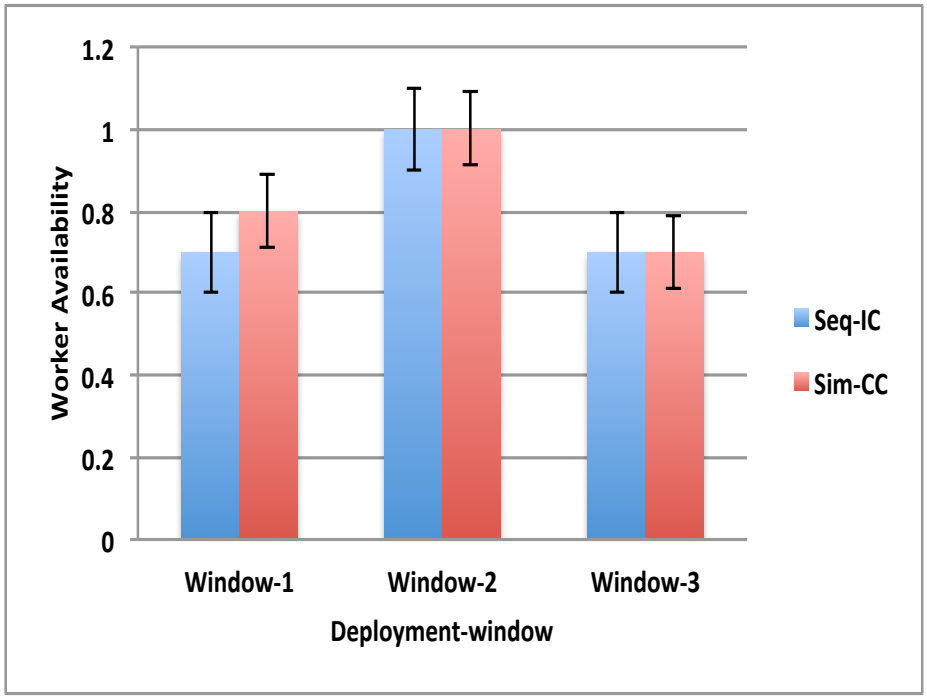


Figure 5.7 Worker availability estimation.

deployed inside the same window. Each HIT was asked to be completed by 10 workers paid $2 each if the worker spent enough time (more than 10 minutes). This way, a total of 80 unique workers were hired for each deployment window, and a total of 240 workers were hired for all three deployments.

Worker Recruitment: For both task types, we recruited workers with a HIT approval rate greater than 90%. For sentence translation, we additionally filtered workers on geographic locations, either US or India. For text creation tasks, we recruited US-based workers with a Bachelor's degree.

Enabling collaboration: After workers were recruited from AMT, they were directed to Google Docs where the tasks were described and the workers were given instructions. The docs were set up in editing mode, so edits could be monitored.

Experiment Design: An experiment is comprised of three steps. In Step-1, all initially recruited workers went through qualification tests. For text creation, a topic (Royal Wedding) was provided and the workers were asked to write 5 sentences related to that topic. For sentence translation, the qualification test comprised of 5 sample sentences to be translated from English to Hindi. Completed qualification tests were evaluated by domain experts and workers with more than 80% or more qualification scores were retained and invited to work on the actual HITs. In Step-2, actual HITs were deployed for 72 hours and the workers were allotted 2 hours for the tasks. In Step-3, after 72 hours of deployment, results were aggregated by domain experts to obtain a quality score. Cost and latency were easier to calculate directly from the raw data.

Summary of Results: Our first observation is that worker availability *can be estimated and does vary over time* (standard error bars added). We observed that for both task types, workers were more available during Window 2 (Monday-Thursday), compared to the other two windows. Detailed results are shown in Figure 5.7.

Our second observation is that each of the deployment parameters (quality, cost, and latency) does have a linear relationship with worker availability for text editing tasks. The first two increases linearly with worker availability, and the latency decreases with increasing worker availability. This linear relationship could be captured and the parameters (α, β) could be estimated. Table 5.6 presents these results and the estimated (α, β) always lie within 90% confidence interval of the fitted line.

Our final observation is that *SEQ-IND-CRO* performs better than *SIM-COL-CRO* for both task types. However, this difference is not statistically significant. On the other hand, *SEQ-IND-CRO* is longer to complete (higher latency). Upon further analysis, we observe that when workers are asked to collaborate and edit simultaneously, that gives rise to an edit war and an overall poor quality. Figure 5.8 presents these results.

5.5.2 Synthetic Experiments

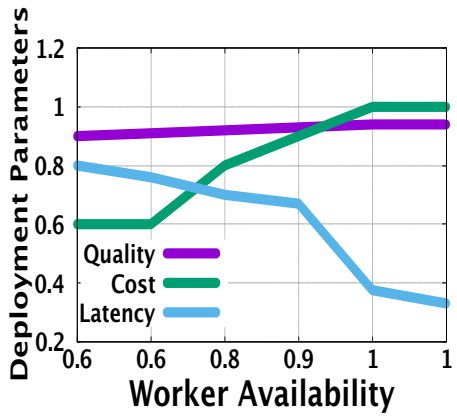
We aim to evaluate the qualitative guarantees and the scalability. Algorithms are implemented in Python 3.6 on Ubuntu 18.10. Intel Core i9 3.6 GHz CPU, 16GB of memory.

Implemented Algorithms We describe different algorithms that are implemented.

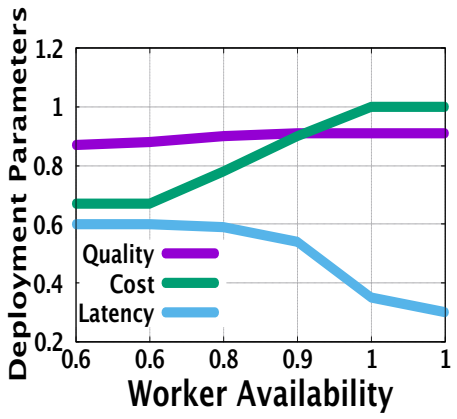
Batch Deployment Algorithms `Brute Force:` An exhaustive algorithm which compares all possible combinations of deployment requests and returns the one that optimizes the objective function.

`BaselineG:` This algorithm sorts the deployment requests in decreasing order of $\frac{f_i}{\vec{w_i}}$ and greedily selects requests until worker availability W is exhausted.

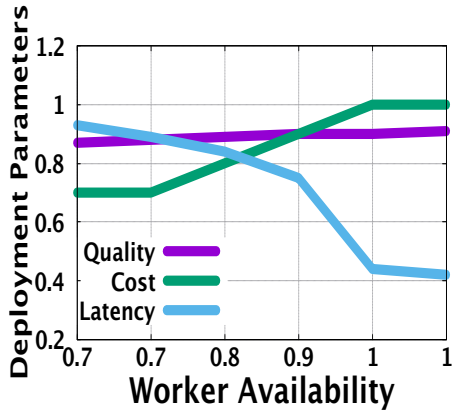
BatchStrat: Our proposed solution described in Section 5.3.



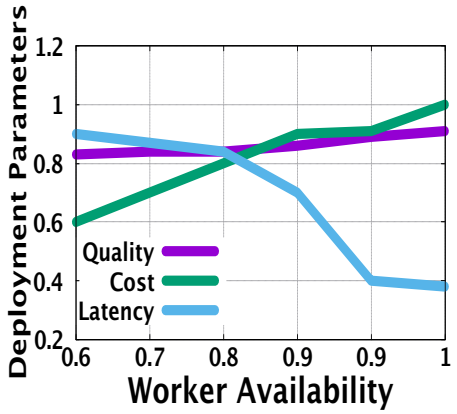
(a) Translation *SEQ-IND-CRO*



(b) Translation *SIM-COL-CRO*



(c) Creation *SEQ-IND-CRO*



(d) Creation *SIM-COL-CRO*

Figure 5.8 Relationship between deployment parameters and worker availability.

Table 5.6 α, β Estimation in Real Experiments

Worker Availability and Deployment Parameters		
Task-Strategy	Parameters	α,β
Translation *SEQ-IND-CRO*	Quality	0.09, 0.85
	Cost	$1.00, 0.00$
	Latency	$-0.98, 1.40$
Translation *SIM-COL-CRO*	Quality	$0.09, 0.82$
	Cost	$0.82, 0.17$
	Latency	$-0.63, 1.01$
Creation *SEQ-IND-CRO*	Quality	$0.10, 0.80$
	Cost	$1.00, 0.00$
	Latency	$-1.56, 2.04$
Creation *SIM-COL-CRO*	Quality	$0.19, 0.70$
	Cost	$1.00, -0.00$
	Latency	$-1.38, 1.81$

ADPaR Algorithms `ADPaRB`: This is a brute force algorithm that examines all sets of strategies of size k. It returns the one that has the smallest distance to the task designer's original deployment parameters. While it returns the exact answer, this algorithm takes exponential time to run.

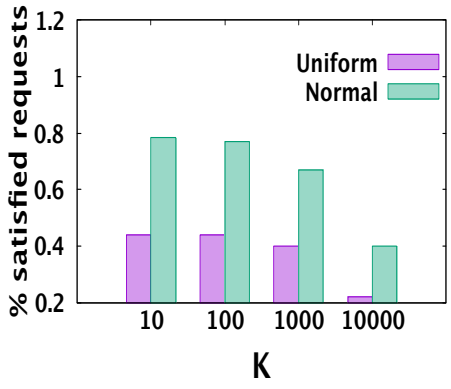
`Baseline2`: This baseline algorithm is inspired by a related work [106]. The main difference though, the related work modifies the original deployment request by just one parameter at a time and is not optimization driven. In contrast, ADPaR-Exact returns an alternative deployment request, where multiple parameters may have to be modified.

`Baseline3`: This one is designed by modifying space partitioning data structure R-Tree [30]. We treat each strategy parameters as a point in a 3-D space and index them using an R-Tree. Then, it scans the tree to find if there is a minimum bounding box (MBB) that exactly contains k strategies. If so, it returns the top-right corner of that MBB as the alternative deployment parameters and corresponding k strategies. If such an MBB does not exist, it will return the top right corner of another MBB that has at least k strategies and will randomly return k strategies from there.
ADPaR-Exact: Our proposed solution in Section 5.4.

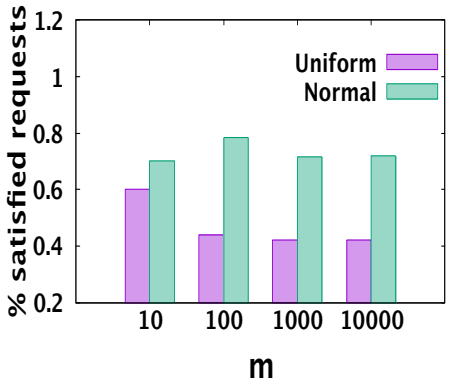
Summary of Results: Our simulation experiments highlight the following findings: **Observation 1:** Our solution BatchStrat returns exact answers for throughput optimization, and the approximation factor for pay-off maximization is always above 90%, significantly surpassing its theoretical approximation factor of 1/2. **Observation 2:** Our solution BatchStrat is highly scalable and takes less than a second to handle millions of strategies, and hundreds of deployment requests, and k. **Observation 3:** Our algorithm ADPaR-Exact returns exact solutions to the ADPaR problem, and significantly outperforms the two baseline solutions in objective function value. **Observation 4:** ADPaR-Exact is scalable and takes a few seconds to return alternative deployment parameters, even when the total number of strategies is large and k is sizable.

Quality Experiment

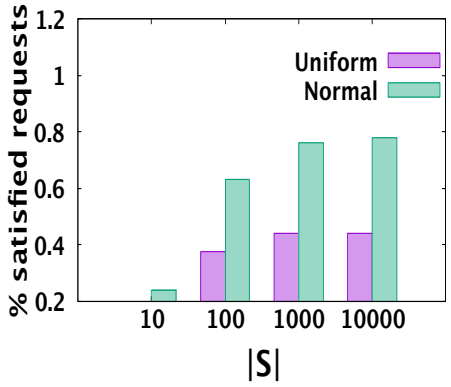
Batch Deployment Recommendation. Goal: We validate the following two aspects: (i) *how many deployment requests* BatchStrat *can satisfy without invoking* ADPaR? (ii) *How does* BatchStrat *fare to optimize different platform-centric goals?* We compare BatchStrat with the other two baselines, as appropriate.



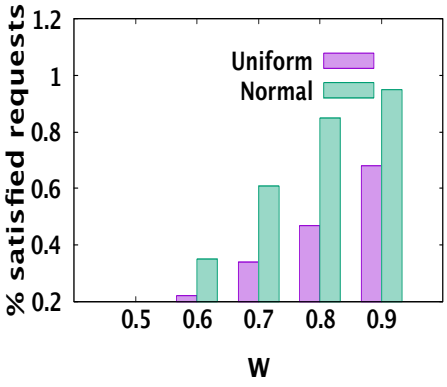
(a) Varying k



(b) Varying m



(c) Varying $\mathcal{S}$



(d) Varying W

Figure 5.9 Percentage of satisfied requests before invoking ADPaR.

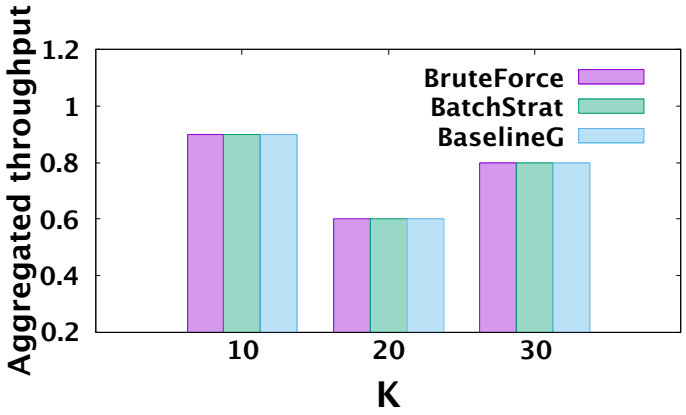
Strategy Generation: The dimension values of a strategy are generated considering uniform and normal distributions. For the normal distribution, the mean and standard deviation are set to 0.75 and 0.1, respectively. We randomly pick the value from 0.5 to 1 for the uniform distribution.

Worker Availability: For a strategy, we generate α uniformly from an interval $[0.5, 1]$. Then, we set $\beta = 1 - \alpha$ to make sure that the estimated worker availability W is within $[0, 1]$. These numbers are generated in consistence with our real data experiments.

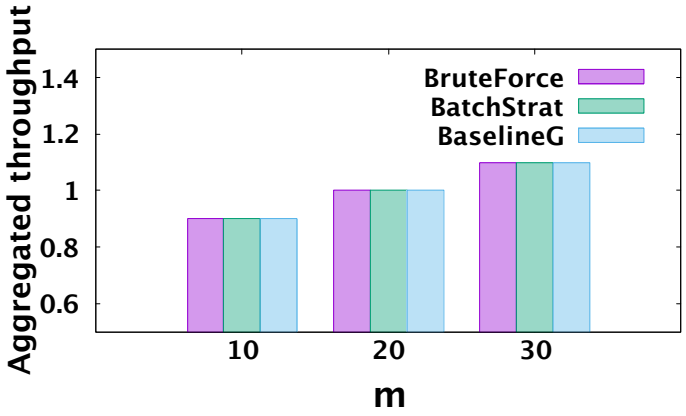
Deployment Parameters: Once W is estimated, the quality, latency, and cost - i.e., the deployment parameters, are generated in the interval $[0.625, 1]$. For each experiment, 10 deployment parameters are generated, and an average of 10 runs is presented in the results.

Figure 5.9 shows the percentage of satisfied requests by BatchStrat with varying k, m, $|\mathcal{S}|$, W. In general, normal distribution performs better than uniform. Upon further analysis, we realize that normal distribution has a very small standard deviation, and is thereby able to satisfy more requests. As shown in Figure 5.9(a), the percentage of satisfied requests decreases with increasing k, which is expected. Contrarily, the effect of increasing batch size m is less pronounced. This is because all requests use the same underlying distribution, allowing BatchStrat to handle more of them. With more strategies $|\mathcal{S}|$, as Figure 5.9(c) illustrates, BatchStrat satisfies more requests, which is natural, because with increasing $|\mathcal{S}|$, it simply has more choices. Finally, in Figure 5.9(d), with higher worker availability BatchStrat satisfies more requests. By default, we set $|\mathcal{S}| = 10000, m = 10, k = 10, W = 0.5$.

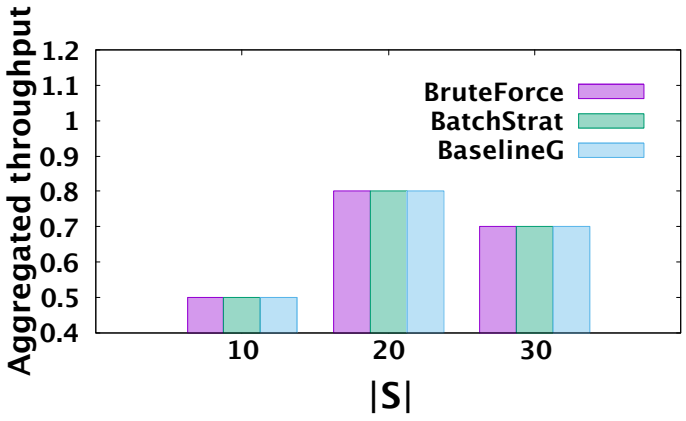
Figure 5.10 shows the results of throughput of BatchStrat by varying k, m,$|\mathcal{S}|$, compare with the two baselines. Figure 5.11 shows the approximation factor of BatchStrat and `BaselineG`. BatchStrat achieves an approximation factor of 0.9 most of



(a) Varying k

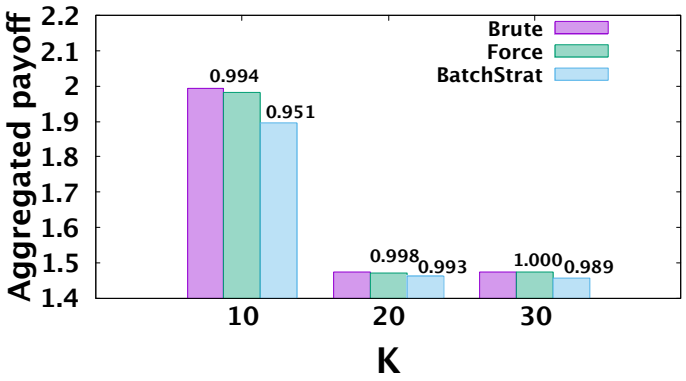


(b) Varying m

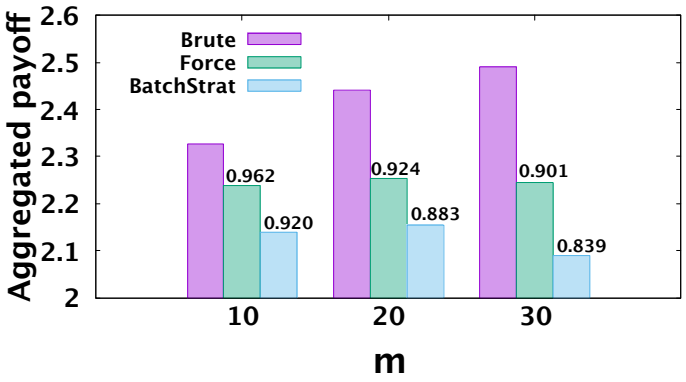


(c) Varying $\mathcal{S}$

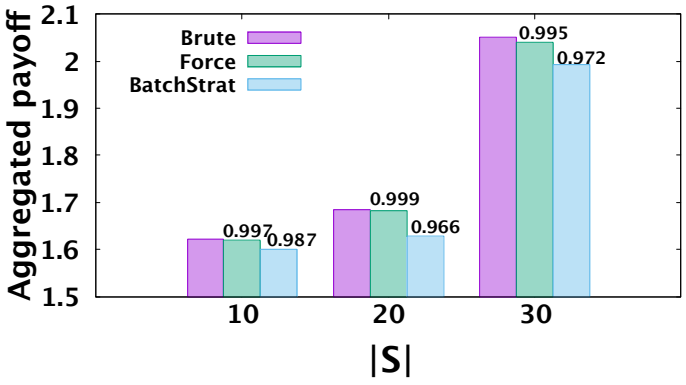
Figure 5.10 Objective function for throughput.



(a) Varying k

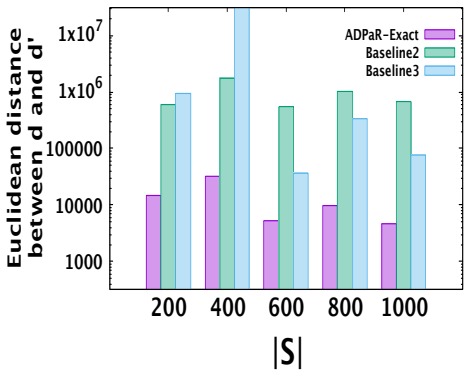


(b) Varying m

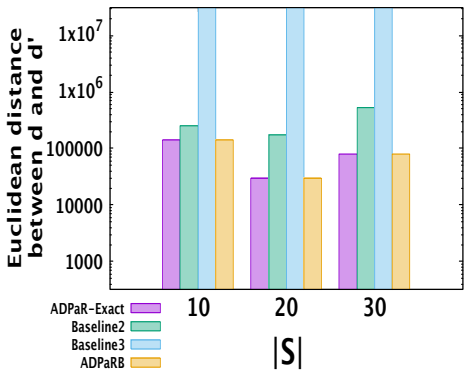


(c) Varying $\mathcal{S}$

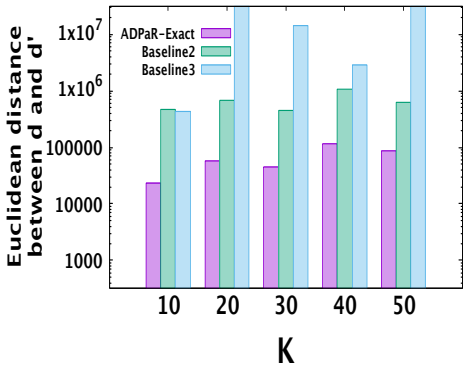
Figure 5.11 Objective function and approximation factor for payoff.



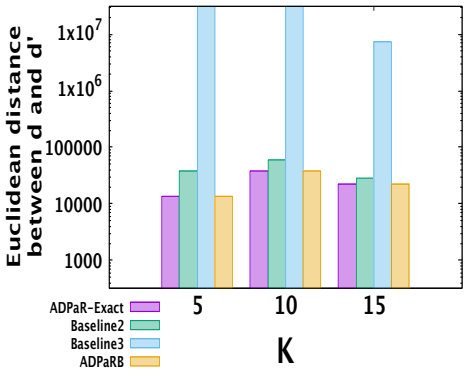
(a) without Brute Force



(b) with Brute Force

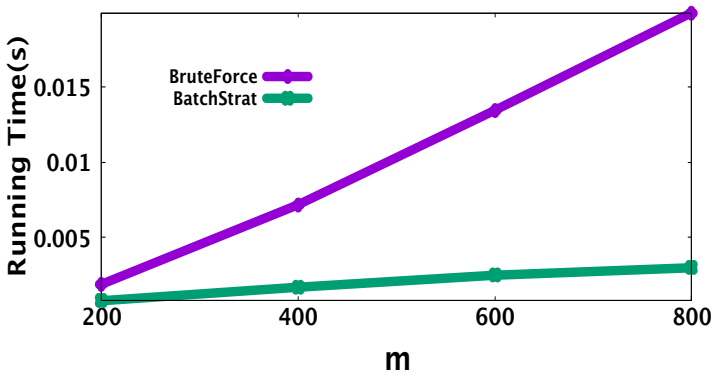


(c) without Brute Force

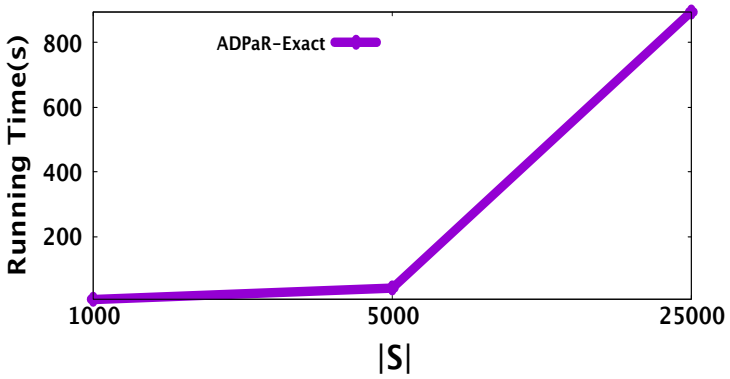


(d) with Brute Force

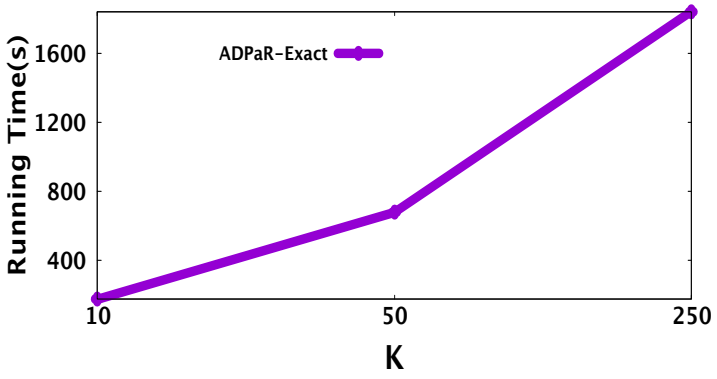
Figure 5.12 Quality experiments for ADPaR.



(a) Batch Deployment Varying m



(b) ADPaR Varying $|\mathcal{S}|$



(c) ADPaR Varying k

Figure 5.13 Results of scalability experiments.

the time. For both experiments, the default values are $k = 10, m = 5, |\mathcal{S}| = 30, W = 0.5$ because brute force does not scale beyond that.

Alternative Deployment Recommendation ADPaR. The goal here is to measure the objective function. Since `ADPaRB` takes exponential time, to be able to compare with this, we set $|\mathcal{S}| = 20$, $k = 5, W = 0.5$ for all the quality experiments that has to compare with the brute force. Otherwise, the default values are $|\mathcal{S}| = 200$, $k = 5$.

In Figure 5.12, we vary $|\mathcal{S}|$ and k and plot the Euclidean distance between d and d' (smaller is better). Indeed, ADPaR-Exact returns exact solution always. The other two baselines perform significantly worse, while `Baseline 3` is the worst. That is indeed expected, because these two baselines are not optimization guided, and does not satisfy our goal. Naturally, the objective function decreases with increasing $|\mathcal{S}|$, because more strategies mean smaller change in d', making the distance between d and d' smaller. As the results depict, optimal Euclidean distance between d and d' increases with increasing k, which is also intuitive, because, with higher k value, the alternative deployment parameters are likely to have more distance from the original ones.

Scalability Experiments Our goal is to evaluate the running time of our proposed solutions. Running time is measured in seconds. We present a subset of results that are representative.

Batch Deployment Recommendation Since the `BaselineG` has the same running time as that of BatchStrat (although qualitatively inferior), we only compare the running time between `Brute Force` and BatchStrat. The default setting for $|\mathcal{S}|$, k and W are 30, 10 and 0.75, respectively.

The first observation we make is, clearly BatchStrat can handle millions of strategies, several hundreds of batches, and very large k and still takes only a few

fractions of seconds to run. It is easy to notice that the running time of this problem only relies on the size of the batch m (or the number of deployment requests), and not on k or $\mathcal{S}$. As we can see in Figure 5.13(a), `Brute Force` takes exponential time with increasing m, whereas BatchStrat scales linearly.

Alternative Deployment Recommendation We vary k and $|\mathcal{S}|$ with defaults set to 5 and 10000 respectively, and evaluate the running time of ADPaR-Exact. W is set to 0.5. As Figures 5.13(b) and 5.13(c) attest, albeit non-linear, ADPaR-Exact scales well with k and $|\mathcal{S}|$. We do not present the baselines as they are significantly inferior in quality.

5.6 Related Work

We discuss three different areas related to our work.

Crowdsourcing Deployment: A very few related works [9, 149] have started to study the importance of appropriate deployment strategies but these works are empirical and do not propose algorithmic solutions.

Skyline and Skyband Queries: Skyline queries playe an essential role in computing favored answers from a database [36, 49]. Based on the concepts of skylines, other classes of queries arise, especially top-k queries and k-skyband problems which aim to bring more useful information than original skylines. Mouratidis et al. [110, 111] study several related problems. In [110], sliding windows are used to track the records in dynamic stream rates. In [111], a geometry arrangement method is proposed for top-k queries with uncertain scoring functions. Because our problem seeks the optimal group of k strategies, it is similar to the top-k queries problem. However, unlike Skyband or any other related work, ADPaR recommends alternative deployment parameters. Thus, these solutions do not extend to solve ADPaR.

Query Refinement: Query reformulation has been widely studied in Information Retrieval [71]. In [106], authors take users' preference into account and propose an interactive method for seeking an alternative query which could satisfy a specific cardinality constraint. This is different from ADPaR since it only relaxes one dimension at a time. Aris et al. [13] proposed a graph modification method to recommend queries that maximize an overall utility. Mottin et al. [109] develop an optimization driven framework but the solutions can only handle Boolean/categorical data. Thus, none of these extend to solving ADPaR.

5.7 Discussion

We discuss two immediate extension of StratRec that are part of our ongoing investigation.

Worker availability as pdf: Instead of an expected value, one possible extension is to use the actual pdf and revisit the problems in StratRec. This will require reformulating `Batch Deployment Recommendation` and ADPaR. For example, the goal of the former problem would be to optimize F in an expected sense, considering W as a pdf. Similarly, ADPaR will produce k different strategies for every possible worker availability value considering its corresponding probability. We note that the current framework stays unaltered under this extension, only the designed algorithms are likely to exhibit higher computational complexity.

Non-linear relationship between worker availability and parameters of a strategy: BatchStrat would need to be adapted for the case where the relationship between worker availability and the parameters of a strategy is non-linear. We note that theoretical guarantees of our designed solutions (BatchStrat and ADPaR-Exact) as well as their running time hold on all the polynomial relations with degree less than five, because these functions do have analytical solutions for finding zero points. As for functions with higher degrees, ADPaR-Exact will remain unaltered. The only

change will be for the Workforce Requirement Computation in Section 5.3.2, that will rely on the Newton-Raphson method, and hence its running time will increase.

5.8 Conclusion

We propose StratRec, an optimization-driven middle layer designed to recommend strategies to multiple deployment requests or recommend alternative deployment parameters if a request cannot be satisfied as formulated. Our work addresses multi-faceted modeling challenges through the generic design of modules in StratRec that could be instantiated to optimize different types of goals by accounting for worker availability. We develop computationally-efficient algorithms. We present extensive real data experiments through multiple deployments in AMT and conduct synthetic experiments to validate the qualitative and scalability aspects of StratRec. Our future investigations are summarized in Section 5.7.

CHAPTER 6

RANK AGGREGATION WITH PROPORTIONATE FAIRNESS

6.1 Introduction

Ranking is a commonly used method to prioritize desirable outcomes among a set of candidates and is an essential step in many high impact applications, such as, hiring candidates for a job, selecting students for school and college admission or scholarship, finding winning candidates in a competition, or approving loans. Traditionally, producing the final ranking involves aggregating potentially conflicting preferences from multiple individuals and is a central problem in the areas of voting and social choice theory, which is traditionally known as the rank aggregation problem [8,59,136]. Our goal in this work is to revisit the rank aggregation problem considering a notion of fairness, namely *proportionate fairness or p-fairness* [26, 134] that ensures proportionate representation of every group based on a protected attribute in every position of the aggregated ranked order. P-fairness has been studied in the theory community to enable resource allocation satisfying temporal fairness or proportionate progress. The classical problem in this context is known as the *Chairman Assignment Problem* [19, 134] which studies how to select a chairman of a union every year from a set of r states such that that at any time the accumulated number of chairmen from each state is proportional to its weight. We formalize the *rank aggregation subject to p-fairness* or **RAPF** to that end.

RAPF is defined formally as follows: m conflicting rankings are given over a database of n candidates, where candidates have a protected attribute A with ℓ associated values (defined, e.g., over seniority level, ethnicity, or gender). Let $f(p)$ denote the fraction of candidates with protected attribute value p, that is, $f(p) = \frac{1}{n}\sum_{v \in V} \mathbf{1}_{A(v)=p}$. The goal is to find an aggregated ranking such that the total number of disagreements between the aggregated ranking and each of the

individual m rankings is minimized, and for every protected attribute value p and every position k in the aggregated ranking, the representation of the candidates with protected attribute value p in the top k candidates is proportional to $f(p)$. P-fairness is desirable in several compelling rank aggregation applications, such as, French process of admitting students to university (Parcoursup), matching medical students to US hospitals for residency, or faculty hiring in the universities, to name a few. Section 6.1.1 describes one such application in depth.

We initiate this investigation by studying the **Individual p-Fairness** or **IPF** problem that finds a closest p-fair ranking to an individual ranking, which we believe is an important problem in its own merit. A similar problem is studied in the past [72] with weaker notion of fairness and the designed solutions are just heuristic. We investigate how a solution designed for **IPF** could solve **RAPF**.

6.1.1 Motivation

Running Example: p-fairness in faculty hiring. Consider a toy database of n (12) applicants who are interviewed to be hired for a small number of faculty positions in a university. The hiring committee comprises of a set of m (4) members, each of whom ranks these n candidates (refer to Table 6.1) based on their credentials and interview performance. After that, these individual ranks are to be aggregated to create an overall order based on which the candidates would be made job offers until the positions are filled. Potential protected attributes of the candidates are seniority level, research areas, and gender. As an example, considering seniority level, 3 applicants are junior, 4 are mid-career, and 5 are senior, making the proportion over seniority level to be 3/12, 4/12, and 5/12, respectively.

The goal of **RAPF** is to produce a ranked order over the 12 candidates by aggregating all 4 ranked lists such that the produced order is closest to the individual 4 ranks and *for each of the* 12 *positions and for each of the values of a particular*

protected attribute the candidates appear proportionate to their representation in the original data. Indeed, it is *important to ensure fairness in each of the* 12 *positions considering the given protected attribute - otherwise, depending on who accepts/declines the job offer, the proportionate representation of the candidates based on the underlying protected attribute would get disrupted.* Intuitively speaking, assuming seniority level as the protected attribute, a solution designed for **RAPF** must ensure that the representation of junior, mid-career, and senior candidates is (0.75, 1, 1.25) up to integral rounding in the top 3 positions, (1, 1.33, 1.67) up to integral rounding in the top 4 positions, and so on.

Table 6.1 Original Ranks Provided by four Members

Candidate Name	**Gender**	**Seniority level**	**Area**	**Mem 1**	**Mem 2**	**Mem 3**	**Mem 4**
Molly	Female	Junior	DB	1	3	4	6
Amy	Female	Junior	DB	2	2	1	5
Abigail	Female	Junior	AI	3	5	2	7
Kim	Male	Mid career	HCI	4	7	3	8
Lee	Male	Mid career	Theory	5	9	6	1
Park	Male	Mid career	Vision	6	1	5	2
Kabir	Male	Mid career	NLP	7	4	8	3
Damien	Male	Senior	ML	8	6	7	4
Andres	Male	Senior	Security	9	8	10	9
Aaliyah	Female	Senior	Systems	10	10	9	10
Kiara	Female	Senior	DM	11	11	12	11
Jazmine	Female	Senior	PL	12	12	11	12

We acknowledge that the existing popular group based fairness definition statistical parity *[58] is somewhat similar to p-fairness, however, the best adapted version of top-k statistical parity studied in a recent paper [87] does not account for proportionate representation in every position of the top-k, limiting its applicability. Section 6 contains further details.*

6.1.2 Contributions

Our first contribution is to formalize two optimization problems, **Individual p-Fairness** or **IPF** and the rank aggregation problem subject to proportionate fairness (**RAPF**) **(Section 6.2)** considering binary ($\ell = 2$) and multi-valued ($\ell > 2$) protected attributes.

Our second contribution is theoretical and algorithmic **(Sections 6.4, 6.5)**. For the **IPF** problem, we present an efficient greedy solution GrBinaryIPF for a binary protected attribute that runs in $O(n)$ time. For a multi-valued protected attribute, we prove that the proposed algorithms studied in a recent work [72] for **IPF** are heuristics and do not ensure optimality (refer to Section 6.3.1 for details). In fact, we claim that solving **IPF** for multi-valued protected attribute is non-trivial. We present two solutions for multi-valued **IPF** - a dynamic programming based exact algorithm ExactMultiValuedIPF that takes linear time when the number of values on the protected attribute is a constant, and ApproxMultiValuedIPF based on a minimum weight matching on convex bipartite graphs [39], that admits a 2 approximation factor.

Since rank aggregation problem under Kemeny Optimization is NP-hard for 4 or more lists [8, 59, 136], **RAPF** is also NP-hard. In Section 6.5, we present two algorithmic frameworks RandAlgRAPF and AlgRAPF for **RAPF**, one is randomized and the other one is deterministic that admit provable approximation factors. Both frameworks are scalable while the randomized one is highly scalable but because of its randomized nature, its approximation factor is expressed in expectation. Both algorithmic frameworks use as subroutine the solutions of **IPF**. They also leverage on variants of the Pick-A-Perm algorithm [8, 59, 136] that is widely used in the classical rank aggregation context. We then prove that the approximation factor of the solution designed for **RAPF** is 2+ the approximation factor of the **IPF** algorithm used as subroutine. This implies that multi-valued **RAPF** with

Table 6.2 Summary of Technical Results

Problem	Protected Attribute	Hardness	Algorithm	Approx Factor	Running Time
IPF	binary	p-time	GrBinaryIPF	exact	$O(n)$
	multi-valued	open	ExactMultiValuedIPF	exact	$O(n\ell 2^{\ell})$
			ApproxMultiValuedIPF	2	$O(n^{2.5}\log n)$
RAPF	binary	NP-hard	RandAlgRAPF+GrBinaryIPF	2	$O(n)$
			AlgRAPF+ GrBinaryIPF	2	$O(m^2 n \log n)$
	multi-valued	NP-hard	RandAlgRAPF+ ExactMultiValuedIPF	3	$O(n\ell 2^{\ell})$
			RandAlgRAPF+ ApproxMultiValuedIPF	4	$O(n^{2.5}\log n)$
			AlgRAPF+ ExactMultiValuedIPF	3	$O(m^2 n \log n + mn\ell 2^{\ell})$
			AlgRAPF+ ApproxMultiValuedIPF	4	$O(m^2 n \log n + mn^{2.5}\log n)$

ExactMultiValuedIPF admits a 3 approximation factor; whereas, it admits a 4 approximation factor when ApproxMultiValuedIPF is used instead. Table 6.2 summarizes our theoretical results.

Our third contribution is experimental (Section 6.6). We run extensive experiments using 3 real world and a large scale synthetic datasets, and compare an implementation of our solution with the implementation of two state-of-the-art solutions DetConstSort [72] for **IPF** and FairILP [87] for **RAPF**. Our first and foremost observation is that, consistent with our theoretical analysis, p-fairness promotes stronger notion of fairness, by ensuring proportionate representation of each of the protected attribute values for every position in the aggregated ranked order. Our experimental results demonstrate that our proposed model and solutions satisfy the fairness criteria proposed in state-of-the-art solutions [72, 87] - however, existing solutions do not extend to satisfy p-fairness. Our experimental results corroborate our theoretical results in terms of approximation factors and demonstrate that our solutions are highly scalable to large number of items and ranks.

Table 6.3 Important Notations

Notation	Meaning
A	Protectedattribute
ℓ	Number ofdifferent values in A
$f(p)$	proportion of candidates with attribute value p
$\sigma(u)$	position of item u in rank σ

6.2 Preliminaries and Formalism

Database. contains n items or candidates. These two terms will be used interchangeably in the chapter. Using the running example, $n = 12$. The set of items will be denoted V, individual items will be denoted by u and v.

Rank. We consider rankings of the items in V. Each such ranking can be viewed as a permutation. We will use the terms ranking and permutation interchangeably.

Multiple Rankings. The input consists m different complete rankings. Using the running example, $m = 4$.

Protected Attribute. Each item/candidate $v \in V$ has a *protected attribute* $A(v)$ that can take any of ℓ different values. As an example, seniority level is a multi-valued protected attribute with three possible values Junior, Mid career, Senior - thus $\ell = 3$. Contrarily, gender is a binary protected attribute with two values male and female, and $\ell = 2$.

Rank Aggregation Measures [8, 59]. In this work we consider two popular rank distance measures Kendall-Tau distance and Spearman's footrule distance.

Definition 1. Kendall-Tau distance. *Given two permutations $\sigma, \eta : V \to [1..n]$, the Kendall-Tau distance between the two permutations is the sum of pairwise*

disagreements between σ and η (bubble-sort distance).

$$\mathcal{K}(\sigma, \eta) = \sum_{\{u,v\} \subseteq V} \mathbf{1}_{(\sigma(v)-\sigma(u))(\eta(v)-\eta(u))<0}$$

Note that the Kendall-Tau distance is symmetric, that is, $\mathcal{K}(\sigma, \eta) = \mathcal{K}(\eta, \sigma)$. It also satisfies the triangle inequality, for any three permutations σ, μ, η we have $\mathcal{K}(\sigma, \mu) + \mathcal{K}(\mu, \eta) \geq \mathcal{K}(\sigma, \eta)$.

Definition 2. Spearman's footrule distance. *Given two permutations $\sigma, \eta : V \to [1..n]$, the Spearman's footrule distance between the two permutations is the sum of the absolute values (ℓ_1 distance) of the differences between two permutations.*

$$\mathcal{S}(\sigma, \eta) = \sum_{u \in V} |(\sigma(u) - \eta(u)|$$

Using the running example, the Kendall-Tau distance between the rankings of Member 1 and Member 2 is 12 because there are 12 pairs of items that appear in opposite order in these two rankings. Spearman's footrule distance between them is 22, which is the sum of the absolute values of the difference in the order between these two rankings.

Relationship between the two measures. Diaconis and Graham [56] proved that for any two permutations the Spearman's footrule distance is at least the Kendall-Tau distance between them, and at most twice the Kendall-Tau distance. That is, for any two permutations σ, η, we have $\mathcal{K}(\sigma, \eta) \leq \mathcal{S}(\sigma, \eta) \leq 2\mathcal{K}(\sigma, \eta)$.

In the rest of the chapter, we focus on Kendall-Tau distance and when we refer to Spearman's footrule distance we will state it explicitly. The Kemeny distance between a single ranking and multiple rankings is based on Kendall-Tau distance.

Definition 3. Kemeny Distance. *For rankings $\rho_1, \rho_2, \ldots, \rho_m$ the* Kemeny Distance *of the ranking σ to these rankings is*

$$\kappa(\sigma, \rho_1, \rho_2, \ldots, \rho_m) = \sum_{i=1}^{m} \mathcal{K}(\sigma, \rho_i)$$

Using the running example, Kemeny Distance between each of the aggregated rankings presented in the three columns of Table 6.4 and the individual member ranks are $34, 34$*, and* 46*, respectively.*

We note that Kemeny distance which is based on Kendall-Tau distance is the most popular and accepted measure for quantifying the quality of rank aggregation and has been widely used in the related work on rank aggregation [7, 8, 58]. The Kemeny distance measure has a maximum likelihood interpretation and it is the only known measure that simultaneously satisfies: neutrality, consistency, and the (extended) Condorcet property. Moreover, Kendall-Tau/Kemeny has also been adopted in the only previously known fair rank aggregation FairILP [87] work. Other distance measures are briefly described in Section 6.3.

Definition 4. Proportionate Fair or p-fair ranking [26,134]. *For any protected attribute value* p*, let* $f(p)$ *denote the fraction of items with this value, that is,* $f(p) = \frac{1}{n}\sum_{v \in V} \mathbf{1}_{A(v)=p}$*. A ranking* σ *is* proportionate fair *or* p-fair *if for every* $k \in [1..n]$*, the number of items with protected attribute value* p *among the* k *top ranked items in* σ *is either* $\lfloor f(p) \cdot k \rfloor$ *or* $\lceil f(p) \cdot k \rceil$*.*

Using the running example, if gender is the protected attribute with 50% representation of male and female, then p-fairness implies 1 male and 1 female in the top-2 items, 2 males and 2 females in the top-4 items , and so on. (Note that for any odd k the difference between the number of males and females in the top-k is exactly 1.) We refer to the 3rd column of Table 6.4 and note that p-fairness is satisfied.

Definition 5. Relaxed p-fair ranking. *Given an integer input* $\delta \geq 0$*, a ranking* σ *is* relaxed proportionate fair *or* relaxed p-fair *if for every* $k \in [1..n]$*, the number of items with protected attribute value* p *among the* k *top ranked items in* σ *is between* $\lfloor f(p) \cdot k \rfloor - \delta$ *and* $\lceil f(p) \cdot k \rceil + \delta$*.*

This alternative fairness definition essentially relaxes p-fair ranking definition, such that for every position, the proportionate representation of items with protected attribute value p is allowed to have at most δ deviation (an input parameter) from its original p-fair ranking. Using the running example, if gender is the protected attribute with 50% representation of male and female, then the relaxed p-fairness with $\delta = 1$ implies at least 1 male and at least 1 female in the top-4 items, at least 2 males and at least 2 females in the top-6 items, and so on.

6.2.1 Problem Formulation

P1: Individual p-fair rank (or IPF). Given a ranking ρ find a *p-fair* ranking that is closest to ρ in Kendall-Tau distance.

P2: Rank aggregation under p-fairness (or RAPF). Given m rankings $\rho_1, \rho_2, \ldots, \rho_m$ find a *p-fair* ranking that minimizes the Kemeny distance to these m rankings. We observe that RAPF is NP-Hard which directly follows from the fact that rank aggregation considering unconstrained Kemeny distance minimization is NP-hard when $m \geq 4$ [8].

We study **IPF** and **RAPF** for binary and multi-valued protected attributes considering fairness as a constraint. By that process, it is likely to deteriorate the Kemeny Distance values, i.e., the Kemeny Distance of an unfair rank aggregation is likely to be smaller than that of a fair one (recall Column 1 and Column 3 of Table 6.4). These choices and other alternative ways of incorporating fairness inside rank aggregation are explored in Section 6.7.

We also study **IPF** and **RAPF** subject to the relaxed p-fairness. Our proposed solutions trivially adapt for this version and we omit those for brevity. Experimental results based on this relaxed definition are included in Section 6.6.4.

Table 6.4 Rank Aggregation Results of Comparable Methods Using Section 6.1.1 Example Considering Gender as the Protected Attribute

Rank	Rank aggregation (without fairness)	Rank aggregation (with statistical parity) [87]	Rank aggregation (with p-fairness)
1	Amy (Female)	Amy (Female)	Amy (Female)
2	Molly (Female)	Molly (Female)	Park (Male)
3	Abigail (Female)	Abigail (Female)	Molly (Female)
4	Kim (Male)	Kim (Male)	Kabir (Male)
5	Lee (Male)	Lee (Male)	Abigail (Female)
6	Park (Male)	Park (Male)	Kim (Male)
7	Kabir (Male)	Kabir (Male)	Lee (Male)
8	Damien (Male)	Damien (Male)	Aaliyah (Female)
9	Andres (Male)	Andres (Male)	Damien (Male)
10	Aaliyah (Female)	Aaliyah (Female)	Kiara (Female)
11	Kiara (Female)	Kiara (Female)	Andres (Male)
12	Jazmine (Female)	Jazmine (Female)	Jazmine (Female)
Kemeny Distance	34	34	46

6.3 Related Work and Comparison

Rank Aggregation. The rank aggregation study was initiated in the early 2000s by Dwork et. al. [59]. Since then, rank aggregation and several of its variants have been well studied, including rank aggregation considering different optimization functions, rank aggregation with partial ranking information, or with ties [7, 8, 24, 38, 63]. Kemeny optimal rank aggregation which minimizes the sum/average Kendall-Tau distances [82,84] to the individually ranked lists is the most popular variant. In [8,24], the authors show that computing the Kemeny optimal rank aggregation is NP-hard for 4 or more rankings. There exist both randomized and deterministic approximation algorithms for rank aggregation [8, 136, 137]. In [8], Ailon et al. introduced a randomized approximation algorithm with a $\frac{4}{3}$ approximation factor. In [136, 137], the authors propose deterministic pivoting algorithms with the same approximation factors. In [53] Conitzer et al. propose an exact integer programming solution for the Kemeny optimal rank aggregation.

One of the early yet popular results in this space is the randomized algorithm Pick-a-Perm [8, 59] that is shown to admit a $\frac{1}{2}$ approximation factor for the Kemeny Rank Aggregation Problem in expectation. We adapt Pick-a-Perm in our proposed solution for the **RAPF** *problem.*

Alternative rank aggregation measures. Other than Kemeny, alternative measures of the quality of rank aggregations, such as, those based on Spearman's Footrule and Borda's Method [59]. We note that finding an optimal rank aggregation using Spearman's Footrule based measure is computationally easy. However, it is *open* whether the **RAPF** problem using Spearman's Footrule distance is computationally tractable. On the other hand, the **IPF** problem using Spearman's Footrule distance is tractable. We design a polynomial time algorithm for the **IPF** problem in Spearman's Footrule distance and use it to approximate the **IPF** problem in Kendall-Tau distance. Borda's method [34] is a "positional" method. It assigns a score corresponding to the position in which a candidate appears within each voter's ranked list of preferences, and the candidates are sorted by their total score. Rank aggregation using Borda's method is also computationally easy, however, it does not satisfy the Condorcet criterion. Since Borda's method does not induce a distance between rankings it is unclear how to extend it to satisfy the p-fairness constraint.

Proportionate Fairness. Based on the Chairman assignment problem [134], the idea of proportionate fairness (p-fairness) was studied in the context of resource scheduling [26]. The Chairman assignment problem simply studies how to select a chairman for a union from k states such that at any time the accumulated number of chairmen from each state is proportional to its weight. In [26], Baruha et al. propose an algorithm for generating the p-fair schedule. Then, [27] introduces a series of algorithms for different single resource p-fair scheduling problems. Note that p-fairness is a group fairness criteria that is close to statistical or demographic parity [50] studied in the context of group fairness. *We note that for the rank aggregation problem, p-fairness is more suitable and stronger than statistical parity, because it ensures statistical parity for every position in the ranked order. This makes the problem significantly harder and the existing solutions do not trivially adapt.*

Social Choice Theory. Various ranking methods have been studied in the field of social choice theory [17, 65, 82, 100, 101, 146]. Early social choice theory literature considered rank aggregation in the context of preference aggregation methods [82, 101, 146]. The social choice theory papers [17, 65] focus on Arrow's impossibility theorem. This theorem states that it is impossible to have a rank aggregation method that simultaneously satisfies several conditions some of which relate to fairness. The paper [100] seeks to identify rank aggregation methods that are "close" to satisfying Arrow's conditions, enabling decisions that are fairer in practice. However, the focus of these works is to propose *models*, whereas, our primary goal is to develop efficient computational framework by adapting some of these proposed models.

6.3.1 Fair Ranking Solutions

Several recent fair ranking studies focus on achieving fairness on a *single* rank [18, 45, 72, 147]. Celis et al. [45] introduce a top-k fairness measure that ensures a given upper and lower bound of the representation of each of the protected attribute values in the top-k, for fixed values of k. They use Spearman's footrule-like distance which is easier than Kendall-Tau distance since it can be modeled by a maximum weight perfect matching problem in a bipartite graph. They provide a dynamic programming exact algorithm, and efficient approximation algorithms. In [147], Zehlike et al. extend group fairness using the standard notion of protected groups and ensure that the proportion of protected candidates in every top-k ranking remains statistically above a given minimum (while not ensuring any upper bound). Asudeh et al. [18] propose sweep-line-based algorithms for a more general fairness ranking problem.

Next, we describe two related works in more detail: the first one is a recent work DetConstSort [72] that studies a variant of the **IPF** problem. The other one is FairILP [87], which to the best of our knowledge is the only recent work that studies

some version of fair rank aggregation alas only with binary protected attributes and thus can be compared to **RAPF**.

DetConstSort Geyik et al. [72] propose Algorithm DetConstSort to produce fairness-aware ranking given an input ranking. This algorithm ensures that for every protected attribute value p, and for every $k \in [1..n]$ the number of items with protected attribute value p among the top k ranked items in the output ranking is at least $\lfloor f(p) \cdot k \rfloor$, where $f(p)$ is the fraction of items with protected attribute value p, that is, $f(p) = \frac{1}{n} \sum_{v \in V} \mathbf{1}_{A(v)=p}$. Essentially, Algorithm DetConstSort produces a ranking that only satisfies the lower bound of p-fairness.

Example 4. Statement: DetConstSort **[72] does not produce the closest ranking that satisfies the p-fairness lower bound.** *We simulate the running of Algorithm* DetConstSort *on the ranking given by Member 1 in Table 6.1 considering* seniority level *as the protected attribute. The algorithm scans the ranked items in descending order starting at the top ($k = 1$), and checks at each position, whether any value of the protected attribute becomes "tight" and thus an item with this value needs to be inserted to the tentative output ranking. For the ranking given by Member 1, no seniority level becomes tight at $k = 1, 2$. At $k = 3$, $\lfloor f(Senior) \cdot k \rfloor = \lfloor 5/12 * 3 \rfloor = 1$ and $\lfloor f(Mid\ career) \cdot k \rfloor = \lfloor 4/12 * 3 \rfloor = 1$. So, the top ranked Senior candidate (Damien) and the top ranked Mid career candidate (Kim) are inserted to the tentative output ranking. Since Kim is ranked higher than Damien in the input ranking, the tentative (ordered) output ranking is $[Kim, Damien]$. At $k = 4$, $\lfloor f(Junior) \cdot k \rfloor = \lfloor 3/12 * 4 \rfloor = 1$ and the top Junior candidate Molly needs to be inserted in the list. Since Molly is ranked higher than both Kim and Damien in the input ranking and since both Kim and Damien can be pushed to position 3 without violating the p-fairness lower bound, Molly is inserted into position 1 of the tentative output ranking which is now $[Molly, Kim, Damien]$. Continuing in the*

same manner, the final output ranking is

$$[Molly, Kim, Lee, Damien, Amy, Park,$$
$$Andres, Abigail, Aaliyah, Kabir, Kiara, Jazmine]$$

The Kendall-Tau distance between the Member 1 ranking and the output ranking is 12. However, consider the following ranking.

$$[Molly, Amy, Kim, Damien, Abigail, Lee,$$
$$Andres, Park, Aaliyah, Kabir, Kiara, Jazmine]$$

It also satisfies the p-fairness lower bound and the Kendall-Tau distance between it and the Member 1 ranking is only 8.

Example 5. Statement: DetConstSort **[72] does not produce a p-fair ranking.** *The ranking produced by* DetConstSort *in Example 4 violates the upper bound of the p-fairness condition, since the seniority level of 2 out of the top 3 candidates is Mid career but* $\lceil f(Mid\ career) \cdot 3 \rceil = \lceil 4/12 * 3 \rceil = 1 < 2$.

FairILP Kuhlman and Rundensteiner [87] consider fairness aware rank aggregation in a setting of a *binary protected attribute.* To measure fairness they propose *pairwise statistical parity.*

Definition 6. Pairwise statistical parity. *For a ranking* σ *with a binary protected attribute, let* V_i *be the set of items with protected attribute value* i*, we define* $R_{par}(\sigma)$ *as:*

$$R_{par}(\sigma) = \frac{1}{|V_1||V_2|} \left| \sum_{\{u \in V_1\}} \sum_{\{v \in V_2\}} \left(\mathbf{1}_{\sigma(u)<\sigma(v)} - \mathbf{1}_{\sigma(v)<\sigma(u)} \right) \right|.$$

The ranking σ satisfies pairwise statistical parity if $R_{par}(\sigma) = 0$. The relaxed pairwise statistical parity requires that $R_{par}(\sigma) \leq \delta$, for a given $\delta \geq 0$. The *unnormalized* pairwise statistical parity is defined as $|V_1||V_2|R_{par}(\sigma)$.

Given m rankings $\rho_1, \rho_2, \ldots, \rho_m$, FairILP finds a ranking σ whose pairwise unnormalized statistical parity is bounded by a given $\delta \geq 0$ that is closest to the input rankings in Kemeny distance.

Example 6. Statement: FairILP **[87] is not necessarily p-fair even with** $\delta = 0$.

Consider the running example and assume that the (binary) protected attribute considered is gender.

Table 6.4 shows three aggregated rankings for the running example, the first without fairness, with second subject to pairwise statistical parity with $\delta = 0$*, and the third subject to p-fairness. Note that the first two rankings are identical, which implies that pairwise statistical parity does not imply p-fairness. Intuitively, the reason for this is that pairwise statistical parity just considers pairs of items with different protected attribute value in an aggregated manner and does not consider the actual positions of the items in the aggregated ranking.*

In summary, **IPF** *is stronger than any of the existing fairness aware single rank problem [18,45,72,147], because we consider proportionate representation considering both lower and upper bound of the protected attributes for every position. Similarly,* **RAPF** *promotes a stronger notion of fairness compared to* FairILP *[87], as well as consider both binary and multi-valued protected attribute.*

6.4 Individual p-fairness (IPF)

In this section, we describe our proposed solutions for the individual p-fairness or the **IPF** problem. First, we consider the binary case, denoted **BinaryIPF**, in which the protected attribute has two values, i.e., $\ell = 2$. We present an exact greedy algorithm GrBinaryIPF for **BinaryIPF**, prove its correctness, and analyze its running time. Then, we consider the general case of **IPF**, denoted **MultiValuedIPF**, when $\ell > 2$. We demonstrate that **MultiValuedIPF** cannot be solved using a greedy algorithm

similar to the binary case, and present two solutions: a dynamic programming based exact algorithm ExactMultiValuedIPF, and an approximation algorithm ApproxMultiValuedIPF based on minimum weight matching. We analyze the running time and prove the correctness of both algorithms.

6.4.1 BinaryIPF

In this subsection we present an exact algorithm to **BinaryIPF** in which the protected attribute value of each of the items can take only two possible values. Algorithm GrBinaryIPF takes an input ranking ρ and the output is a p-fair ranking σ with the minimum Kendall-Tau distance to ρ. The algorithm builds on Lemma 7 that implies that if item u is the i-th item (counting from the top) with protected attribute value p in ranking ρ, then the same item is also the i-th item with protected attribute value p in ranking σ.

Baruah et al. [26] proved the following for any p-fair ranking. Consider the ranks of the items with protected attribute value p in the p-fair ranking. Then, for $i \in [1..f(p) \cdot n]$, the i-th such item has to be ranked within the interval

$$[\left\lfloor \frac{i-1}{f(p)} \right\rfloor + 1, \left\lceil \frac{i}{f(p)} \right\rceil].$$

Thus, for every item $v \in V$, we define the interval $[\texttt{top}(v), \texttt{bot}(v)]$ as the feasible positions of this item in any p-fair ranking that is closest to permutation ρ.

Algorithm GrBinaryIPF whose pseudo code is given in Algorithm 9 starts by computing $\texttt{bot}(v)$, for every $v \in V$ (Line 1). Then, it sorts the items according to their rank, and partitions the sorted list into two sub-lists, one for each protected attribute value (Line 3). The ranking σ is constructed from top to bottom. For each position i, Algorithm GrBinaryIPF considers the current top items u_1 and u_2 in each of the sub-lists. In case $\texttt{bot}(\cdot)$ of one of these two items is "tight", that is, equals i, Algorithm GrBinaryIPF assigns i to σ of this item. (In Lemma 8, we show that

both items cannot be tight.) Otherwise, Algorithm GrBinaryIPF assigns i to σ of the item among u_1 and u_2 that is ranked higher in ρ. (Lines 5–12).

Algorithm 9 GrBinaryIPF

1: compute $\texttt{bot}(v)$ for each item $v \in V$
2: sort the items according to the ranking ρ
3: partition the sorted list into two sub-lists L_1, L_2, one for each protected attribute value
4: **for** $i = 1$ to n **do**
5: Let u_1 and u_2 be the current top items in L_1 and L_2
6: **if** $\texttt{bot}(u_1) = i \vee \texttt{bot}(u_2) = i$ **then**
7: $v \leftarrow$ the tight item among u_1 and u_2
8: **else**
9: $v \leftarrow$ the higher ranked item in ρ among u_1 and u_2
10: **end if**
11: $\sigma(v) \leftarrow i$
12: remove v from its ordered list
13: **end for**
14: **return** σ

We demonstrate the algorithm considering as input the initial ranking provided by Member 2 in the running example and the binary protected attribute gender. The following two sub-lists are obtained:

$$L_{\text{female}} = [Amy, Molly, Abigail, Aaliyah, Kiara, Jazmine]$$

$$L_{\text{male}} = [Kim, Lee, Park, Kabir, Damien, Andres]$$

Note that $\texttt{bot}(Amy) = \texttt{bot}(Kim) = 2$. Amy is put into the first position in σ since Amy is ranked higher than Kim in ρ. Since $\texttt{bot}(Kim) = 2$, Kim is assigned

be the second place in σ. By repeating this procedure, we end up with the ranking:

$$[Amy, Kim, Molly, Lee, Abigail, Park,$$
$$Kabir, Aaliyah, Damien, Kiara, Andres, Jazmine],$$

where $\mathcal{K}(\rho, \sigma) = 17$.

Running Time Analysis: It is straightforward to see that Algorithm GrBinaryIPF runs in $O(n)$ time, since all the computations can be done by a constant number of linear scans over the items.

As mentioned above the algorithm is based on the following lemma.

Lemma 7. *Consider two elements u and v with the same protected attribute value ($A(u) = A(v)$). If $\rho(u) < \rho(v)$ then $\sigma(u) < \sigma(v)$ in any p-fair ranking σ that is closest to permutation ρ,*

Proof Sketch:

The proof is by contradiction. Assume that $\sigma(v) < \sigma(u)$. Consider the ranking σ' given by swapping $\sigma(v)$ and $\sigma(u)$, that is, $\sigma'(u) = \sigma(v)$, $\sigma'(v) = \sigma(u)$, and for every $w \in V \setminus \{u, v\}$, $\sigma'(w) = \sigma(w)$. The ranking σ' is also p-fair (since $A(u) = A(v)$), and we prove below that $\mathcal{K}(\sigma', \rho) < \mathcal{K}(\sigma, \rho)$; a contradiction.

Since $\rho(u) < \rho(v)$ and $\sigma(u) > \sigma(v)$ we have

$$(\sigma(u) - \sigma(v))(\rho(u) - \rho(v)) < 0$$
$$(\sigma'(u) - \sigma'(v))(\rho(u) - \rho(v)) > 0.$$

So, the pair (u, v) contributes 1 to $\mathcal{K}(\sigma, \rho)$ and 0 to $\mathcal{K}(\sigma', \rho)$. Since for $x, y \in V \setminus \{u, v\}$, we have $(\sigma(x) - \sigma(y)) = (\sigma'(x) - \sigma'(y))$ all such pairs (x, y) contribute the same to $\mathcal{K}(\sigma, \rho)$ and $\mathcal{K}(\sigma', \rho)$. Consider $w \in V \setminus \{u, v\}$, such that either $\sigma(w) > \sigma(u)$ or $\sigma(w) < \sigma(v)$. We have $(\sigma(w) - \sigma(u)) = (\sigma'(w) - \sigma'(u))$ and $(\sigma(w) - \sigma(v)) = (\sigma'(w) - \sigma'(v))$. Thus, the contribution of the pairs (u, w) and (v, w) is the same to $\mathcal{K}(\sigma, \rho)$ and $\mathcal{K}(\sigma', \rho)$.

We are left with the case $w \in V \setminus \{u, v\}$, such that $\sigma(w) \in (\sigma(v), \sigma(u))$. In this case

$$(\sigma(v) - \sigma(w)) = (\sigma'(u) - \sigma'(w)) < 0$$
$$(\sigma(u) - \sigma(w)) = (\sigma'(v) - \sigma'(w)) > 0.$$

Clearly, $\rho(v) - \rho(w) > \rho(u) - \rho(w)$. Thus

$$(\sigma(v) - \sigma(w))(\rho(v) - \rho(w)) < (\sigma'(u) - \sigma'(w))(\rho(u) - \rho(w))$$
$$(\sigma(u) - \sigma(w))(\rho(u) - \rho(w)) < (\sigma'(v) - \sigma'(w))(\rho(v) - \rho(w)),$$

which implies that the contribution of the pairs (u, w) and (v, w) to $\mathcal{K}(\sigma', \rho)$ is at most their contribution to $\mathcal{K}(\sigma, \rho)$. ∎

Lemma 8. *For any iteration i of the algorithm one cannot have* $\texttt{bot}(u_1) = i$ *and* $\texttt{bot}(u_2) = i$.

Proof. Since $\texttt{bot}(\cdot)$ is nondecreasing as we iterate over the items, then for all items v that are added to σ up to iteration i we have $\texttt{bot}(v) \leq i$. Suppose that both $\texttt{bot}(u_1) = i$ and $\texttt{bot}(u_2) = i$. In this case thare are $i + 1$ items ($i - 1$ items from previous iterations together with u_1 and u_2) that need to be ranked in the top i places, which is infeasible, and thus in contradiction to the feasibility of p-fair ranking as proved in [26]. □

Theorem 22. *Algorithm* GrBinaryIPF *returns the exact solution to the* **BinaryIPF** *problem.*

Proof. To obtain a contradiction assume that μ is the p-fair ranking with minimum distance to ρ, and that $\mu \neq \sigma$. Let i be the top rank where μ and σ differ. Let u be the item ranked i in μ and v be the item ranked i in σ. Since the order of the items with the same value of the protected attribute has to be the same in both μ and σ, we must have that $A(u) \neq A(v)$. Without loss of generality assume that u is in sub-list L_1 and v is in sub-list L_2. Certainly $\texttt{bot}(u) \neq i$ and $\texttt{bot}(v) \neq i$ as otherwise either μ or σ would not be p-fair. Thus, according to our algorithm $\rho(v) < \rho(u)$. Let j be the rank of v in μ. Since i is the top rank where μ and σ differ we must have $j > i$. Also, all items ranked $i, \ldots, j - 1$ in μ must be in sub-list L_1. As otherwise, the order of the items from L_2 would not be the same in both μ and σ.

Since $\rho(v) < \rho(u)$, then for item w ranked $j - 1$ in μ, we also have $\rho(v) < \rho(w)$. Item w is ranked lower than $j - 1$ in σ, thus $\texttt{bot}(w) \geq j$. Since the rank of item v in σ is i, $\texttt{top}(v) \leq i \leq j - 1$. However, then the ranking μ' given by swapping the items w and v ranked $j - 1$ and j in μ is p-fair and similar to Lemma 7 it can be shown to be closer than μ to the ranking ρ, a contraction. □

6.4.2 MultiValuedIPF

In this subsection we present an approximate and an exact algorithms for **MultiValuedIPF**. The input is a ranking ρ and the output is a p-fair ranking σ that minimizes the Kendall-Tau distance to ρ.

MultiValuedIPF is a harder problem. We begin this section by demonstrating that a simple greedy scheme is not adequate to solve **MultiValuedIPF**

like in the binary case. Consider the following artificial example consisting of 20 items with four possible values of their protected attribute. There are 5 items with protected attribute value a, 10 items with protected attribute value b, 4 items with protected attribute value c, and 1 item with protected attribute value d. Note that we must have one item with protected attribute value a in each block of 4 ranked items starting from the top, one item with protected attribute value b in each block of 2 ranked items starting from the top, and one item with protected attribute value c in each block of 5 ranked items starting from the top.

Now, consider the ranking $\rho = 1, \ldots, 20$ (the identity permutation [131]). The value of the protected attribute for these items in given in the table 6.5.

Table 6.5 Protected Attribute Values

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a	a	b	b	b	d	c	b	c	b	a	b	c	b	a	b	c	b	a	b

The greedy algorithm creates a p-fair ranking by starting with the highest ranked item 1 (with protected attribute value a), then the 2 items 3, and 4 (with value b), then the item 6 (with value d), next it must pick item 7 (with value c), and 5 (with value b), and only then item 2 (with value a). From this point on the p-fair ranking coincides with the original ranking; that is, items $8, \ldots, 20$ appear in order, and the resulting ranking is $(1, 3, 4, 6, 7, 5, 2, 8, \ldots, 20)$. The Kendall-Tau distance of this ranking from ρ is 7.

However, the optimal p-fair ranking is $(1, 3, 4, 7, 2, 5, 6, 8, \ldots, 20)$ which is closer to ρ with Kendall-Tau distance 5.

Approximation Algorithm We first present an efficient algorithm ApproxMultiValuedIPF for computing **IPF** that is based on minimum weight matching in a bipartite graph. Then, we proceed with the exact algorithm that is more complex.

The proposed algorithm ApproxMultiValuedIPF whose pseudo code is given in Algorithm 10 considers as the underlying abstraction a weighted bipartite graph $G(V, Y, E)$, where $|V| = |Y| = n$. The nodes in V represent the items and the nodes in Y represent their potential position.

Recall that Baruah et al. [26] proved that in any p-fair ranking the position of an item $v \in V$ must be within $[\texttt{top}(v)..\texttt{bot}(v)]$. Consequently, an edge $e_{vy} \in E$ exists, if $y \in [\texttt{top}(v)..\texttt{bot}(v)]$. The algorithm assigns the weight $|\rho(v) - y|$ to every edge $e_{vy} \in E$ (Line 2), which is the Spearman's footrule distance between position of v in ρ and y. Then, ApproxMultiValuedIPF finds a perfect matching in this graph (Line 4), and the output of the perfect matching induces a p-fair ranking σ (Line 6) that is closest to ρ in Spearman's footrule distance.

Algorithm 10 ApproxMultiValuedIPF(G)

1: **for** $e_{vy} \in E$ **do**
2: $\quad weight(e_{vy}) \leftarrow |\rho(v) - y|$
3: **end for**
4: Find M a minimum weight perfect matching in G
5: **for** $e_{vy} \in M$ **do**
6: $\quad v$ is set to be the item ranked y in σ
7: **end for**
8: **return** σ

We demonstrate the algorithm considering as input the initial ranking provided by Member 2 and the ternary protected attribute *seniority level*. The top ranked candidate of Member 2 is *Park* whose seniority level is *Mid career*. Note that $f(Mid\ career) = \frac{4}{12} = \frac{1}{3}$. Thus, $\texttt{top}(Park) = 1$ and $\texttt{bot}(Park) = \lceil 1 \cdot \frac{3}{1} \rceil = 3$. Thus, there are 3 edges connecting to nodes in Y: $e_{Park,1}, e_{Park,2}, e_{Park,3}$ with weights $0, 1, 2$, respectively. The rest of the edges of the bipartite graph are computed similarly. The minimum weight perfect matching in the created bipartite graph implies the following

ranking, and the Spearman's footrule distance to the original ranking of Member 2 is 18:

$$[Park, Amy, Damien, Kabir, Andres, Molly,$$
$$Kim, Aaliyah, Abigail, Kiara, Lee, Jazmine].$$

Running Time: The running time of Algorithm ApproxMultiValuedIPF is dominated by the running time of the minimum weight perfect matching which is $O(n^{2.5} \log n)$.

Theorem 23. ApproxMultiValuedIPF *admits a* 2*-approximation factor for the* **IPF** *problem.*

Proof Sketch:

A lemma similar to Lemma 7 holds also for the Spearman's footrule distance, and thus any p-fair ranking that is closest in Spearman's footrule distance to the ranking ρ has to correspond to a perfect matching in G. It follows that the minimum weight perfect matching implies a p-fair ranking σ that is closest in Spearman's footrule distance to the ranking ρ. Let μ be a p-fair ranking that is closest (in Kendall-Tau distance) to ρ. Clearly, $\mathcal{S}(\rho, \sigma) \leq \mathcal{S}(\rho, \mu)$. Thus, by the inequalities in [56] $\mathcal{K}(\rho, \sigma) \leq \mathcal{S}(\rho, \sigma) \leq \mathcal{S}(\rho, \mu) \leq 2\mathcal{K}(\rho, \mu)$. We conclude that σ is a 2-approximation to a p-fair ranking that is closest (in Kendall-Tau distance) to the ranking ρ. ∎

Exact Algorithm We present a dynamic programming based exact algorithm ExactMultiValuedIPF for the **MultiValuedIPF** problem. We prove that when ℓ, the number of different values of the protected attribute, is a constant (or even logarithmic in n), Algorithm ExactMultiValuedIPF computes the closest p-fair ranking in polynomial time. The running time of ExactMultiValuedIPF is exponential in ℓ, and thus when $\ell = \Omega(n)$, the running time of ExactMultiValuedIPF

is exponential. Due to space constraints we just describe the intuition behind this algorithm.

Consider an index $1 \leq k < n$. Suppose that we wish to break the problem of computing the closest p-fair ranking into two subproblems. One is computing the top k items of the p-fair ranking and the other is computing the bottom $n-k$ items. Let's concentrate on computing the bottom $n-k$ items, that is, which items are in positions $k+1, \ldots, n$ and their order. Certainly, this depends on which items are ranked in the top k (but it does not depend on the order of these top k items). The algorithm is based on the observation that the amount of this information is limited. Note that for each item u if $\texttt{bot}(u) \leq k$ then u must be in the top k and if $\texttt{top}(u) > k$ then u must be in the bottom $n-k$. The only ambiguity is regarding the items u for which $\texttt{top}(u) \leq k$ and $\texttt{bot}(u) \geq k+1$. It follows that for all these items u we have $k, k+1 \in [\texttt{top}(u), \texttt{bot}(u)]$. By the definition of $\texttt{top}(\cdot)$ and $\texttt{bot}(\cdot)$, for any two items u and v such that $A(u) = A(v)$, the intersection of $[\texttt{top}(u), \texttt{bot}(u)]$ and $[\texttt{top}(v), \texttt{bot}(v)]$ consists of no more than a single item. It follows that for each of the ℓ possible values of the protected attribute we have exactly one item u with this value for which $k, k+1 \in [\texttt{top}(u), \texttt{bot}(u)]$. Since each such item can be either in the top k or in the bottom $n-k$, the number of possibilities is bounded by 2^ℓ.

The dynamic programming based exact algorithm ExactMultiValuedIPF for the **MultiValuedIPF** problem works as follows. The algorithm works in n iterations. For $k = 1, \ldots, n$, and for every subset L_k of the items u for which $k, k+1 \in [\texttt{top}(u), \texttt{bot}(u)]$, it computes the optimal rank of the top k elements of the closest p-fair ranking subject to the constraint that the items in L_k are in the bottom $n-k$. (Note that there may not be a feasible solution for some of these subsets.) The computation is done using the optimal ranking of the top $k-1$ elements computed for all possible subsets L_{k-1}. It is not difficult to see that each such computation

can be implemented in $O(\ell 2^{\ell})$ time, and thus the algorithm is polynomial as long as $\ell = O(\log n)$.

Theorem 24. *The running time complexity of* ExactMultiValuedIPF *is linear when ℓ is constant and polynomial when ℓ is $O(\log n)$.*

6.5 Rank Aggregation Subject to p-fairness (RAPF)

In this section, we present two scalable solution frameworks for the **RAPF** problem both for binary and multi-valued protected attribute. Algorithm RandAlgRAPF is randomized, highly scalable, but the approximation factor is in expectation. Algorithm AlgRAPF, on the other hand, produces a deterministic approximation factor, but less scalable than its randomized counterpart.

6.5.1 Randomized Algorithm

We start with the randomized algorithm RandAlgRAPF. Input to this algorithm are m rankings $\rho_1, \rho_2, \ldots, \rho_m$ and the output is σ, the aggregated p-fair ranking. Algorithm RandAlgRAPF randomly selects one of the $\rho_1, \rho_2, \ldots, \rho_m$ input rankings uniformly, denoted $\rho_{randInd}$. Then, Algorithm RandAlgRAPF calls the subroutine AlgIPF(ρ), with the parameter $\rho_{randInd}$. The subroutine AlgIPF(ρ) computes the p-fair ranking that is closest to this selected ranking $\rho_{randInd}$ (either exactly or approximately). The resulting p-fair ranking σ is the output of Algorithm RandAlgRAPF.

The subroutine AlgIPF(ρ) can invoke any of the algorithms described in Section 6.4. Recall that GrBinaryIPF produces an exact p-fair solution of the binary **IPF** problem. For multi-valued **IPF**, the highly scalable Algorithm ApproxMultiValuedIPF produces a 2 approximation factor, whereas, the dynamic programming based solution Algorithm ExactMultiValuedIPF is more expensive but produces an exact solution for the multi-valued **IPF** problem. Depending on the underlying **IPF** problem, any of these could be used inside AlgIPF.

We prove that *in expectation* the approximation factor of the aggregate ranking returned by this incredibly simple Algorithm RandAlgRAPF is 2+ the approximation factor of the algorithm for the **IPF** problem invoked in AlgIPF(ρ).

Running Time: The running time of Algorithm RandAlgRAPF is the same as the running time of Algorithm AlgIPF.

Theorem 25. *Let α be the approximation factor of the algorithm for the* **IPF** *problem invoked in* AlgIPF(ρ). *The expected Kemeny distance of the ranking returned by Algorithm* RandAlgRAPF *to $\rho_1, \rho_2, \ldots, \rho_m$ is bounded by $\alpha+2$ times the minimum Kemeny distance of any p-fair ranking to $\rho_1, \rho_2, \ldots, \rho_m$.*

Proof. Let OPT_U be the optimal (unconstrained) aggregate ranking of $\rho_1, \rho_2, \ldots, \rho_m$, and let OPT_F be the optimal p-fair aggregate ranking. For $i \in [1..m]$, let $\sigma_i =$ AlgIPF(ρ_i). Note that for $i \neq randInd$, we do not compute σ_i; it is just used in the proof. We have

$$\kappa(OPT_U, \rho_1, \rho_2, \ldots, \rho_m) \leq \kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m).$$

Since for $i \in [1..m]$, $\mathcal{K}(\sigma_i, \rho_i) \leq \alpha\mathcal{K}(OPT_F, \rho_i)$ we also have

$$\sum_{i=1}^{m} \mathcal{K}(\sigma_i, \rho_i) \leq \alpha\kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m).$$

By the triangle inequality for any $i \in [1..m]$ we have

$$\begin{aligned}
&\kappa(\sigma_i, \rho_1, \rho_2, \ldots, \rho_m) \\
&\quad \leq \sum_{j=1}^{m} [\mathcal{K}(\sigma_i, \rho_i) + \mathcal{K}(\rho_i, OPT_U) + \mathcal{K}(OPT_U, \rho_j)] \\
&\quad = m\,[\mathcal{K}(\sigma_i, \rho_i) + \mathcal{K}(\rho_i, OPT_U)] + \kappa(OPT_U, \rho_1, \rho_2, \ldots, \rho_m)
\end{aligned}$$

Summing over all $i \in [1..m]$ we get

$$\begin{aligned}
&\sum_{i=1}^{m} \kappa(\sigma_i, \rho_1, \rho_2, \ldots, \rho_m) \\
&\quad = m \sum_{i=1}^{m} [\mathcal{K}(\sigma_i, \rho_i) + \mathcal{K}(\rho_i, OPT_U)] \\
&\qquad\qquad + m \cdot \kappa(OPT_U, \rho_1, \rho_2, \ldots, \rho_m) \\
&\quad = m \sum_{i=1}^{m} \mathcal{K}(\sigma_i, \rho_i) + 2m \cdot \kappa(OPT_U, \rho_1, \rho_2, \ldots, \rho_m) \\
&\quad \leq m \cdot \alpha \cdot \kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m) + 2m \cdot \kappa(OPT_U, \rho_1, \rho_2, \ldots, \rho_m) \\
&\quad \leq m \cdot (\alpha + 2) \cdot \kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m)
\end{aligned}$$

The expected distance is $\frac{1}{m}$ of this sum, that is

$$E\left[\kappa(\sigma_{randInd}, \rho_1, \rho_2, \ldots, \rho_m)\right] \leq (\alpha + 2) \cdot \kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m).$$

□

6.5.2 Deterministic Algorithm

We proceed to describe the deterministic algorithm AlgRAPF. Input to this algorithm are m rankings $\rho_1, \rho_2, \ldots, \rho_m$ and the output is σ, the aggregated p-fair ranking. Algorithm AlgRAPF whose pseudo code is given in Algorithm 11 runs in two steps. (1) Invoke the subroutine AlgIPF(ρ) to solve the **IPF** problem for each of the m input rankings (Line 2); (2) out of the m fair rankings produced in step 1, find the ranking that minimizes the Kemeny distance to the input rankings and output it as the aggregated p-fair ranking (Lines 4–11).

As in the randomized case, the subroutine AlgIPF(ρ) inside AlgRAPF computes an approximation to the closest p-fair of ρ by invoking any of the algorithms described in Section 6.4. The resulting approximation factor is 2+ the approximation factor of the chosen algorithm.

Algorithm 11 AlgRAPF($\rho_1, \rho_2, \ldots, \rho_m$)

1: **for** $i \in [1..m]$ **do**
2: $\quad \sigma_i \leftarrow$AlgIPF($\rho_i$)
3: **end for**
4: $min \leftarrow m \cdot n^2$ ▷ upper bound on the distance
5: **for** $i \in [1..m]$ **do**
6: $\quad$ **if** $\kappa(\sigma_i, \rho_1, \rho_2, \ldots, \rho_m) < min$ **then**
7: $\quad\quad min \leftarrow \kappa(\sigma_i, \rho_1, \rho_2, \ldots, \rho_m)$
8: $\quad\quad minInd \leftarrow i$
9: $\quad$ **end if**
10: **end for**
11: **return** σ_{minInd}

We demonstrate Algorithm AlgRAPF using the running example. It first calls subroutine AlgIPF to find the p-fair rankings that are closest to the rankings of each of the 4 members. The Kendall-Tau distances between the resulting p-fair rankings and the original rankings are 6, 3, 4, and 9, respectively. Next, Algorithm AlgRAPF outputs the ranking among these 4 p-fair rankings that minimizes the Kemeny distance to original rankings. The output is the one closest to the ranking of member 2, shown below, and its Kemeny distance to the original rankings is 50.

$$[Park, Amy, Molly, Kabir, Abigail, Damien,$$
$$Kim, Aaliyah, Andres, Kiara, Lee, Jazmin]$$

Running Time: The running time of Algorithm AlgRAPF is m times the running time of algorithm AlgIPF plus $O(m^2 n \log n)$. Note that the Kendall-Tau distance between two rankings can be computed in $O(n \log n)$ time using a binary search tree.

Theorem 26. *Let α be the approximation factor of the algorithm for the* **IPF** *problem invoked in* AlgIPF(ρ). *The aggregate ranking returned by Algorithm* AlgRAPF *is an $\alpha + 2$ approximation of the closest p-fair aggregate ranking of $\rho_1, \rho_2, \ldots, \rho_m$.*

Proof. In Theorem 25 we proved that

$$\sum_{i=1}^{m} \kappa(\sigma_i, \rho_1, \rho_2, \ldots, \rho_m) \leq m \cdot (\alpha + 2) \cdot \kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m).$$

It follows that the minimum distance is bounded by $\frac{1}{m}$ of this sum, and thus

$$\kappa(\sigma_{minInd}, \rho_1, \rho_2, \ldots, \rho_m) \leq (\alpha + 2) \cdot \kappa(OPT_F, \rho_1, \rho_2, \ldots, \rho_m).$$

□

Lemma 9. *Algorithms* RandAlgRAPF *and* AlgRAPF *admit* 3, 3, *and* 4 *approximation factors for the* **RAPF** *problem when* GrBinaryIPF, ExactMultiValuedIPF, *and* ApproxMultiValuedIPF, *respectively, are used as the underlying solutions for the* **IPF** *problem.*

6.6 Experimental Evaluations

The goal of this study is to evaluate the quality and scalability of our proposed solutions, designed for **IPF** and the **RAPF** problems. We also compare our solutions with multiple state-of-the-art solutions [72, 87] to demonstrate how our studied problems promote stronger notion of fairness for the rank aggregation problem.

All algorithms are implemented in Python 3.8. All experiments are conducted on a cluster server machine with 32GB RAM memory, OS: Scientific Linux release 7.8 (Nitrogen), CPU: Intel(R) Xeon(R) CPU E3-1245 v6 @ 3.70GHz. All numbers are presented as an average of 10 runs. For brevity, we present a subset of results that are representative. The code and the data is available at [1].

6.6.1 Dataset Description

We perform evaluations considering 3 real world datasets. (a) Fantasy players choose real athletes for their fantasy teams and generate scores based on the athlete's real

[1] `https://github.com/MouinulIslamNJIT/Rank-Aggregation_Proportionate_Fairness.git` Retrieved on May/01/2021

performance. Rankings of the athletes are provided by real human voters. We use rankings of National Football League (NFL) players for 16 weeks of the 2019 football season from the top 25 experts. (b) German Credit Score: This is a publicly available dataset in the UCI repository. It is based on credit ratings generated by Schufa, a German private credit agency based on a set of variables for each applicant, including age, gender, and marital status, among others. Schufa Score is an essential determinant for every resident in Germany when it comes to evaluating credit rating before getting a phone contract, a long-term apartment rental or almost any loan. We use the credit-worthiness as scores just it is done in [145], and create a protected attribute with 4 different values. (c) MoveLens Dataset: We use MovieLens 25 million movie dataset to select a set of movies that are all rated by the same set of users. The individual user rating is used to create individual ranking. We use the movie genres as the protected attribute. Table 6.6 has further details.

Table 6.6 Real World Datasets

Dataset	#records (n)	# ranks (m)	protected attributes (ℓ)
Fantasy football ranking	55	25	American Football Conference (AFC):**proportion: 50%** , National Football Conference (NFC): **proportion: 50%**
German Credit Score	1000	1	age<35 & sex = female: **proportion: 33.5%** , age≥35 & sex = female: **proportion:** 35.5% age<35 & sex = male: **proportion:** 21.3%, age≥35 & sex = male: **proportion: 9.7%**
MovieLens	268	7	Thriller: **proportion: 2.24%**, Western: **proportion: 6.72%**, Documentary: **proportion: 3.36%**, Comedy: **proportion: 21.64%**, Horror: **proportion: 4.85%**, Musical: **proportion: 0.37%**, Film-Noir: **proportion: 1.49%**, Drama: **proportion: 59.33%**

Synthetic dataset We generate large scale synthetic data [87, 145] using Mallows' Model [95]. The Kemeny rank aggregation has been shown to be a maximum likelihood estimator for this model [145]. It contains two parameters - (i) θ that controls the degree of consensus among the rankings (higher values shows more agreement); (ii) p that dictates the probability of elements of the first group to be ranked higher than elements in the Second group. We refer to [87] for further details. The θ and p are set to 0.9 and 0.7 respectively in our experiments.

6.6.2 Implemented Algorithms

DetConstSort [72] is a fairness-aware ranking algorithm designed towards mitigating algorithmic bias for a single rank. DetConstSort only ensures the lower bound of proportionate representation. As shown in Section 6.3.1, it neither guarantees smallest Kendall-Tau distance nor ensures p-fairness. We implement this for **IPF**.

FairILP [87] finds the closest aggregate ranking that satisfies a bound on the pairwise statistical parity. The original implementation of FairILP is specified for a binary protected attribute. To adapt it for multi-valued protected attribute we ensure that for each value of the protected attribute, the bound on the pairwise statistical parity is satisfied between the items with this value and the rest of the items. In our experiments we set $\delta = 1$ as the (unnormalized) bound on the pairwise statistical parity. We note that due to the definition of pairwise statistical parity, it may be infeasible in many instances to find a solution for $\delta = 0$.

OptIPF is the exact solution for **IPF** produced by solving an Integer Linear Programming (ILP) model using Gurobi Optimizer 9.1. The optimizer does not scale and thus exact solutions cannot be computed for large-scale datasets.

OptRAPF is the exact solution for **RAPF** produced by solving an ILP model using Gurobi Optimizer 9.1. Again, the optimizer only produces the optimal solution on small datasets.

OptRA is the exact solution for rank aggregation without considering fairness, and is produced by solving an ILP model.

Measures. For quality evaluation we use the following measures. (i) Kendall-Tau and Kemeny Distances, (ii) percentage of items satisfying p-fairness, and (iii) approximation factors. For scalability evaluation, we measure the running time.

6.6.3 Summary of Results

Our first observation is that, consistent with our theoretical analysis, p-fairness promotes stronger notion of fairness, by ensuring proportionate representation of each of the protected attribute values for every position in the ranked order. Naturally, incorporating p-fairness inside rank aggregation comes with a cost - the Kendall-Tau and Kemeny distances are typically higher (albeit not substantially worse) for the p-fair rank aggregation than that of OptRA. Second, our experimental results demonstrate that our proposed model and solutions satisfy the fairness criteria proposed in state-of-the-art solutions [72, 87] - however, these aforementioned existing solutions do not extend to satisfy p-fairness. Third, our experimental results corroborate our theoretical results, that is, GrBinaryIPF is exact, ApproxMultiValuedIPF admits a solution that is no more than twice the optimal for **MultiValuedPF**, and AlgRAPF in conjunction with ApproxMultiValuedIPF admits tighter approximation factor compared to our proposed theoretical bound 4. Finally, our scalability results indicate that our proposed solutions are scalable considering very large number of items ($1,000,000$) and ranks ($10,000$). In fact, RandAlgRAPF is insensitive to the number of ranks. We extend our experiments and consider relaxed p-fairness varying $\delta \geq 0$ values as defined in Definition 5.

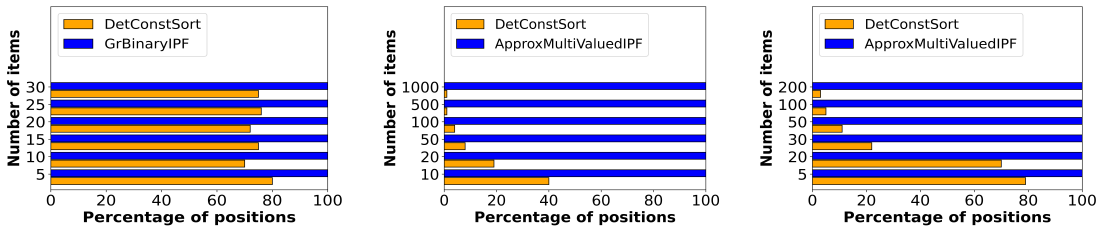
6.6.4 Quality Experiments

In this section we describe the results of our qualitative analysis.

BinaryIPF Results Figures 6.1(a) and 6.2(a) compare the fairness of GrBinaryIPF and DetConstSort. These results clearly indicate that GrBinaryIPF consistently satisfies p-fairness, whereas, DetConstSort does not.

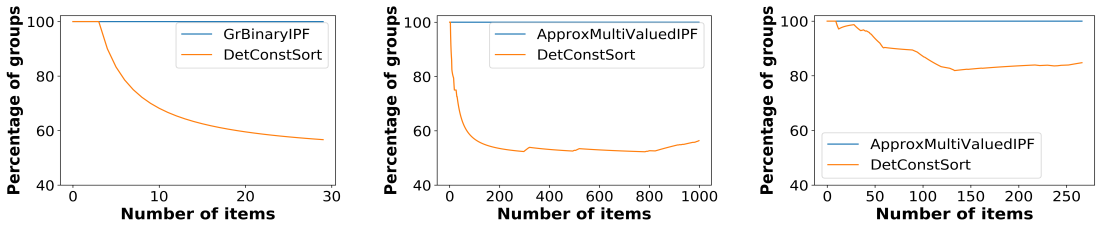
Figure 6.3(a) compares the Kendall-Tau distance between the input ranking and the ranking computed by OptIPF, GrBinaryIPF, and DetConstSort. Consistent with our theoretical analysis OptIPF and GrBinaryIPF always produce the same distance. At times DetConstSort computes a ranking with a smaller distance. This can indeed happen, as DetConstSort does not necessarily compute a p-fair ranking.

Figure 6.4(a) plots the Kendall-Tau distance of the ranking computed by GrBinaryIPF as we relax the p-fairness using $\delta \geq 0$ values. We note that for a small value of δ the relaxed output is the same as input unfair ranking, and the Kendall-Tau distance is 0.



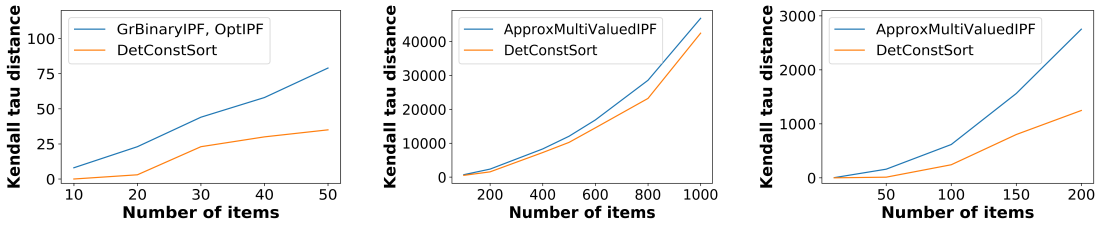
(a) Fantasy football: GrBinaryIPF vs DetConstSort (b) German Credit: ApproxMultiValuedIPF vs DetConstSort (c) MovieLens: ApproxMultiValuedIPF vs DetConstSort

Figure 6.1 Percentage of positions satisfying p-fairness (IPF).



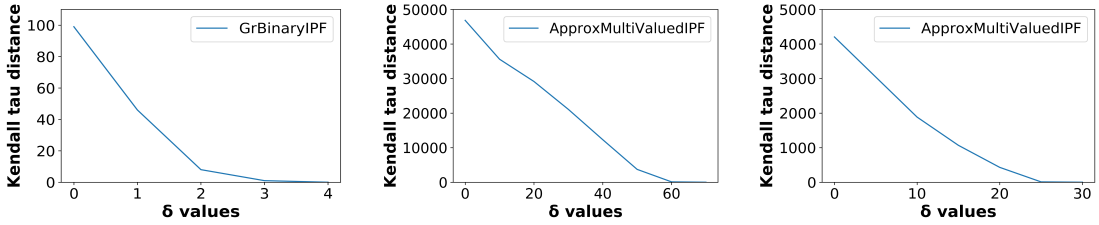
(a) Fantasy football: GrBinaryIPF vs DetConstSort (b) German Credit: ApproxMultiValuedIPF vs DetConstSort (c) MovieLens: ApproxMultiValuedIPF vs DetConstSort

Figure 6.2 Percentage of groups satisfying p-fairness (**IPF**).



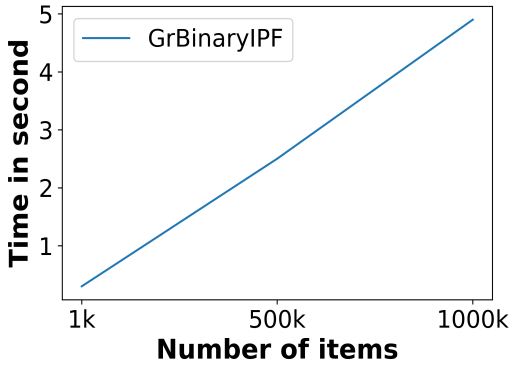
(a) Fantasy football: OptIPF, GrBinaryIPF , DetConstSort (b) German Credit: ApproxMultiValuedIPF vs DetConstSort (c) MovieLens: ApproxMultiValuedIPF vs DetConstSort

Figure 6.3 Kendall-Tau distance for **IPF** algorithms.

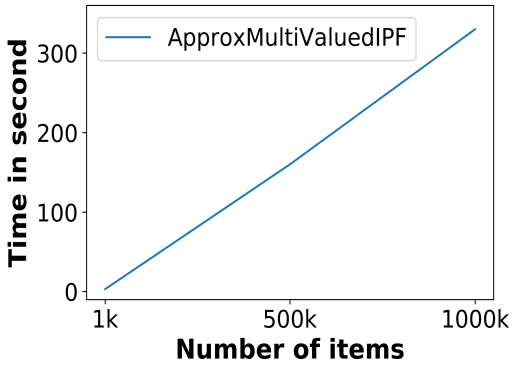


(a) Fantasy Football: GrBinaryIPF (b) German Credit : ApproxMultiValuedIPF (c) MovieLens: ApproxMultiValuedIPF

Figure 6.4 Varying δ analysis IPF.

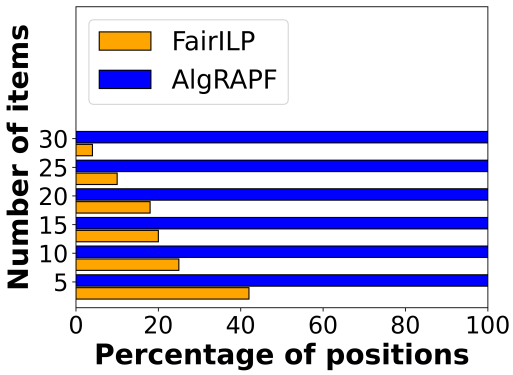


(a) GrBinaryIPF

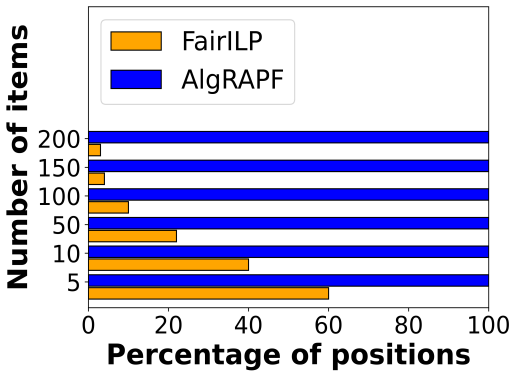


(b) ApproxMultiValuedIPF

Figure 6.5 Running time analysis of **IPF**.

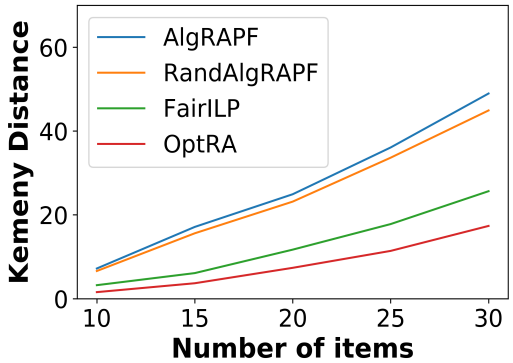


(a) Fantasy football: p-fairness: AlgRAPF vs FairILP [87]

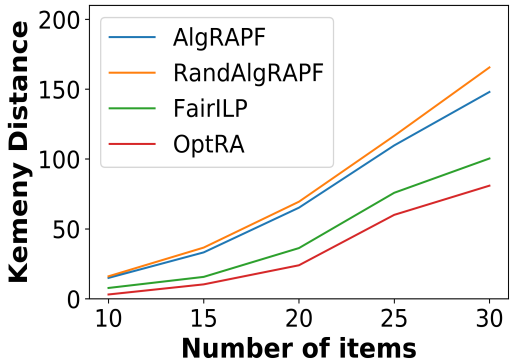


(b) MovieLens: p-fairness: AlgRAPF vs FairILP [87]

Figure 6.6 % of positions satisfying p-fairness (**RAPF**).



(a) Fantasy football: AlgRAPF vs RandAlgRAPFvs OptRA vs FairILP [87]



(b) MovieLens: AlgRAPF vs RandAlgRAPFvs OptRA vs FairILP [87]

Figure 6.7 Kemeny Distance **RAPF**.

Table 6.7 Approximation Factors of the Algorithms for IPF/RAPF

Number of items	10	15	20	25	30
GrBinaryIPF (Football)	1.0	1.0	1.0	1.0	1.0
ApproxMultiValuedIPF (MovieLens)	1.52	1.46	1.37	1.33	1.30
ApproxMultiValuedIPF (Credit Score)	1.8	1.76	1.60	1.57	1.52
AlgRAPF (Football)	2.86	2.76	2.15	2.14	2.01
AlgRAPF (MovieLens)	1.90	1.21	1.18	1.11	1.10
RandAlgRAPF (Football)	2.98	2.77	2.15	2.13	2.06
RandAlgRAPF (MovieLens)	2.10	1.71	1.6	1.70	1.60

MultiValuedIPF Results We use the MovieLens and German Credit Score datasets to demonstrate the effectiveness of our proposed solution ApproxMultiValuedIPF and compare it with DetConstSort. Figures 6.1(b), 6.1(c), 6.2(b), and 6.2(c) demonstrate that also in this case ApproxMultiValuedIPF consistently satisfies p-fairness whereas DetConstSort fails to satisfy p-fairness. Figures 6.3(b), 6.3(c) compares the Kendall-Tau distance between the input ranking and the ranking computed by ApproxMultiValuedIPF and DetConstSort. Again, at times DetConstSort computes a ranking with a smaller distance since DetConstSort does not necessarily compute a p-fair ranking.

Figures 6.4(b), 6.4(c) plot the Kendall-Tau distance of the rankings by ApproxMultiValuedIPF, as we relax the p-fairness using $\delta \geq 0$. Unsurprisingly, for large δ, the Kendall-Tau values become 0.

RAPF Results Next, we evaluate the **RAPF** problem by studying the effectiveness of our proposed AlgRAPF using GrBinaryIPF (Fantasy football) and

Table 6.8 Case Study Results on MovieLens Dataset

Movie	User1	User2	User3	User4	User5	OptRAPF	FairILP	Genre
Bad News Bears, The (1976)	9	7	7	7	4	7	3	Comedy
True Grit (2010)	7	5	1	9	3	9	6	Western
My Darling Clementine (1946)	2	3	3	3	10	4	4	Western
Last Picture Show, The (1971)	4	1	5	1	5	5	1	Drama
Man with the Golden Arm, The (1955)	6	8	4	10	6	8	10	Drama
Heaven Can Wait (1978)	10	10	8	8	8	10	9	Comedy
Rio Bravo (1959)	1	4	6	5	7	1	5	Western
Elephant Man, The (1980)	5	2	2	4	2	6	2	Drama
Buddy Holly Story, The (1978)	3	6	10	6	9	2	8	Drama
Animal House (1978)	8	9	9	2	1	3	7	Comedy

ApproxMultiValuedIPF (MovieLens), and compare it with FairILP [87] and OptRAPF, whenever appropriate.

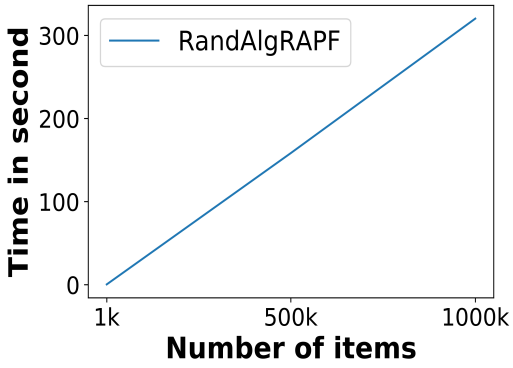
Figures 6.6(a) and 6.6(b) demonstrate that AlgRAPF consistently satisfies p-fairness whereas FairILP fails to satisfy p-fairness. Figures 6.7(a) and 6.7(b) compare the Kemeny distance between the input rankings and the aggregate ranking produced by AlgRAPF, RandAlgRAPF, FairILP, and OptRA. As expected OptRA achieves the smallest distance, followed by FairILP, since it does not require p-fairness, and then AlgRAPF and RandAlgRAPF. Algorithm RandAlgRAPF is inferior to AlgRAPF in practice, since its performance is same as the latter one only in expectation.

Figures 6.9(a) and 6.9(b) plot the Kemeny distance of the ranking computed by OptRA, AlgRAPF, RandAlgRAPF as we relax the p-fairness using $\delta \geq 0$ values. Unsurprisingly, with large δ, our algorithms become very close to OptRA.

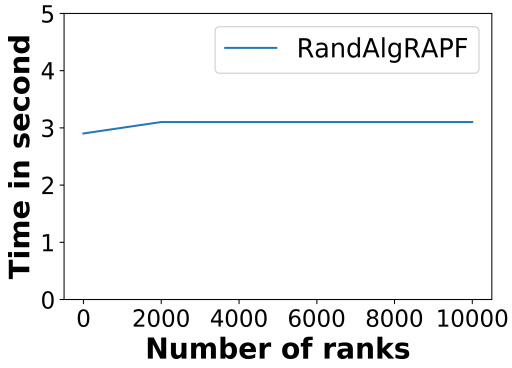
Finally, Table 6.7 presents the actual approximation factors of the different algorithms proposed in this work. Because of the exponential nature of the OptIPF this comparison could be conducted only on small datasets. As evident from the table the actual approximation factors are lower than the proven theoretical bounds.

6.6.5 Case Study

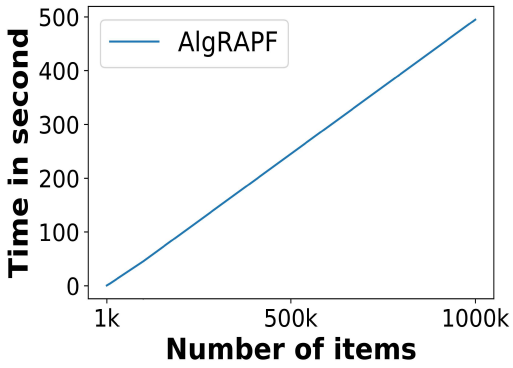
For the case study we use the 10 popular movies based on 5 different IMDB users. All these movies belong to 3 different genres (protected attribute): Drama, Western, Comedy. The proportion of these genres are 0.4, 0.3, and 0.3, respectively. The last two columns of the table 6.8 show the ranked order of the results based on FairILP [87] and our proposed OptRAPF, respectively. It is easy to notice that compared to FairILP, OptRAPF ranks the movies in a manner where different genres are proportionally distributed in all 10 ranked positions, thereby promoting improved user experience.



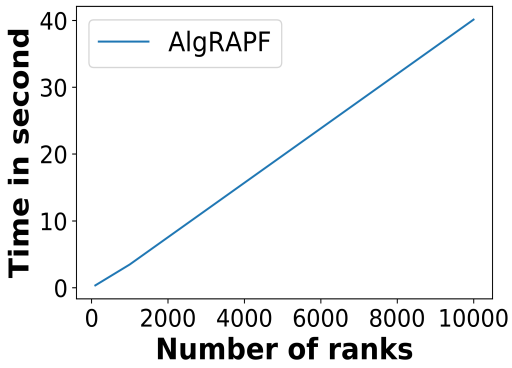
(a) Vary n, $m = 100$:RandAlgRAPF



(b) Vary m, $n = 1000$: RandAlgRAPF

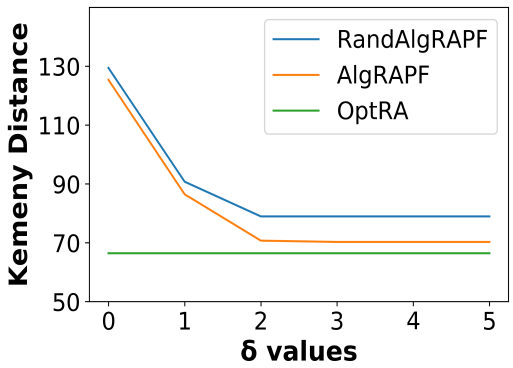


(c) Vary n, $m = 100$:AlgRAPF

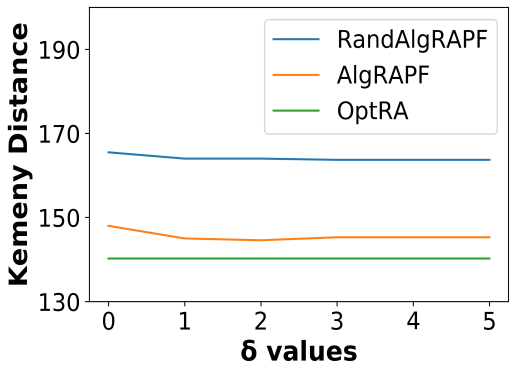


(d) Vary m, $n = 1000$:AlgRAPF

Figure 6.8 Running time analysis



(a) Fantasy Football : AlgRAPF vs RandAlgRAPF varying δ



(b) Movie Lens : AlgRAPF vs RandAlgRAPF varying δ

Figure 6.9 Varying δ analysis RAPF.

6.6.6 Scalability Experiment

We present the running times of RAPF, RandAlgRAPF, GrBinaryIPF, ApproxMultiValuedIPF. We do not present these results wrt any other baselines because of two reasons: first, we have shown that the baselines DetConstSort [72] and FairILP [87] do not satisfy the p-fairness criteria; second, the baseline algorithm FairILP [87] is inherently not scalable. We use synthetically generated data using Mallows' model for this purpose. We vary n and m. Figures 6.5, and 6.8 show these results and demonstrate that our solution easily scale to 1 million items (n) and $10,000$ ranks (m). These results also corroborate our theoretical analysis and shows that the running time of RandAlgRAPF is not dependent on m.

6.7 Conclusion and Future Work

We propose the **RAPF** problem to incorporate a group fairness criteria (p-fairness) considering binary and multi-valued protected attributes with the classical rank aggregation problem. We first study how to produce a p-fair ranking that is closest to a single input ranking (**IPF**). **IPF** can be solved exactly using a greedy technique when the protected attribute is binary. When the protected attribute is multi-valued such an approach fails. We then present two solutions for multi-valued

IPF, ExactMultiValuedIPF is optimal and ApproxMultiValuedIPF admits 2 approximation factor. Next, we design two computational frameworks to solve **RAPF**: RandAlgRAPF and AlgRAPF that exhibit 3 and 4 approximation factors when designed using ExactMultiValuedIPF and ApproxMultiValuedIPF, respectively. The effectiveness of our proposed solutions is demonstrated by comparison to state-of-the-art solutions using multiple real world and large scale synthetic datasets.

Our work opens up several interesting research directions.

A. Alternative models. There exist alternative ways to incorporate p-fairness inside rank aggregation. As an example, one can study the problem of minimizing "weighted" Kemeny distance where the weights are derived considering p-fairness criteria. A slightly different problem is to ensure proportionate fairness not on every position, but for every x (given as input) positions. This problem would be important in applications where every x consecutive individuals in a ranked order are eligible to get the same preferable outcome (such as, top-5% of employees get 100% bonus of their base salary, etc). Studying **RAPF** considering Spearman's Footrule remains part of our ongoing investigation.

B. RAPF for Top-k or considering incomplete information. We are studying how to adapt **RAPF** to produce only top-k aggregated rank. This will require us to adapt Kendall-Tau and Kemeny Optimization for top-k results. One possible approach is to consider all items in the individual rank starting at place $k+1$ as ties, and generalize Kemeny based on ties [7, 63]. We are also interested to study how to obtain an aggregate p-fair ranking when each member inputs only a partial ranking [7, 63].

C. Hardness of IPF. We note that **IPF** essentially finds a perfect matching in a convex bipartite graph while minimizing crossings. The problem of minimizing the number of crossings in a (geometric) bipartite matching is known to be NP-Hard

for general bipartite graphs [1]. For convex bipartite graphs, we currently explore if and how existing works that aim at finding a maximum matching without any crossing [47, 96] can adapt to crossing minimization of a perfect matching.

CHAPTER 7

SATISFYING COMPLEX TOP-K FAIRNESS CONSTRAINTS BY PREFERENCE SUBSTITUTIONS

7.1 Introduction

Preference aggregation is important in finding top-k outputs that represent plurality preference [103] and has wide variety of applications in recommender systems, search results listing [122], electoral systems [90, 104], or allocating resources among candidates, such as, in hiring or admission [148]. A natural variant of the top-k preference aggregation problem is defined as follows: given m users (voters) and n items (candidates), each user (voter) casts her preference for a single item (candidate) as a ballot, and the k items (candidates) from the n that have the highest number of preferences are selected. However, this variant may not produce a desired outcome when applications need to promote fairness by ensuring proportionate representation of the items (candidates) in the top-k results based on their protected attributes. For example, given race as a protected attribute with values, Asian vs. Pacific Highlander vs. White, a proportionate representation of each of the values must be present in the top-k results. We study how to guarantee fairness by *single ballot substitutions*, where each such substitution replaces a vote for an item (candidate) i by a vote for an item (candidate) j.

Our goal in this work is to optimize preference substitution to satisfy complex top-k fairness constraints, where the fairness requirement is defined over a set R of protected attributes. The objective is to *minimize the number of single ballot substitutions that guarantee fairness in the top-k results*. Borrowing terminology from Election Theory [43], we define this minimum number of single ballot substitutions as the *margin*. To the best of our knowledge we are one of the first to propose

a systematic study of the *margin finding problem via single ballot substitutions considering multiple protected attributes* **(Section 7.6 contains details)**.

Our first contribution is to formalize several variants of the *margin finding problem via single ballot (preference) substitutions* considering complex fairness constraints **(Section 7.2)**. (i) In MFBinaryS, proportionate representation is required over a single binary protected attribute, such as male and female of the protected attribute gender; (ii) In MFMultiS, it is defined over a single multi-valued protected attribute, such as, race that contains more than two different values; (iii) Contrarily, in MFMulti2, proportionate representation is required over two different protected attributes, such as gender and race; and finally, (iv) in MFMulti3+, we study the *margin finding problem via preference substitutions* considering three or more protected attributes, such as, race, gender, and ethnicity.

Our second contribution is to study the defined problems theoretically and make principled algorithmic contributions **(Sections 7.3 and 7.4)**. We prove that both MFBinaryS and MFMultiS are computationally easy, i.e., finding margin is polynomial time solvable and we design exact algorithms Alg1AttBOpt and Alg1AttMOpt for both these variants that run in $O(n \log n)$. Next, we consider MFMulti2 and MFMulti3+ in which two or more attributes are involved in defining fairness requirement. To the best of our knowledge we are the first to study these problems rigorously from computational standpoint. Clearly, a trivial solution is to take a Cartesian product over the attribute values, enumerate over all combinations of possible values of the cells in the Cartesian product, and find the margin for each such combination by converting the requirement to a single multi-valued protected attribute. However, if the domain size of the involved protected attributes are not constant, the Cartesian product may create an exponential number of possible combinations for the converted multi-valued protected attribute, making the process computationally intractable. When there are two different protected attributes

involved in outlining the fairness requirement, we prove that the decision version of that problem, i.e., MFMulti2, is (weakly) NP-hard by reducing the well known NP-hard Partition problem to our problem [70]. We design an efficient algorithm Alg2AttApx that obtains a 2 approximation factor and runs in $O(n^2\ell \log m)$ time, by casting this problem as a min cost flow problem, where ℓ is the total number of possible attribute values. Finally, for MFMulti3+, we prove that the satisfiability problem itself is (strongly) NP-hard through a reduction from the 3-dimensional matching (3DM) problem [70]. Namely, it is NP-hard just to decide whether there exists a feasible solution that satisfies the fairness requirement defined over those 3 or more attributes.Our technical results are summarized in Table 7.1.

Our final contribution is experimental **(Section 7.5)**. We conduct rigorous large scale experiments involving 3 real world (involving election and movie applications) and one synthetic datasets and compare multiple state-of-the-art solutions [66, 129] after appropriate adaptation. Despite non-trivial adaptation, these related works fail to optimize margin values and do not turn out to be effective choices. Our experimental results corroborates our theoretical analysis, the designed algorithms match theoretical guarantees qualitatively, and demonstrate to be highly scalable. **Section 7.7** discusses some extensions of the problems, and we conclude in **Section 7.8**.

7.2 Data Model & Problem Definitions

In this section, we describe the data model and illustrate that with a running example, following which we define the studied problems.

7.2.1 A Toy Running Example

Table 7.2 describes the ballots of 12 voters and the outcome of a voting process with 6 candidates (C1,C2,C3,C4,C5, C6). For example, V1, V2, V4 and V7 vote for candidate C1, and C1 becomes the top candidate with 4 votes. Each candidate has

Table 7.1 Summary of Technical Results

Problem	**Protected Attribute**	**Hardness**	**Algorithm**	**Approx Factor**	**Running Time**
MFBinaryS	single attribute binary valued	p-time	Alg1AttBOpt	exact	$\mathcal{O}(n \log n)$
MFMultiS	single attribute multi (ℓ) valued	p-time	Alg1AttMOpt	exact	$\mathcal{O}(n \log n)$
MFMulti2	2 attributes ℓ possible values	Weak NP-hard	Alg2AttApx	2	$\mathcal{O}(n^2 \ell \log m)$
MFMulti3+	3+ attributes	NP-hard			
MFMulti2 MFMulti3+	2+ attributes const size (c) of Cartesian prod	p-time	AlgCartOpt	exact	$\mathcal{O}(n^{c+1} \log n)$

Table 7.2 12 Voters, 6 Candidates, and a Voting Outcome

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	$\sum V_i$
C1 (M,Sr,si)	1	1	0	1	0	0	1	0	0	0	0	0	4
C2 (M,Jr,si)	0	0	1	0	1	0	0	0	1	0	0	0	3
C3 (M,Jr,ma)	0	0	0	0	0	1	0	0	0	0	0	1	2
C4 (F,Jr,si)	0	0	0	0	0	0	0	0	0	1	1	0	2
C5 (F,Jr,ma)	0	0	0	0	0	0	0	1	0	0	0	0	1
C6 (F,Sr,di)	0	0	0	0	0	0	0	0	0	0	0	0	0

three protected attributes: **Gender** *(M, F),* **Seniority Level** *(Senior and Junior, abbreviated as Sr and Jr, respectively), and* **Marital status** *(Married, Single, and Divorced, abbreviated as ma, si, and di, respectively.*

An Example Complex fairness constraint. Imagine the goal is to select $k = 4$ candidates from the voting outcome described in Table 7.2 with the following fairness constraints described in Table 7.3.

Table 7.3 Fairness Constraints in Top-4 Results of Running Example

Attribute	**Value**	**Fairness constraint**
Gender	M	2
	F	2
Seniority Level	Sr	2
	Jr	2
Marital Status	ma	2
	si	1
	di	1

Table 7.4 Table of Notations

Notation	**Meaning**
n, m, k	#candidates, #voters,#results
A_i	protected attribute
ℓ_i	#values of a protected attribute A_i
L_C	list of candidates
L_V	respective list of number of votes
t	threshold
$a_*(t)$	#candidates form group G_A with at least t votes
C, c	set of candidates, $\Pi_{i=1}^{\ell} \ell_i$

Preference Elicitation and Aggregation. Each user (voter) casts her top-1 preference (vote) through a ballot, and the k items (candidates) who get the highest number of votes are elected[1]. In Section 7.7, we discuss an extension of this settings.

Database. The database contains the outcome of a voting process based on the top-1 preference of m voters over n candidates. The set of candidates will be denoted as C, individual candidate will be denoted by i and j. Considering the running example, $m = 12$ voters provide preferences over a set of $n = 6$ candidates, and the aggregated preference is shown in Table 7.2.

[1]For the remainder of the paper, users and voters are synonymous, as well as items and candidates are used interchangeably.

Note that the outcome may not be unique, and there may be more than one set of k candidates who get the highest number of votes. We refer to such a situation as a *tie*. A *reasonable* tie breaking is one in which none of the k elected candidates have received less votes than any non-elected candidate.

Protected Attribute. Each candidate has one or more *protected attribute*, where each protected attribute A_i can take any of ℓ_i different values. When $\ell_i = 2$, it is a binary protected attribute; when $\ell_i \geq 2$ it is a multi-valued protected attribute. As an example, the attributes Marital Status and Gender are multi-valued and binary protected attributes, respectively.

Top-k [75, 87, 129] Fairness Constraints. When fairness is defined over a single protected attribute containing ℓ different groups G_1, G_2,.,G_ℓ, such that the required representation of each group G_i is a_i, then the top-k is a fair top-k, if $\sum_i^\ell a_i = k$. Generalizing this, if fairness is defined over a set R of different protected attributes with a required representation on each group of each attribute, a fair top-k result must simultaneously satisfy proportionate representation for all attributes in R.

One such complex fairness constraint is described using Table 7.3. Based on this, $\{C1, C3, C5, C6\}$ is a feasible top-4 outcome, as it satisfies all these requirements.

7.2.2 Problem Definitions

Definition 7. *Given two candidates i and j, a **single ballot substitution** is defined as removing one vote from candidate i and assigning it to candidate j; thus, after the ballot change, the number of votes obtained by candidate i is decreased by one, and the number of votes obtained by candidate j is increased by one.*

Problem 5. MFBinaryS. **Margin Finding for a Single Binary Protected Attribute.** *Given a protected attribute A with $\ell = 2$ different protected groups, an outcome of a voting process, and a fairness constraint that requires to have a_1*

candidates from group G_1 and a_2 candidates from group G_2 in the top-k, with $a_1+a_2 = k$, find the margin that guarantees a fair outcome.

Using Example 7.2, consider a fairness constraint defined over the binary protected attribute Gender, such that, $a_M = a_F = 2$. The top-4 (C1,C2,C3,C4) candidates consist of 3 males and 1 female. To satisfy the fairness constraint, one can remove a single vote from C3 and assign that it to C5. After the substitution, C3 and C5 will have $2 - 1 = 1$ and $1 + 1 = 2$ votes, respectively. The resulting top-4 (C1,C2,C4,C5) satisfies the fairness constraint and the margin is 1.

Problem 6. MFMultiS. **Margin Finding for a Single Multi-valued Protected Attribute.** *Given a protected attribute A with $\ell > 2$ different protected groups, an outcome of a voting process, and a fairness constraint that requires for every $i \in [1..\ell]$, to have a_i candidates from group G_i in the top-k, with $\sum_i^\ell a_i = k$, find the margin that guarantees a fair outcome.*

Consider Table 7.2 again with Marital Status as the multi-valued protected attribute, with $\ell = 3$. Consider a top-4 fairness constraint such that, $a_{ma} = 2 \wedge a_{si} = 1 \wedge a_{di} = 1$. The top-4 candidates (C1,C2,C3,C4) consist of 1 married and 3 single candidates. To satisfy the fairness constraint, remove two votes from C2 and one vote from C4 and assign one vote to C5 and two votes to C6. After the substitutions the votes of candidates C2, C4, C5, and C6 become 1, 1, 2, 2, respectively. The resulting top-4 (C1,C3,C5,C6) satisfies the fairness constraint. In this case, the margin is 3.

Problem 7. Margin Finding over Multiple Protected Attributes. *Given a set $R = \{A_1, \ldots, A_{|R|}\}$ of protected attributes, where attribute A_i has ℓ_i different protected groups, an outcome of a voting process, and fairness constraints that require for every $i \in [1..|R|]$, and $j \in [1..\ell_i]$ to have $a[i,j]$ candidates from group G_j of attribute A_i in the top-k, with $\sum_j^{\ell_i} a[i,j] = k$, for $i \in [1..|R|]$, find the margin that guarantees a fair outcome.*

MFMULTI2. **Margin Finding for two Protected Attributes.** When $|R| = 2$, this problem instantiates to finding the margin when the fairness constraints are defined over two different attributes.

Consider Table 7.2 again, and let R consist of the two attributes Gender and Seniority level. The top-4 fairness constraints are as follows: $a_M = 2 \wedge a_F = 2 \wedge a_{Si} = 2 \wedge a_{Jr} = 2$. The top-4 candidates (C1,C2,C3,C4) consist of 1 female and 3 male candidates, and 1 senior and 3 junior candidates. To satisfy the fairness constraints, remove two votes from candidate C3 and assigning them to candidate C6. After the ballot substitutions, C3 has 0 votes, and C6 has 2 votes. The resulting top-4 candidates C1, C2, C4, and C6 with 4, 3, 2, 2 votes, respectively, satisfy the fairness constraints. It is easy to verify that a fair outcome cannot be obtained by performing a single substitution. Thus, in this case, the margin is 2.

MFMULTI3+. **Margin Finding for More than two Protected Attributes.** When $|R| \geq 2$, this problem instantiates to finding the margin when the fairness constraints are defined over three or more different attributes.

Consider Table 7.2 again and the fairness constraint presented in Table 7.3. To satisfy the fairness constraints, perform 3 single ballot substitutions, by removing 2 votes from C2 and 1 vote from C4 and assigning 2 votes to candidate C6 and 1 vote to C5. After the substitutions the votes of candidates C2, C4, C5, and C6 are 1, 1, 2, 2, respectively. The resulting top-4 (C1,C3,C5,C6) with votes 4, 2, 2, 2 satisfy the fairness constraints. It is easy to verify that a fair outcome cannot be obtained by performing less than 3 substitutions. Thus, in this case, the margin is 3.

7.3 Single Protected Attribute

In this section, we study two problems, namely MFBINARYS and MFMULTIS, the first one finds margin via single ballot substitutions, when the fairness constraint is

defined by a single binary protected attribute, and the second one does so when the fairness constraint is defined by a single multi-valued protected attribute.

7.3.1 Binary Protected Attribute

The algorithm Alg1AttBOpt finds an exact solution to MFBinaryS. The input to the problem is an initial vote outcome, and a fairness constraint defined by a single binary protected attribute. The binary attribute partitions the candidates into two groups G_A and G_B. The fairness constraint requires that the top-k consists of a candidates from G_A and b candidates from G_B, where $k = a + b$. The initial vote outcome is represented by two lists, L_C - the list of candidates and L_V - the respective list of the number of votes casted to each candidate. We sort both lists in non increasing order of number of votes, implying that, $L_C(1)$ is a candidate with the most number of votes $L_V(1)$, and so on. The output is, B, a set of ballot substitutions of minimum size that guarantees a fair outcome (or guarantees, in case of a tie, that all outcomes that can be produced by a reasonable tie breaking are fair).

Algorithm Alg1AttBOpt as well as subsequent algorithms use the notion of a *threshold* of an election outcome defined as follows.

Definition 8. *The* threshold *of an election outcome is the number of votes, denoted by t, such that each of the top-k candidates got at least t votes and at least one such candidate got exactly t votes.*

Using running example, $G_{Sr} = \{C1, C6\}$, $G_{Jr} = \{C2, C3, C4, C5\}$, $L_C = [C1, C2, C3, C4, C5, C6]$, $L_V = [4, 3, 2, 2, 1, 0]$. For $k = 4$, threshold is $t = 2$ where all top-4 candidates got at least 2 votes and both C3 and C4 got exactly 2 votes. We note that for the original election outcome the threshold is $L_V(k)$, and that in case of a tie any reasonable outcome will have the same threshold.

The core of Alg1AttBOpt is a subroutine FindBallotSubB that finds for a given threshold t the minimum number of single ballot substitutions that guarantee

a fair outcome. This subroutine is invoked repeatedly inside Alg1AttBOpt to efficiently search for the margin, obtained by finding the threshold that achieves the minimum number of single ballot substitutions that guarantee a fair outcome over all possible thresholds. The pseudo code of FindBallotSubB and Alg1AttBOpt is shown in Algotithms 12 and 13.

Let i_a be the index in L_C of the a-th candidate from G_A. That is, $L_C(i_a) \in G_A$ and the number of candidates from G_A in $L_C(1), \ldots, L_C(i_a)$ is exactly a. Similarly, let i_b be the index of the b-th candidate from G_B in L_C. Below, we assume that $i_a < i_b$ and thus $L_V(i_a) \geq L_V(i_b)$. The other case is symmetric. In Lemma 13 we prove that the optimal threshold t must be in the interval $[L_V(i_b), L_V(i_a)]$. Thus, from now on we just consider this interval.

For any threshold t in the open interval $(L_V(i_b), L_V(i_a))$ the optimal set of ballot additions and removals is determined in Case 3 of subroutine FindBallotSubB (described below). For a specific t, ballots are added to candidates in group G_B and removed from candidates in group G_A. Later we show how to replace the vote additions and removals by single ballot substitutions. The number of these single ballot substitutions is the maximum between the number of vote additions and vote removals.

The number of votes subtracted from candidates in G_A declines as t grows in this interval and the number of votes added to candidates in G_B grows as t grows in this interval. The optimal t can thus be found using binary search. We can make the binary search even more efficient, and instead of doing it on the interval $(L_V(i_b), L_V(i_a))$ which may be $\Omega(m)$ we can do it on the interval (i_a, i_b). After completing this binary search we identify an index $i \in (i_a, i_b)$, such that the optimal threshold is in the interval $[L_V(i), L_V(i{+}1))$. Later we show how the optimal threshold in this interval can be computed in constant time.

Algorithm 12 FindBallotSubB

Inputs: t, L_V, L_C, a, b, i_a, i_b

Outputs: S = number of ballot Substitutions

1: Calculate $a_*(t), b_*(t)$

2: $I_a = \{(a_*(t) + b_*(t) + 1), \ldots, i_b\}$

3: $B_a = t(b - b_*(t)) - \sum_{i \in I_a \ \& \ L_C(i) \in G_B} L_V(i)$

4: $I_r = \{(i_a + 1), \ldots, (a_*(t) + b_*(t))\}$

5: $B_r = \sum_{i \in I_r \ \& \ L_C(i) \in G_A} L_V(i) - (t-1)(a_*(t) - a)$

6: $S = \max\{B_a, B_r\}$

7: Return S

7.3.2 Subroutine FindBallotSubB

Given a threshold t, FindBallotSubB finds the minimum number of single ballot substitutions that result in a fair outcome with this threshold. For simplicity we first assume that the fair outcome does not have a tie. Later, we show how to remove this assumption.

Let $a_*(t)$ and $b_*(t)$ be the number of candidates from groups G_A and G_B respectively who received at least t votes. Note that $L_V(a_*(t) + b_*(t)) = t$ and $L_V(a_*(t) + b_*(t) + 1) < t$.

This subroutine is designed by distinguishing the following cases.

Case 1: $t \leq L_V(i_b)$ (and $L_V(i_b) > 0$). In this case the numbers of candidates from groups G_A and G_B who got at least t votes are at least a and b, respectively; namely, $a_*(t) \geq a$ and $b_*(t) \geq b$. We decrease the number of votes of the $a_*(t) - a$ candidates from G_A in $L_C(i_a + 1), \ldots, L_C(a_*(t) + b_*(t))$ and the number of votes of the $b_*(t) - b$ candidates from G_B in $L_C(i_b + 1), \ldots, L_C(a_*(t) + b_*(t))$ to $t - 1$. To reconcile for the decrease of these votes, we add votes of the candidate in $L_C(1)$.

Case 2: $t > L_V(i_a)$. In this case the numbers of candidates from groups G_A and G_B who got at least t votes are less than a and b, respectively; namely, $a_*(t) < a$ and

Algorithm 13 ALG1ATTBOPT

Inputs: L_V, L_C, a, b

Outputs: M = minimum number of ballot substitutions

1: Calculate i_a, i_b

2: S_a = num of single ballot substitution for threshold $L_V(i_a)$

3: S_b = num of single ballot substitution for threshold $L_V(i_b)$

4: **Binary Search** over all $i \in (i_a, i_b)$

$t = L_V(i)$

S_i = FINDBALLOTSUBB(t,L_V,L_C,a, b, i_a, i_b)

If found i such that M lies in S_i, S_{i+1}; break

5: Calculate M for thresholds in the range $[L_V(i), L_V(i+1)]$

6: Return $\min\{S_a, S_b, M\}$

$b_*(t) < b$. We increase the number of votes of the $a - a_*(t)$ candidates from G_A in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_a)$ and the number of votes of the $b - b_*(t)$ candidates from G_B in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_b)$ to t. To reconcile for the increase, we decrease the number of votes of the candidates from G_A in $L_C(i_a + 1), \ldots, L_C(n)$ and the number of votes of the candidates from G_B in $L_C(i_b + 1), \ldots, L_C(n)$ to 0, as needed. If this is not enough we can decrease the number of votes of the candidates in $L_C(1), \ldots, L_C(a_*(t+1) + b_*(t+1))$ to t, as needed. Note that for this case to be feasible we must have $n \geq k \cdot t$.

Case 3: $L_V(i_b) < t \leq L_V(i_a)$. In this case the number of candidates from group G_A who got at least t votes is at least a and the number of candidates from group G_B who got at least t votes is less than b; namely, $a_*(t) \geq a, b_*(t) < b$. We increase the number of votes of the $b - b_*(t)$ candidates from G_B in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_b)$ to t. Then, we decrease the number of votes of the $a_*(t) - a$ candidates from G_A in $L_C(i_a + 1), \ldots, L_C(a_*(t) + b_*(t))$ (if such exist) to $t - 1$. Finally, one has to reconcile the increase in the votes of the candidates from G_B with the decrease in the votes of

the candidates from G_A. If this is not enough, further reconciliation is done similar to the previous two cases. Note again that for this case to be feasible, one must have $n \geq k \cdot t$.

Using the running example, consider the binary attribute Seniority Level and $a = 2,\ b = 2$. We have $i_{Jr} = 3, i_{Sr} = 6, L_V(i_{Jr}) = 2, L_V(i_{Sr}) = 0$. Consider a threshold, $t = 1$ then $a_*(1) = 4$ and $b_*(1) = 1$. To satisfy fairness constraint, one can reduce votes of $a_*(1) - a = 4 - 2 = 2$ junior candidate to $t - 1 = 1 - 1 = 0$. When these two candidates are C4 and C5, the minimum ballot reduction $2 - 0 + 1 - 0 = 3$ is obtained for this threshold $t = 1$. Similarly, to obtain fairness, votes of $b - b_*(1) = 2 - 1 = 1$ senior candidate has to be increased to $t = 1$. The minimum ballot increase will occur when candidate C6 vote is increased from 0 to 1. To reconcile the 3 ballots that were removed from C4 and C5, one vote is matched to vote added to candidate C6 and 2 of them are matched to two votes added to candidate C1. After the ballot substitution the votes of candidates C1, C2, C3, C4, C5, C6 are $6, 3, 2, 0, 0, 1$ and the candidates who got at least $t = 1$ votes are C1, C2, C3, C6. For threshold $t = 1$, the minimum number of ballot substitution that guarantees fairness is 3.

Converting ballot additions and removals to ballot substitutions. In each of the cases we showed how to reconcile in case there are more vote additions than removals or vice versa. In case there are more vote additions than removals we are guaranteed to achieve the reconciliation only if $n \geq k \cdot t$. This raises the question what if for the optimal threshold t we have $n < k \cdot t$. Suppose that this is the case; that is, even after reducing the number of votes of all the unelected candidates to 0 and reducing the number of votes of the all the elected candidates to the threshold t we still have some unmatched ballot additions. It is easy to see that in this case we have more vote additions than removals, and thus if $t > 1$, reducing the threshold by 1 will reduce the number of vote additions and thus t cannot be optimal. It follows that the only assumption we need to make is for $t = 1$, and in this case we assume

that $n \geq k$. This is a reasonable assumption since we need to elect k and thus we should have more than k ballots.

Finally, we consider the case of a tie. This may be optimal only when the threshold is either $L_V(i_a)$ or $L_V(i_b)$. For threshold $t = L_V(i_a)$ we need to also consider the possibility of increasing the number of votes of the $b - b_*(t+1)$ candidates from G_B in $L_C(a_*(t+1) + b_*(t+1) + 1), \ldots, L_C(i_b)$ to $t+1$. Note that after this increase we may have more than a candidates from G_A with at least t votes, but strictly less than a candidates from G_A with at least $t+1$ votes. On the other hand we have exactly b candidates from G_B with at least $t+1$ votes, but no candidates from G_B with t votes. Thus, we have a tie only if $a_*(t) - a_*(t+1) > a - a_*(t+1)$, and any reasonable way to break such a tie is by varying the subset of size $a - a_*(t+1)$ of elected candidates from G_A with t votes. For threshold $t = L_V(i_b)$ we need to also consider the possibility of decreasing the number of votes of the $a_*(t) - a$ candidates from G_A in $L_C(i_a + 1), \ldots, L_C(a_*(t) + b_*(t))$ to $t - 1$. The analysis is similar to the one given above.

Using the running example, consider the binary attribute Seniority Level with G_A and G_B the Junior and Senior groups, and $a = 2, b = 2$. At threshold $t = 2$, there is a tie situation for group junior because $a_*(2) - a_*(3) = 3 - 1 = 2 > a - a_*(3) = 2 - 1 = 1$. One way of achieving fairness is to increase the votes of candidate C6 from 0 to $t + 1 = 2 + 1 = 3$. After the increase there are 3 junior candidates with votes at least 2 and there is no senior candidate with exactly 2 votes.

Running Time. The running time of Alg1AttBOpt is determined by the subroutine FindBallotSubB. We precompute $a_*(t), a_*(t)$, for $t \in (i_a, i_b)$ in $\mathcal{O}(n)$ time. We can also precompute required ballot additions and removals for t $in(i_a, i_b)$ which also requires $\mathcal{O}(n)$. As a result Subroutine FindBallotSubB takes constant time. This subroutine is called $\mathcal{O}(\log n)$ times in Alg1AttBOpt. Overall running

time is $\mathcal{O}(n + \log n) = \mathcal{O}(n)$. The time complexity is dominated by the $\mathcal{O}(n \log n)$ time it takes to sort the lists L_C and L_V.

Lemma 10. ALG1ATTBOPT *always produces a fair outcome.*

Proof. Consider Case 3, where $L_V(i_b) < t \leq L_V(i_a)$. Before the substitution, the number of candidates who got at least t votes were $a_*(t)$ and $b_*(t)$ from groups G_A and G_B respectively. After the substitution, the number of candidates who got at least t votes from group G_B increased by $b - b_*(t)$, and from G_A decreased by $a_*(t) - a$. The total number of candidates from G_B who got at least t votes $= b_*(t) + (b - b_*(t)) = b$. The total number of candidates from G_A who got at least t votes $= a_*(t) - (a_*(t) - a) = a$. The total number of candidates from both G_A and G_B who got at least t votes $= a + b = k$. Hence, candidates who got at least t votes constitute the top-k results and the top-k has a and b candidates from group G_A and G_B respectively. In this case, top-k satisfies fairness constraint. Similarly, we can prove that our algorithm produces a fair outcome for the other two cases, and it still produces a fair outcome when there is a tie. □

Lemma 11. *Any optimal algorithm for finding the minimum number of single ballot substitutions that guarantee fairness can be viewed as a* ALG1ATTBOPT.

Proof. Any optimal algorithm that performs minimum number of single ballot substitution, will output a top-k set having a, b candidates from group G_A, G_B respectively. We can define a threshold t such that, after the substitutions, the number of candidates from G_A who got at least $t + 1$ votes is less than a but the number of candidates from G_A who got at least t votes is equal to or greater than a , and the number of candidates from G_B who got at least $t + 1$ votes is less than b but the number of candidates from G_B who got at least t votes is equal to or greater than b. Any optimal algorithm is essentially finding a threshold t that requires minimum number of ballot substitutions. □

Lemma 12. *For a given threshold t, subroutine* FINDBALLOTSUBB *returns a minimum number of ballot substitutions to satisfy fairness constraint.*

Proof. At first, we show that we are doing a minimum number of additions and a minimum number of removals for a given threshold t. Consider Case 3, to achieve fairness we need to reduce votes of $a - a_*(t)$ candidates who already got t votes from group G_A to $t-1$. Algorithm decreases the number of votes of the $a_*(t) - a$ candidates from G_A in $L_C(i_a + 1), \ldots, L_C(a_*(t) + b_*(t))$ to $t - 1$. This is the minimum number of vote removals to satisfy a candidates in the top-k. Because if we reduce votes of candidates who are not in the range of candidates $L_C(i_a + 1), \ldots, L_C(a_*(t) + b_*(t))$, it will either produce unfair result or result will not be minimum. If we reduce votes

of candidates from G_A in $L_C(1), \ldots, L_C(i_a)$, then the number of vote removals is not minimum because all candidates in that range have higher votes than all the candidates in $L_C(i_a+1), \ldots, L_C(a_*(t)+b_*(t))$. We can not reduce votes of candidate from G_A in $L_C(a_*(t)+b_*(t)+1), \ldots, L_C(n)$ to $t-1$, because they got less than t votes. Similarly, to achieve fairness we need to increase votes of $b_*(t)-b$ candidates who got less than t votes from group G_B to t. We can show that number of vote additions is minimized when we add votes from G_B in $L_C(a_*(t)+b_*(t)+1), \ldots, L_C(i_b)$. As the number of vote substitutions is the maximum of vote additions and vote removals, for a given threshold t, subroutine FindBallotSubB returns the minimum number of ballot substitutions that guarantee fairness in Case 3. Similarly, we can prove the same statement for the other 2 cases and when there is a tie. □

Lemma 13. *The optimal threshold is in interval* $[L_V(i_b), L_V(i_a)]$.

Proof. Consider a threshold $t > L_V(i_a)$, to satisfy fairness, the number of votes of the $a - a_*(t)$ candidates from G_A in $L_C(a_*(t)+b_*(t)+1), \ldots, L_C(i_a)$ and the number of votes of the $b - b_*(t)$ candidates from G_B in $L_C(a_*(t)+b_*(t)+1), \ldots, L_C(i_b)$ need to be increased to at least t. On the other hand vote removals are not needed. It follows that the number of ballot substitutions equals the total number of ballot additions. Clearly, the number of vote additions required to guarantee fairness in case the threshold is $L_V(i_a)$ is lower, and thus t cannot be optimal. Similarly, for threshold $t < L_V(i_b)$, the number of vote removals required to guarantee fairness is more than this number when the threshold is $L_V(i_b)$. Hence, the optimal threshold t must be in the interval $[L_V(i_b), L_V(i_a)]$. □

Lemma 14. *The minimum number of ballot additions to G_B increases and the minimum number of ballot removals from G_A decreases monotonically with t in the interval* $[L_V(i_b), L_V(i_a)]$.

Proof. We get minimum number of ballot additions when we increase votes of the $b-b_*(t)$ candidates from G_B in $L_C(a_*(t)+b_*(t)+1), \ldots, L_C(i_b)$ to t. When t increases, both $b - b_*(t)$ and distance from t to votes of candidates from G_B in $L_C(a_*(t) + b_*(t)+1), \ldots, L_C(i_b)$ increases. Hence, the minimum number of ballot additions to G_B increases monotonically with t in the interval $[L_V(i_a), L_V(i_b)]$. Similarly, we can prove that the minimum number of ballot removals from G_A decreases monotonically with t in the interval $[L_V(i_a), L_V(i_b)]$.

□

Theorem 27. Alg1AttBOpt *produces optimal result.*

Proof. We proved that for a given threshold t, FindBallotSubB calculates minimum ballot substitutions required to satisfy fairness. Optimal threshold t is in the interval of $[L_V(i_b), L_V(i_a)]$. Since the minimum number of ballot additions

to G_B increases and the minimum number of ballot removals from G_A decreases monotonically with t in the interval $[L_V(i_b), L_V(i_a)]$ the optimal number of ballot substitutions can be found by performing a binary search in the range $[L_V(i_b), L_V(i_a)]$. Hence, Alg1AttBOpt produces optimal result.

□

7.3.3 Multi-valued Protected Attribute

Next we consider a multi-valued protected attribute. Consider an attribute A with ℓ possible values, denoted $A[1], \ldots, A[\ell]$. The fairness constraint requires that the top-k consists of $a[j]$ candidates with attribute value $A[j]$, where $\sum_{j=1}^{\ell} a[j] = k$.

We first describe the subroutine FindBallotSubM for multi valued attribute. Then we use it to perform a binary search for the optimal threshold similar to Alg1AttBOpt. For a given threshold t, the subroutine FindBallotSubM computes the minimum number of single ballot substitutions that result in a fair outcome. For simplicity we first assume that the fair outcome does not have a tie. Later, we show how to remove this assumption. Define $i_{a[j]}$ as the index in L_C of the $a[j]$-th candidate with attribute value $A[j]$. That is, the candidate $L_C(i_{a[j]})$ has attribute value $A[j]$ and the number of candidates with this attribute value in $L_C(1), \ldots, L_C(i_{a[j]})$ is exactly $a[j]$. Below, we assume that $i_{a[1]} \leq \ldots \leq i_{a[\ell]}$. Other cases are symmetric. Define $a_{j*}(t)$ as the number of candidates with attribute value $A[j]$ who received at least t votes (before any ballot changes). Note that $L_V(a_{1*}(t) + \ldots + a_{\ell*}(t)) = t$ and $L_V(a_{1*}(t) + \ldots + a_{\ell*}(t) + 1) < t$.

Below we distinguish several cases.

Case 1: $t \leq L_V(i_{a[\ell]})$. We decrease the number of votes of the $a_{j*}(t) - a_j$ candidates with attribute value $A[j]$ in $L_C(i_{a[j]} + 1), \ldots, L_C(a_{1*}(t) + \cdots + a_{\ell*}(t))$ to $t - 1$ for all $j \in [1..\ell]$.

Case 2: $t > L_V(i_{a[1]})$. We increase the number of votes of the $a[j] - a_{j*}(t)$ candidates with attribute value $A[j]$ in $L_C(a_{1*}(t) + \ldots + a_{\ell*}(t) + 1), \ldots, L_C(i_{a[j]})$ to t for all $j \in [1..\ell]$.

Case 3: $L_V(i_{a[j+1]}) < t \leq L_V(i_{a[j]})$. We increase the number of votes of the $a[q] - a_{q*}(t)$ candidates with attribute value $A[q]$ in $L_C(a_{1*}(t)+\cdots+a_{\ell*}(t)+1), \ldots, L_C(i_{a[q]})$ to t for $q \in [j+1..\ell]$. We decrease the number of votes of the $a_{p*}(t) - a[p]$ candidates with attribute value $A[p]$ in $L_C(i_{a[p]}+1), \ldots, L_C(a_{1*}(t)+\cdots+a_{\ell*}(t))$ (if such exist) to $t-1$ for all $p \in [1..j]$.

In all three cases, we reconcile the ballot additions and removals to obtain single ballot substitutions the same way it is done in the binary case described previously.

Finally, we consider the case of a tie that may occur when the threshold is any of $L_V(i_{a[j]})$ (or multiple of them). We find the maximum of the number of candidates with the same attribute value who got exactly t votes (after the vote manipulations). Let this attribute value be p. We do not change the votes of these candidates. For the rest of the candidates we do the following. For all candidates with attribute value $A[q]$, for which $q \neq p$ and $t \geq L_V(i_{a[q]})$, we increase the number of votes of the $a[q]-a_{q*}(t+1)$ candidates with attribute value $A[q]$ in $L_C(a_{1*}(t+1)+\cdots+a_{\ell*}(t+1)+1), \ldots, L_C(i_{a[q]})$ to $t+1$. For all candidates with attribute value $A[q]$, for which $q \neq p$ and $t \leq L_V(i_{a[q]})$, we decrease the number of votes of the $a_{q*}(t) - a[q]$ bottom candidates with attribute value $A[q]$ in $L_C(i_{a[q]}+1), \ldots, L_C(a_{1*}(t)+\cdots+a_{\ell*}(t))$ (if such exist) to $t-1$.

Running Time. The running time of Alg1AttMOpt is also dominated by the $\mathcal{O}(n \log n)$ time it takes to sort the lists L_C and L_V as in Alg1AttBOpt. We note that we use to a priority queue to implement FindBallotSubM efficiently. The initialization of this priority queue takes $\mathcal{O}(n)$ time, and each iteration takes $\mathcal{O}(\log \ell)$ time. Thus the overall running time of Alg1AttMOpt (excluding the sorting) is $\mathcal{O}(n + \log n \log \ell) = \mathcal{O}(n)$.

Theorem 28. Alg1AttMOpt *always produces a fair outcome.*

Proof. The proof is similar to the proof of Theorem 27. □

7.4 Multiple Protected Attributes

In this section we assume that there are ℓ attributes, denoted $A_1, \ldots, A_\ell$. For $i \in [1..\ell]$, attribute A_i has ℓ_i possible values, denoted $A[i,j]$, for $j \in [1..\ell_i]$. Each candidate is associated with a specific value from each attribute. In addition, we are given target quantities $a[i,j]$, for $i \in [1..\ell]$, and $j \in [1..\ell_i]$, with property that all marginals some to k. Namely, for every $i \in [1..\ell]$, $\sum_{j=1}^{\ell_i} a[i,j] = k$. A fair election outcome should satisfy the fairness condition that for $i \in [1..\ell]$, and $j \in [1..\ell_i]$, exactly $a[i,j]$ candidates whose A_i attribute value is $A[i,j]$ are elected. The goal is to find the margin, namely the minimum number of ballot substitutions, that guarantees a fair outcome.

We consider the complexity of computing the margin in the multi attribute case. We first show that if the total number of possible values of the ℓ-dimensional attribute vector is constant then the multi attribute case can be reduced to the single attribute case. Next, we consider the general 3 attribute case and show that even deciding the feasibility of a fair outcome is NP-Complete in this case. Then, we consider the 2 attribute case and show that it is *weakly* NP-Complete. (Recall that problem is *weakly* NP-complete if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem and the *magnitudes* of the data involved, provided these are given as integers, rather than the base-two logarithms of their magnitudes.) On the positive side, we show a 2 approximation algorithm for this case by designing an algorithm that minimizes the sum of ballot additions and removals.

7.4.1 Constant Number of Attribute Configurations

We propose AlgCartOpt by converting multiple protected attribute to a single multi-valued attribute. Suppose that $\Pi_{i=1}^{\ell} \ell_i$ is constant c. This means that we have a constant number c of possible values of the ℓ-dimensional attribute vector. For $i \in [1..c]$, let $\vec{V}[i] = V[i,1], V[i,2], \ldots, V[i,\ell]$ be the i-th possible value of ℓ-dimensional

attribute vector. We enumerate over all c-tuples $(n_1, \ldots, n_c)$ such that $\sum_{i=1}^{c} n_i = k$. Each such c-tuple represents a possible outcome of the election in which n_i candidates with attribute vector $\vec{V}[i]$ are elected. For each such c-tuple we first check that it is a feasible outcome by making sure that there are at least n_i candidates with attribute vector $\vec{V}[i]$, for $i \in [1..c]$. If so, we further check if having n_i candidates with attribute vector $\vec{V}[i]$ results in the desired outcome. This is the case if the following is satisfied:

$$\forall\, j \in [1..\ell]\ \forall\, r \in [1..\ell_j] \quad \sum_{i=1}^{c} n_i \cdot \mathbf{1}_{V[i,j]=A[j,r]} = a[j,r]. \tag{7.1}$$

In this case we compute the margin required to guarantee n_i candidates with attribute vector $\vec{V}[i]$, for $i \in [1..c]$, by reducing this to the single attribute case, where the single attribute has c possible values corresponding to the possible values of the attribute vector.

Using the running example, consider the attributes Gender and Marital Status where $\ell_{Gender} = 2$, $\ell_{MaritalStatus} = 3$ and $c = 3 \times 2 = 6$. The required numbers of candidates with each attribute value are $a[M] = 2 \wedge a[F] = 2 \wedge a[ma] = 2 \wedge a[si] = 1 \wedge a[di] = 1$. Here, $V[1] = \{M, si\}$, $V[2] = \{M, si\}$, $V[3] = \{M, ma\}$, $V[4] = \{F, si\}$, $V[5] = \{F, ma\}$, and $V[6] = \{F, di\}$. One of the possible tuples that satisfy fairness is $(n_1, \ldots, n_6) = (1, 0, 1, 0, 1, 1)$ where $\sum_{i=1}^{6} n_i = 4$.

Running time. Since the number of c-tuples is $\mathcal{O}(n^c)$, we can solve the case of constant number of attribute configurations by $\mathcal{O}(n^c)$ calls to the single attribute case and then choosing the call that produced the smallest margin. Alg1AttBOpt has a running time of $\mathcal{O}(n)$. Overall running time is $\mathcal{O}(n^{c+1})$.

Theorem 29. AlgCartOpt *finds the optimal set of single ballot substitutions.*

Proof. We first prove that the outcome is fair. In AlgCartOpt, each c-tuple represents a possible outcome of the election in which n_i candidates with attribute vector $\vec{V}[i]$ are elected, and all c-tuples satisfy equation 7.1. As $\sum_{i=1}^{c} n_i = k$, the output top-k has $a[j, r]$ candidates from group A[j,r]. Hence, AlgCartOpt always produces fair outcome. Since we enumerate over all possible c-tuples $(n_1, \ldots, n_c)$ that satisfy fairness, AlgCartOpt produces optimal result. □

7.4.2 The 3 Attribute Case

In the 3 attribute case, each candidate has 3 attributes $A[1, j_1]$, $A[2, j_2]$ and $A[3, j_3]$, where $j_i \in [1..\ell_i]$, for $i \in \{1, 2, 3\}$. The outcome needs to have exactly $a[i, j]$ candidates with attribute $A[i, j]$, for $i \in \{1, 2, 3\}$ and $j \in [1..\ell_i]$.

Theorem 30. *Deciding the feasibility of a general instance of the 3 attribute case (and thus any $d \geq 3$ attributes as well) is NP-Complete.*

Proof. Given a solution that specifies the ballot substitutions in an instance of the 3 attribute case it is easy to check whether the solution satisfies the fairness conditions. To prove the hardness we reduce the 3-Dimensional Matching problem (3DM) to our problem. Our Technical Report contains further details. □

7.4.3 Hardness of the 2 Attribute Case

In the 2 attribute case, each candidate has 2 attributes $A[1, j_1]$, $A[2, j_2]$, where $j_1 \in [1..\ell_1]$ and $j_2 \in [1..\ell_2]$. A fair outcome needs to have exactly $a[i, j]$ candidates with attribute $A[i, j]$, for $i \in \{1, 2\}$ and $j \in [1..\ell_i]$. The problem is to find the minimum number of ballot substitutions needed to guarantee a fair outcome.

Theorem 31. *Deciding whether a given number of ballot substitutions guarantees a fair outcome of a general instance of the 2 attribute case is weakly NP-Complete.*

Proof. Given a solution that specifies the ballot substitutions in an instance of the 2 attribute case it is easy to check whether the solution satisfies the fairness conditions. To prove the hardness we reduce the weakly NP-Hard Partition problem to our problem. The proof is rather involved and we refer to our Technical Report for details. □

7.4.4 Approximating the Margin in the 2 Attribute Case

We show a 2 approximation algorithm Alg2AttApx (Algorithm 15) for computing the margin in the 2 attribute case. Let OPT_C be the optimal number of ballot substitutions required to guarantee a fair outcome. Let OPT_{A+R} be the optimal number of ballot additions and removals required to guarantee a fair outcome. The approximation algorithm first computes OPT_{A+R} ballot additions and removals that

yield a fair outcome. Then, this solution is converted to a solution with at most OPT_{A+R} ballot substitutions that has a fair outcome. Note that $OPT_{A+R} \leq 2OPT_C$, since any solution with $x \geq 0$ ballot substitutions can be converted to a solution with x ballot additions and x ballot removals. Thus the solution with OPT_{A+R} ballot substitutions is a 2 approximation.

Computing the optimal ballot additions and removals We compute OPT_{A+R} ballot additions and removals that yield a fair outcome. The subroutine FindBallotSub2Att is shown in Algorithm 14. This is done by enumerating all possible thresholds and for each threshold t we compute the optimal number of ballot additions and removals that yield a fair outcome by casting the problem as a min-cost b matching problem. We also show that the number of possible thresholds is $3n - 1$ (where n is the number of candidates).

The b-matching problem is defined on a bipartite graph $G(X, Y, E)$, where the nodes in X correspond to the possible values of the first attribute, the nodes in Y correspond to the possible values of the second attribute, and the edges correspond to the candidates. Specifically, for $i \in [1..\ell_1]$, node $x_i \in X$ corresponds to attribute value $A[1, i]$, for $j \in [1..\ell_2]$, node $y_j \in Y$ corresponds to attribute value $A[2, j]$, and a candidate c with attributes $A[1, i]$, $A[2, j]$ corresponds to an edge $e_c = (x_i, y_j)$. Note that we may have parallel edges in case there are more than one candidate with the same attributes. Next, we define the weight of each edge. The weight of edge e_c, denoted $w(e_c)$ depends on the number of votes of the candidate c. Suppose that $c = L_C(i)$ and thus this candidate has $L_V(i)$ votes. If $L_V(i) < t$ then $w(e_c) = t - L_v(i)$. Otherwise, that is $L_V(i) \geq t$, then $w(e_c) = (t - 1) - L_V(i) < 0$.

Define a b-matching in the graph G as a collection of edges such that exactly $a[1, i]$ edges are adjacent to node $x_i \in X$, for $i \in [1..\ell_1]$, and exactly $a[2, j]$ edges are adjacent to node $y_j \in Y$, for $j \in [1..\ell_2]$. Note that total number of edges in the

b-matching is $\sum_{j=1}^{\ell_1} a[1,j] = \sum_{j=1}^{\ell_2} a[2,j] = k$. Consider a b-matching $M \subseteq E$ in the graph G. Clearly, this matching corresponds to a subset of k candidates that satisfy the fairness conditions. Let $w(M) = \sum_{e \in M} w(e)$ denote the weight of the matching M.

Using the running example, for the attributes Gender and Marital Status $X = \{M, F\}$ and $Y = \{ma, si, di\}$. The candidates correspond to edges: C1 to $e_{c1} = (M, si)$, C2 to $e_{c2} = (M, si)$, C3 to $e_{c3} = (M, ma)$, and so on. Notice that edges e_{c1} and e_{c2} are parallel as both connecting node M to si. Consider a threshold $t = 2$, weight of edge e_{C1} is, $w(e_{C1}) = t - 1 - L_V(1) = 2 - 1 - 4 = -3$ because in this case $L_V(1) \geq t$. On the other hand, weight of edge e_{C5} is, $w(e_{C5}) = t - L_V(5) = 2 - 1 = 1$ since $L_V(5) < t$. The weights of the 6 edges corresponding to candidates C1, C2, C3, C4, C5, and C6 are $\{-3, -2, -1, -1, 1, 2\}$. To satisfy the fairness constraint that requires 2 male and 2 female to be in the top-4, the b-matching has 2 edges adjacent to each of the nodes M and F. Similarly, To satisfy the fairness constraint that requires 2 married, 1 single, and 1 divorced to be in the top-4, the b-matching has 2 edges adjacent to node ma and 1 edge adjacent to each of the nodes si and di. The total number of edges in b-matching is $= 2 + 2 = 2 + 1 + 1 = 4 = k$. The result of b-matching is $M^* = \{e_{C1}, e_{C3}, e_{C5}, e_{C6}\}$ and $w(M^*) = -3 - 1 + 1 + 2 = -1$. Here, $R = 3 + 2 + 1 + 1 = 7$, and $AplusR = -1 + 7 = 6$ (see below).

Theorem 32. *Let $M^* \subseteq E$ be a minimum cost matching. The number of ballot additions and removals needed to guarantee the election of the candidates that correspond to the edges of M^* with threshold t is minimum among all fair outcomes that can be obtained with threshold t.*

Proof. Let $R = \sum_{e \in E} \max\{-w(e), 0\}$ Consider the sum $w(M^*) + R$. For each edge $e_c \in M^*$ that corresponds to a candidate with less than t votes, the weight $w(e_c)$ is exactly the number of vote additions required to bring candidate c to the threshold t. Since this weight is non-negative the respective term of e_c in R is 0. For each edge $e_c \in M^*$ that corresponds to a candidate with at least t votes, its weight is negative and thus its contributions to $w(M^*)$ and R cancel each other. Each edge $e_c \in E \setminus M^*$ that corresponds to a candidate with at least t votes contributes just to R and this contribution is exactly the number of vote removals required to bring candidate c

below the threshold t. Each edge $e_c \in E \setminus M^*$ that corresponds to a candidate with less than t votes does not contribute anything to the sum. It follows that $w(M^*) + R$ is the number of ballot additions and removals needed to guarantee the election of the candidates that correspond to the edges of M^* with threshold t. Since R is independent of any specific matching the matching M^* minimizes $w(M) + R$ over all feasible matching $M \subseteq E$. The theorem follows. □

To compute O_{A+R} we need to iterate the min cost matching over all possible threshold values. We show that it is enough to consider no more than $2n$ threshold values. It is easy to see that we just need to consider threshold values in the interval $[L_V(1), L_V(n)]$. For $i \in [1..n-1]$ consider the open sub-interval $(L_V(i), L_V(i+1))$. Note that the set of candidates below this threshold and the set of candidates above this threshold are identical for all thresholds in this sub-interval. We claim that it is enough to just consider the two extreme threshold values in this sub-interval, namely, $L_V(i) + 1$ and $L_V(i+1) - 1$. Consider any threshold $t \in [L_V(i) + 2..L_V(i+1) - 2]$ and the subset of candidates that yield a fair outcome with the minimum number of ballot additions and removals with threshold t. If this subset of candidate has more candidates that are below the threshold, then the number of of ballot additions and removals required to elect this subset of candidates with threshold $L_V(i) + 1$ is lower. Otherwise, that is, at least half the candidates in this subset are not above the threshold, then the number of of ballot additions and removals required to elect this subset of candidates with threshold $L_V(i+1) - 1$ is not higher. It follows that the only threshold values that need to be checked are $L_V(i), L_V(i) + 1, L_V(i+1) - 1$, for $i \in [1..n-1]$, and $L_V(n)$.

Approximating the number of single ballot substitutions Given the O_{A+R} ballot additions and removals that yield a fair outcome with the threshold t, we generate a set of no more than $2O_C$ ballot substitutions that yield a fair outcome. Suppose that the optimal set of ballot additions and removals is composed from a additions and r removals, where $a + r = O_{A+R}$. We distinguish two cases.

Case 1: $a \leq r$. In this case we create a ballot substitutions by matching a ballot addition with a ballot removal. We are left with $r-a$ ballot removals that we convert to ballot substitutions by adding $r-a$ ballots all of them with votes to any of the already elected candidates.

Case 2: $a > r$. In this case we create r ballot substitutions by matching a ballot removal with a ballot addition. We are left with $a-r$ ballot additions. We match these addition with ballot removals that subtract votes from some (or all) the unelected candidates. Suppose that even after reducing the number of votes of all the unelected candidates to 0 we still have some unmatched ballot additions. In this case we subtract votes from some (or all) the elected candidates reducing their number of votes to the threshold t. Suppose that this is still not enough to match all the ballot additions. In this case we lower the threshold down from t. Note that as long as the threshold is not lowered to 0 the outcome remains the same (since all the unelected candidates have now 0 votes). As we lower the threshold the number of ballot that needs to be added is reduced and we can also reduce further the number of votes of the elected candidates. We claim that if the number of ballots is at least k then this process has to stop when all the ballot additions are matched at some threshold $t' > 0$. Suppose that this is not the case then at threshold 1 we still have unmatched ballot additions. However, since in this case all the elected candidates have one vote and the there are still unmatched additions then the number of ballot substitutions and thus the number of ballots is less than k. Since we need to elect k candidates it is reasonable to assume that we have more than k ballots.

Running Time. The running time of Alg2AttApx is determined by the time complexity of subroutine FindBallotSub2Att and specifically by the computation of a minimum cost b-matching in G. The b-matching problem can be solved via a min cost flow algorithm on a graph with $\ell = \ell_1 + \ell_2$ nodes and n edges. It follows that the min cost flow problem can be solved in $\mathcal{O}(n\ell \log m)$ time [6]. The subroutine

FindBallotSub2Att is called $\mathcal{O}(n)$ times from Alg2AttApx. Thus, the running time of Alg2AttApx is $\mathcal{O}(n^2\ell \log m)$.

7.5 Experimental Evaluations

We evaluate both the quality and scalability of the proposed algorithms. The quality studies focus on finding the margin values and comparing them to the implemented (optimal) baselines for the problems MFBinaryS, MFMultiS, MFMulti2, and MFMulti3+. Thus, providing computational evidence to the correctness of our exact algorithms and evaluating the computational approximation ratios of our approximation algorithms versus the approximation ratios that were proven analytically. The scalability measures the running time of the implemented algorithms by varying the number of candidates (N), number of voters (m), number of top items (k), and the complexity of the fairness constraints (single binary, vs single multi-valued, vs two attributes, or more than three attributes).

7.5.1 Experiment Design

All the algorithms are implemented in Python 3.8 on a machine with Windows 11, core i7 with 16gb memory. All numbers are presented as an average of 10 runs. Code and data could be found in the github.

Datasets Description Algorithms are evaluated using multiple real world and a synthetic datasets. The datasets descriptions are presented in Table 7.5.

Synthetic dataset. We generate large scale synthetic data using normal distribution for each binary protected attribute, as voting outcomes tend to follow such distributions [105]. The process runs as follows -a loop is repeated m times to generate an id between 1...n (top candidate choice of a voter) that follows normal distribution with certain mean ($mean$) and standard deviation (sd). Additionally, there is a parameter p that dictates the probability of candidates of the first group

Table 7.5 Real World Datasets

Dataset	# candidates(n)	# voters(m)	protected attributes (ℓ)
New South Wales (NSW) Senate Elections	105	4695326	single multi-valued $\ell = 36$ on the political parties and the election history
Bronx Justice of the Supreme Court Election in New York City	$107,163$	4	single binary - democrat and republican
MovieLens	2926	382323	3 attributes on movie genre, production company and original language.

to be ranked higher than that in the second group. The *mean*, *sd* and p are set to $0.5 * n$, $0.3 * n$ and 0.5, respectively. We consider the top-1 choice of each voter to generate the voting outcome.

Implemented Algorithms We implement the following baseline algorithms. The first two baselines are heuristics, whereas, the last one gives exact solution of the problem. These algorithms are compared with our proposed solutions Alg1AttBOpt, Alg1AttMOpt, Alg2AttApx, AlgCartOpt.

- LEXIMIN [66] + `FindBallotSub`. This existing work is not designed to solve the margin finding problem, but it produces a *probability distribution* of a set of possible top-k candidates, where each set satisfies fairness constraints. We draw one such top-k set from the output distribution based on the associated probability and consider that to be the set of selected candidates in top-k. Given this top-k, we run the `FindBallotSub` to compute the margin.

- FAIR-TOPK-SET [129]+`FindBallotSub`. This related work also does not solve the margin finding problem. Rather it proposes a greedy algorithm to satisfy fairness constraint over a single protected attribute on the top-k set, similar to one of our prior work [140]. When multiple protected attributes are considered, it converts it to a single multi-valued protected attribute considering the Cartesian Product over the attribute values and taking the joint distribution assuming the attributes to be independent. As before, once these steps are performed, we run the `FindBallotSub` to compute the margin.
- **Integer Linear Programming.** We implement an exact algorithm for MFBINARYS, MFMULTIS, MFMULTI2, and MFMULTI3+ problems using ILP. We refer to these variants as `ILPBinaryS`, `ILPMultiS`, `ILPMulti2`, `ILPMulti3+`, respectively.

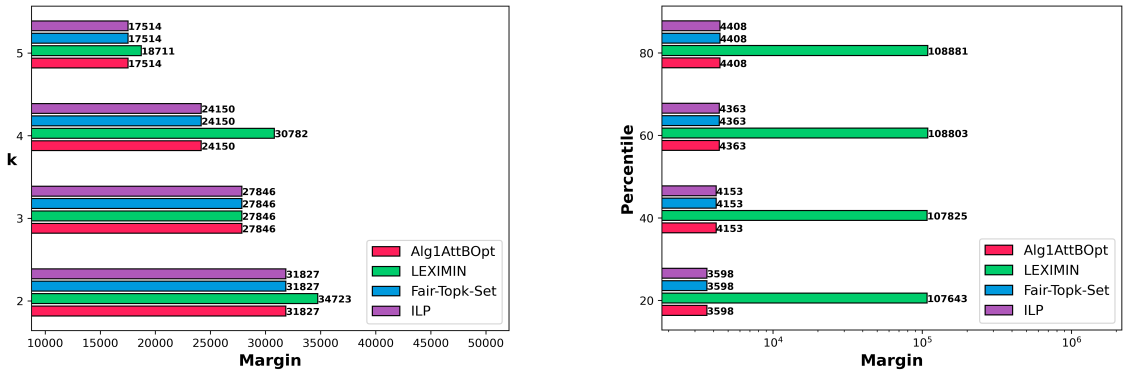
7.5.2 Summary of Results

Our first and foremost observation is, consistent with our theoretical analysis ALG1ATTBOPT, ALG1ATTMOPT, ALGCARTOPT produce exact solutions of the underlying problems, i.e., they satisfy the fairness constraints, while minimizing the margin values, whereas, ALG2ATTAPX demonstrates better approximation factors compared to the theoretical bound 2. Our second observation is, the implemented state-of-the-art solutions LEXIMIN [66] and FAIR-TOPK-SET [129], despite adapting them non-trivially to our problem, fail to optimize margin values and do not turn out to be effective to be used. Our final observation is, consistent with our theoretical analysis, ALG1ATTBOPT, ALG1ATTMOPT, and ALG2ATTAPX are highly scalable, and run well on outcome consisting of a very large number of candidates, ballots, or large k. We find that the value of margin depends on both k and the vote gaps between candidates: with increasing k, margin generally decreases with larger k. On the other hand, we also notice that the margin increases with larger gap between the candidates.

7.5.3 Quality Experiments Results

Results for MFBinaryS The Figure 7.1(a) includes the data from the election for 2021 Bronx Justice of the Supreme Court. There are 6 candidates(5 Democrats and 1 Republican), in which the candidate from Republican receives the least votes. We set the Republican must be included in the top-k (otherwise, the margin would be zero). We can observe that the margin decreases with increasing k.

We construct a synthetic attribute for the candidates in 2019 NSW Senate Election dataset in Figure 7.1(b): isAtxPercentile, where x is the specific percentile we want to evaluate (only one member will be assigned $True$ and all others are $False$). This figure illustrates the relation between the margin and vote gaps across candidates, we calculate the required margin for promoting one candidate at the specific percentile position into the top-5.

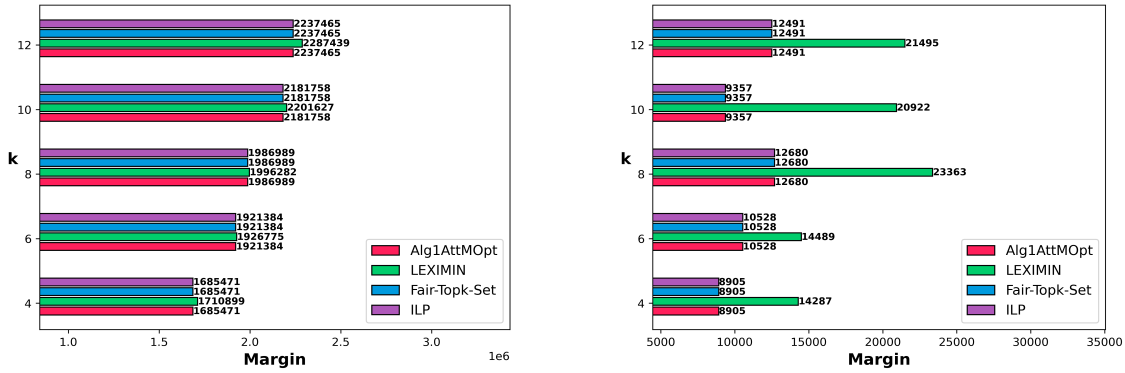


(a) 2021 Bronx Justice Election

(b) 2019 NSW Senate Election

Figure 7.1 Results for MFBinaryS.

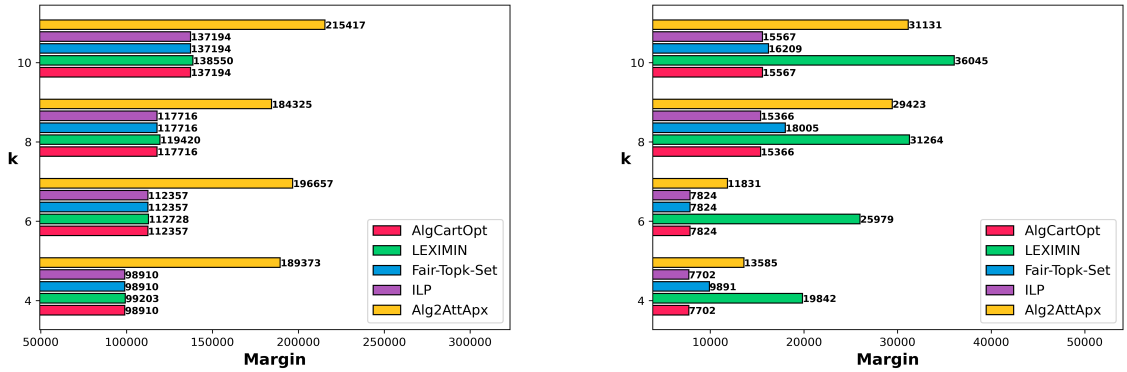
Results for MFMultiS We present the results for MFMultiS in Figure 7.2. We consider the party of candidates in 2019 NSW Senate Election and the movie genre in MovieLens as the protected attributes. In 2019 NSW Senate Election, candidates are from 36 parties and there 18 genres in MovieLens dataset.



(a) 2019 NSW Senate Election (b) MovieLens

Figure 7.2 Results for MFMultiS.

Results for MFMulti2 Figure 7.3 shows the results for MFMulti2. We consider whether the candidate has been elected before as the additional protected attribute in NSW Senate Election dataset, and only four candidates have been elected before. On the other hand, the binary attribute: whether produced by American companies of each movie is the additional protected attribute in MovieLens dataset, and 540 movies are produced by American companies. We can observe that the margins from Alg2AttApxis bounded by 2 times of ILP's result and AlgCartOptproduces the exact results which match our expectations.



(a) 2019 NSW Senate Election (b) MovieLens

Figure 7.3 Results for MFMulti2.

Results for MFMULTI3+ Similar to 7.5.3, we consider the binary attribute: whether the original language of movie is English as the third protected attribute in MoiveLens and there are 2112 movie's original language is English.

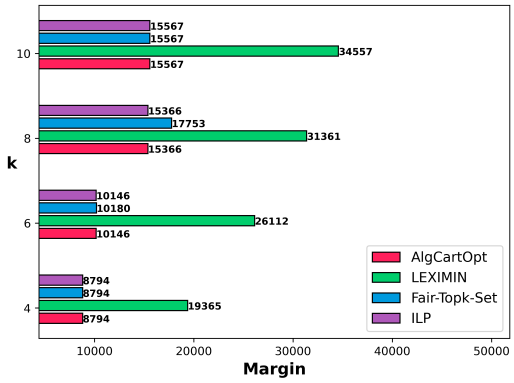
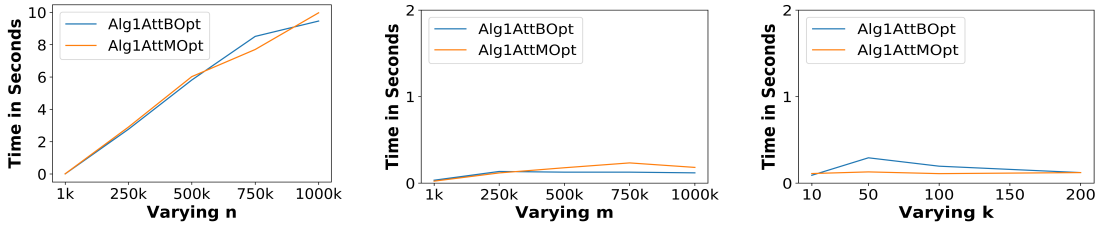


Figure 7.4 Results for MFMULTI3+.

7.5.4 Scalability Results



(a) ALG1ATTBOPT vs ALG1ATTMOPT vary n, $k = 50, m = 10000k$

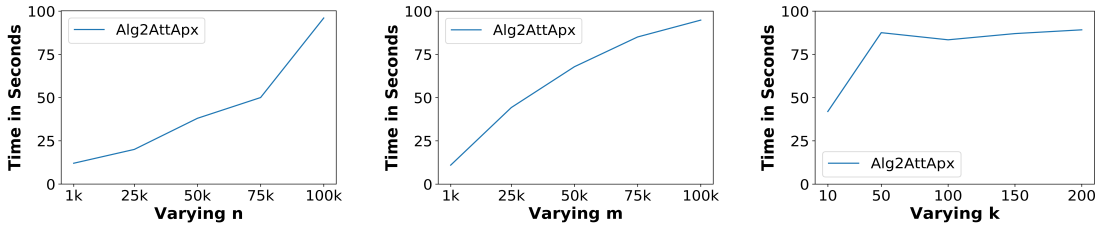
(b) ALG1ATTBOPT vs ALG1ATTMOPT vary m, $k = 50, n = 10k$

(c) ALG1ATTBOPT vs ALG1ATTMOPT vary k, $n = 10k, m = 10000k$

Figure 7.5 Running time for MFBINARYS, MFMULTIS.

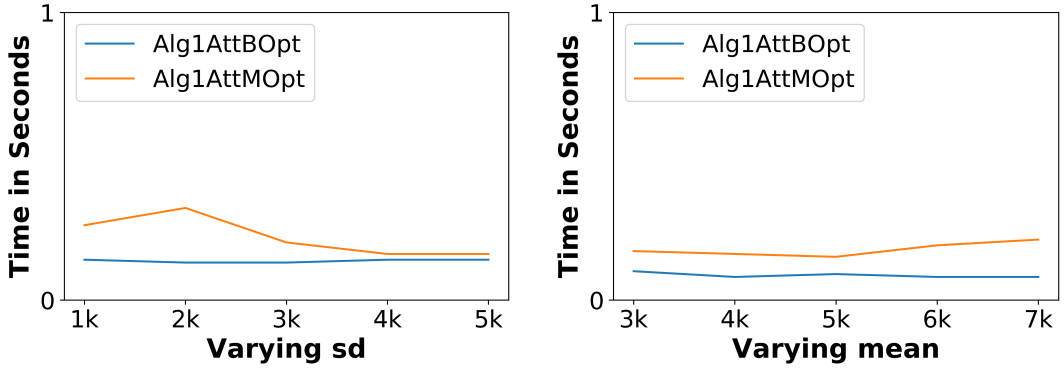
For these experiments, we use the synthetically generated normally distributed data to validate the effect of the parameters n, m, k on the running time of the proposed algorithms, considering one, two, or three or more protected attributes.

Results for MFBINARYS, MFMULTIS In these experiments, we set the default parameters as follows: $n = 10k, m = 10000k, k = 50, \ell = 2$ for ALG1ATTBOPT



(a) ALG2ATTAPX vary n, $m = 100k, k = 50$ (b) ALG2ATTAPX vary m, $n = 10k, k = 50$ (c) ALG2ATTAPX vary k, $n = 10k$, $m = 100k$

Figure 7.6 Running time for MFMULTI2.



(a) Vary sd, $m = 1000k, n = 10k$: ALG1ATTBOPT vs ALG1ATTMOPT (b) Vary $mean$, $m = 1000k, n = 10k$: ALG1ATTBOPT vs ALG1ATTMOPT

Figure 7.7 Varying distribution MFBINARYS, MFMULTIS.

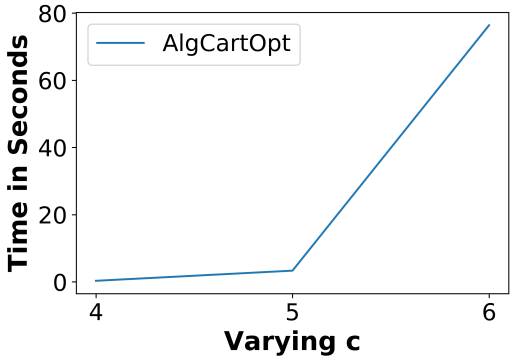


Figure 7.8 Running time for MFMULTI3+

and $\ell = 3$ for ALG1ATTMOPT considering a single attribute. Figure 7.5 shows that our proposed algorithms ALG1ATTBOPT and ALG1ATTMOPT are scalable up to millions of candidates and voters. In Figure 7.5(a), the running time increases log-linearly w.r.t number of candidates n, whereas the running time does not change significantly while varying number of voters m and the size of the result set k (Figure 7.5(b), 7.5(c)).

Figure 7.7(b) and 7.7(a) show running times of ALG1ATTBOPT and ALG1ATTMOPT varying standard deviation (sd) and $mean$. We observe non-significant change in running time due to sd and $mean$.

Results for MFMULTI2 The default parameters for the experiments of MFMULTI2 are as follows: $n = 10k$, $m = 1000k$, $k = 50$ considering two attributes. The running time of ALG2ATTAPX w.r.t n,m and k are shown in Figure 7.6. The running time of ALG2ATTAPX is sub-quadratic w.r.t number of candidates n as shown in Figure 7.6(a), while it increases linearly w.r.t number of voters m (shown in Figure 7.6(b)) and does not change significantly with the size of the result set k (Figure 7.6(c)).

Results for MFMULTI3+ In these experiments, we set the default parameters as follows: $n = 100, m = 1000$ considering three or more attributes. In Figure 7.8, we present the running time of ALGCARTOPT w.r.t c where $c = \Pi_{i=1}^{\ell} \ell_i$. Here, the running time increases exponentially with c.

7.6 Related Work

We primarily discuss three types of existing work that are related to our proposed problem.

Preference Elicitation and Aggregation. Preference of the individual users could be elicited as pairwise comparison [55], in form of a binary vector [124] known as

Approval Voting [37], in an ordinal scale [12,86], or considering Arrowian social choice, where users provide partial or complete preference order over the items [32, 42, 85, 127]. Similarly, The properties of social welfare functions for aggregating preferences have been studied by mathematicians since the 18th century [41, 46, 48]. Different preference aggregation methods are proposed, including majority voting, plurality voting [90, 104, 113], their weighted versions, as well as aggregation methods that consider positional preference [32, 42, 127], such as Kemeny rule [59, 82], Condorcet rule [52], or Borda Count [60]. Our adopted preference elicitation model allows users to provide their top-choice and aggregate these choices using plurality voting, which is natural for our problem setting.

Fairness in Preference Aggregation and Top-k. In [67, 121], authors study the fairness of preference aggregation in the context of Arrow's Impossibility Theorem. In a very recent work of ours [140], we study how to ensure proportionate fairness in aggregating preferences that are provided as ranked orders. The idea of proportionate fairness (p-fairness) is studied in the context of resource scheduling [26, 27]. Earlier, existing works study proportionate representation considering group fairness in the top-k ranked order [72, 87]. Authors in [69] study how to strike a balance between individual and group fairness in selecting top-k order. In [18], the authors study how to tune the weights of the attributes to promote fairness in the top-k ranked results. We are also aware of prior works that select top-k set [66, 129] to maximize fairness or diversity. In a recent paper [44], the authors maximize a monotone submodular function given only upper bound of fairness constraints. They study three different functions, namely, Chamberlin-Courant rule, SNTV (similar to ours) and $\alpha - CC$, where SNTV is similar to ours. Unlike our work, it does not study ballot substitution as well as does not consider exact fairness constraints.

Preference Update Models. Preference update could elicited by adding a new vote, deleting an existing vote, or substituting the original preference with another

choice. In the absence of adversaries, the last one is most realistic that we adopt in this work. Preference substitution of preference transfer has received significant interests in electoral systems, in particular, to understand the mechanism of Single Transferable Vote (STV) [64, 133]. In [144], the authors study *margin*, defined as the number of ballot manipulation to introduce a different election result for different voting systems, including approval voting, all positional scoring rules (which include Borda, plurality, and veto). In [33, 43, 126] margin is calculated in STV setting. In [25], Orlin and Bartholdi prove margin finding is NP-hard even for a single candidate selection for STV where voters provide a full preference of the candidates.

While we adapt popular preference elicitation and preference aggregation models, we are the first to study the margin finding problem under single ballot substitutions considering complex fairness constraints defined over multiple protected attributes and present principled solutions with theoretical guarantees.

7.7 Discussion

In this section, we present extensions of the proposed problems. **Fair top-k considering top-p ranked preference.** One extension is to study the preference substitution problem where the ballot contains a ranked order of top-p preference. Clearly, when multiple ranked preferences are provided by voters, the preference substitution is much more complex and has to be guided by specific mechanism. We intend to study this problem considering the preference substitution model outlined in the STV process (Single Transferable Vote) [25, 64]. In STV, the vote substitution is guided by the notion of *quota*, which is the minimum number of votes a candidate needs to be selected in top-k that depends on the value of the number of candidates (n), and the number of seats (k). STV operates in rounds, where, in each round the mechanism allows either selecting one or more candidates to be part of the top-k, or eliminating a candidate from further consideration. STV provides a mechanism to "transfer" excess votes from the winning and losing candidates to be distributed

amongst the next best choice of the voters in the ballot. An ongoing effort is to simply understand the margin finding problem under STV model considering even a single protected attribute.

Alternative preference update models. In this proposed work, we study the preference update model considering *only substitutions*. One can potentially consider other update models, such as, just adding votes, as well as deleting votes. Clearly, these alternative update models do not satisfy the vote invariance property, which makes these models contrived for certain applications. However, under the presence of an adversary, such update models may appear realistic. As an ongoing work, we are studying these alternative models under adversarial consideration and we believe our algorithmic framework can be applied for these models as well.

7.8 Conclusion

We initiate the study of *the margin finding problem for preference aggregation under single ballot substitutions*, considering one and multiple protected group attributes to promote fairness, present a suite of algorithms with provable guarantees, and conduct rigorous experimental analysis.

In Section 7.7, we discuss how our proposed work could be used as a starting point for studying fairness criteria in some generalized settings, such as, ensuring fairness where the substitution is governed by STV as well as considering other preference update models that we continue to study as ongoing works.

Algorithm 14 FINDBALLOTSUB2ATT

Inputs: L_C, L_V, A, t

Outputs: $AplusR$ = Sum of vote addition and removal

1: $X = \{x_i : x_i \in A_1\}$

2: $Y = \{y_j : y_j \in A_2\}$

3: $E = \{e_c = (x_i, y_j) : c \in L_C$ with attributes $A[1,i]$ $A[2,j].\}$

4: **for** $i = 1$ to n **do**

5: $\quad c = L_C(i)$

6: $\quad$ **if** $L_V(i) < t$ **then**

7: $\quad\quad w(e_c) = t - L_V(i)$

8: $\quad$ **else**

9: $\quad\quad w(e_c) = (t-1) - L_V(i)$

10: $\quad$ **end if**

11: **end for**

12: Construct the graph G $= (X, Y, E, w)$

13: Set the constraints on the number of adjacent edges of nodes x_i and y_j to $a[1,i]$ and $a[2,j]$ respectively

14: Find M^* a min cost b-matching in G subject to the constraints on the number of adjacent edges

15: $R = \sum_{e \in E} \max\{-w(e), 0\}$

16: $AplusR = w(M^*) + R.$

17: Return $AplusR$

Algorithm 15 ALG2ATTAPX

Inputs: L_V,L_C,A

Outputs: OPT_{A+R}

1: $OPT_{A+R} = k \cdot m$

2: $U = \{L_V(i), L_V(i)+1, L_V(i+1)-1 : i \in [1 \dots n-1]\} \cup \{L_V(n)\}$

3: **for** $t \in U$ **do**

4: $\quad AplusR =$ FINDBALLOTSUB2ATT(L_C, L_V, A, t)

5: $\quad$ **if** $OPT_{A+R} > AplusR$ **then**

6: $\quad\quad OPT_{A+R} = AplusR$

7: $\quad$ **end if**

8: **end for**

9: Return OPT_{A+R}

CHAPTER 8

CONCLUSION AND FUTURE WORK

The goal of this dissertation is in designing quantitative models and algorithms that recognize and capture human characteristics and study optimization in the HITL computational paradigm. In this chapter, we summarize the contribution in Section 8.1 and discuss some future works in Section 8.2.

8.1 Conclusion

A HITL process requires holistic treatment and optimization from multiple standpoints considering all stakeholders: a. applications, b. platforms, c. humans. In application-centric optimization, the factors of interest usually are latency (how long it takes for a set of tasks to finish), cost (the monetary or computational expenses incurred in the process), and quality of the completed tasks. Platform-centric optimization studies throughput, or revenue maximization, while human-centric optimization deals with the characteristics of the human workers, referred to as human factors, such as their skill improvement and learning, to name a few. Finally, fairness and ethical consideration are also of utmost importance in these processes.

In Chapter 2, we propose StratRec, an optimization-driven middle layer designed to recommend strategies for multiple deployment requests or recommend alternative deployment parameters if a request cannot be satisfied as formulated. Our work addresses multi-faceted modeling challenges through the generic design of modules in StratRec that could be instantiated to optimize different types of goals by accounting for worker availability. We develop computationally-efficient algorithms. We present extensive real data experiments through multiple deployments in AMT and conduct synthetic experiments to validate the qualitative and scalability aspects of StratRec.

We focus on how to enhance the workers' self-improvement in Chapter 3. To the best of our knowledge, our work is the first to examine algorithmic group formation with affinities for peer learning. We examine online group formation where members seek to increase their *learning potential* via task completion with two learning models and affinity structures. We formalize the problem of forming a set of k groups with the purpose of optimizing peer learning under different affinity structures and propose constrained optimization formulations. We show the hardness of our problems and develop 4 scalable algorithms with constant approximation factors. Our experiments with real workers demonstrate that considering affinity structures drastically improves learning potential, and our synthetic data experiments corroborate the qualitative and scalability aspects of our algorithms.

We investigate the study of peer learning processes in a dynamic manner in Chapter 4. The problem is motivated by practical considerations of groups in online social networks or offline classroom learning. In each round, the participants are split into groups and learn from interactions within their group. The objective is to find a *sequence* of groupings that will maximize the total knowledge at the end of the process. We introduce a model and an associated algorithmic framework DyGroups for this problem. Using the insights we gain from our theoretical study, we design experiments with human subjects that provide evidence corroborating our hypothesis that the choice of groupings can indeed significantly impact the total amount of learning.

In Chapter 5, we introduce the p-fairness to the rank aggregation problem and propose the **RAPF** problem to incorporate a group fairness criteria (p-fairness) considering binary and multi-valued protected attributes with the classical rank aggregation problem. We first study how to produce a p-fair ranking that is closest to a single input ranking (**IPF**). **IPF** can be solved exactly using a greedy technique when the protected attribute is binary. When the protected attribute is

multi-valued such an approach fails. We then present two solutions for multi-valued **IPF**, ExactMultiValuedIPF is optimal and ApproxMultiValuedIPF admits 2 approximation factor. Next, we design two computational frameworks to solve **RAPF**: RandAlgRAPF and AlgRAPF that exhibit 3 and 4 approximation factors when designed using ExactMultiValuedIPF and ApproxMultiValuedIPF, respectively.

In Chapter 6, we initiate the study of *the margin finding problem for preference aggregation under single ballot substitutions*, considering one and multiple protected group attributes to promote fairness, where given an aggregated preference of m voters over n candidates, a single substitution amounts to removing a vote from candidate i and assigning it to candidate j, and the goal is to minimize the number of single ballot substitutions to satisfy complex fairness constraints in the top-k results. We systematically study several variants of this problem, considering how fairness constraints are specified over a single binary or multi-valued protected attribute or over multiple protected attributes. We study these variants theoretically and present a suite of algorithms with provable guarantees. We conduct rigorous large-scale experiments involving multiple real-world datasets by appropriately adapting multiple state-of-the-art solutions to demonstrate the effectiveness and scalability of our proposed methods.

8.2 Future Work

In this section, we outline a few ongoing and future works that are related.

8.2.1 Recommending Deployment Strategies for Considering Worker Availability as a Probability Density Function

Our proposed work StratRec, is an optimization-driven middle layer that is designed to recommend strategies to multiple deployment requests or recommend alternative

deployment parameters if a request cannot be satisfied as formulated. In this future work, we will study the design of StratRec considering worker availability as a discrete probability distribution function. We believe that under such an assumption, the proposed solution will be different from the existing one, as the problem formulation itself will require a non-trivial extension.

8.2.2 Fair Task Assignment in HITL

Our current works have studied how to form groups to maximize skill improvements of the workers, considering both static and dynamic group formation characteristics [61, 141]. The theoretical and experimental results in [141] however, indicate that, in order to maximize cumulative skill improvement (i.e., the sum of learning potential) over all the members, the current optimization function tends to sacrifice the skill improvement of the least skilled workers, making the process less equitable and fair. Therefore, in future work, we intend to study a bi-criteria optimization problem considering both fairness and other utilities related to learning, such as skill improvement. Such a problem will perhaps strike a better balance in improving the aggregated learning potential while ensuring the equity of skill improvements across all workers.

REFERENCES

[1] Akanksha Agrawal, Grzegorz Guspiel, Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Connecting the dots (with minimum crossings). In Gill Barequet and Yusu Wang, editors, *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA*, volume 129 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[2] Rakesh Agrawal, Behzad Golshan, and Evangelos E. Papalexakis. Toward data-driven design of educational courses: A feasibility study. In Tiffany Barnes, Min Chi, and Mingyu Feng, editors, *Proceedings of the 9th International Conference on Educational Data Mining, EDM 2016, Raleigh, North Carolina, USA, June 29 - July 2, 2016*, page 6. International Educational Data Mining Society (IEDMS), 2016.

[3] Rakesh Agrawal, Behzad Golshan, and Evimaria Terzi. Grouping students in educational settings. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 1017–1026. ACM, 2014.

[4] Rakesh Agrawal, Sharad Nandanwar, and Narasimha Murty Musti. Grouping students for maximizing learning from peers. In Xiangen Hu, Tiffany Barnes, Arnon Hershkovitz, and Luc Paquette, editors, *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017, Wuhan, Hubei, China, June 25-28, 2017*. International Educational Data Mining Society (IEDMS), 2017.

[5] Rakesh Agrawal, Sharad Nandanwar, and Narasimha Murty Musti. Grouping students for maximizing learning from peers. In Xiangen Hu, Tiffany Barnes, Arnon Hershkovitz, and Luc Paquette, editors, *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017, Wuhan, Hubei, China, June 25-28, 2017*. International Educational Data Mining Society (IEDMS), 2017.

[6] Ravindra K. Ahuja, Andrew V. Goldberg, James B. Orlin, and Robert Endre Tarjan. Finding minimum-cost flows by double scaling. *Math. Program.*, 53:243–266, 1992.

[7] Nir Ailon. Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica*, 57(2):284–300, 2010.

[8] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

[9] BJ Allen, Deepa Chandrasekaran, and Suman Basuroy. Design crowdsourcing: The impact on new product performance of sourcing design solutions from the "crowd". *Journal of Marketing*, 82(2):106–123, 2018.

[10] Sihem Amer-Yahia and Senjuti Basu Roy. The ever evolving online labor market: Overview, challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1978–1981, 2019.

[11] Sihem Amer-Yahia and Senjuti Basu Roy. Data management to social science and back in the future of work. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2876–2877. ACM, 2021.

[12] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawla, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *Proceedings of the VLDB Endowment*, 2(1):754–765, 2009.

[13] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, and Aristides Gionis. An optimization framework for query recommendation. In Brian D. Davison, Torsten Suel, Nick Craswell, and Bing Liu, editors, *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*, pages 161–170. ACM, 2010.

[14] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. Power in unity: forming teams in large-scale community systems. In Jimmy Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An, editors, *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 599–608. ACM, 2010.

[15] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi. Online team formation in social networks. In Alain Mille, Fabien Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 839–848. ACM, 2012.

[16] Michael R Anderson, Dolan Antenucci, and Michael J Cafarella. Runtime support for human-in-the-loop feature engineering system. *IEEE Data Eng. Bull.*, 39(4):62–84, 2016.

[17] Kenneth J Arrow. A difficulty in the concept of social welfare. *Journal of political economy*, 58(4):328–346, 1950.

[18] Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019,*

Amsterdam, The Netherlands, June 30 - July 5, 2019, pages 1259–1276. ACM, 2019.

[19] Pieter Cornelis Baayen and Z Hedrlin. *On the existence of well distributed sequences in compact spaces*. Stichting Mathematisch Centrum. Zuivere Wiskunde, 1964.

[20] Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 44–54. ACM, 2006.

[21] Sanaz Bahargam, Dóra Erdos, Azer Bestavros, and Evimaria Terzi. Team formation for scheduling educational material in massive online classes. *arXiv preprint arXiv:1703.08762*, 2017.

[22] Sanaz Bahargam, Theodoros Lappas, and Evimaria Terzi. The guided team-partitioning problem: Definition, complexity, and algorithm. *arXiv preprint arXiv:1905.03037*, 2019.

[23] John A Bargh and Yaacov Schul. On the cognitive benefits of teaching. *Journal of Educational Psychology*, 72(5):593, 1980.

[24] John Bartholdi, Craig A Tovey, and Michael A Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[25] John J Bartholdi and James B Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

[26] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[27] Sanjoy K Baruah, Johannes E Gehrk, C Greg Plaxton, Ion Stoica, Hussein Abdel-Wahab, and Kevin Jeffay. Fair on-line scheduling of a dynamic set of tasks on a single resource. *Information Processing Letters*, 64(1):43–51, 1997.

[28] Senjuti Basu Roy, Ioanna Lykourentzou, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Task assignment optimization in knowledge-intensive crowdsourcing. *The VLDB Journal*, 24(4):467–491, 2015.

[29] Adil Baykasoglu, Turkay Dereli, and Sena Das. Project team selection using fuzzy optimization approach. *Cybernetics and Systems: An International Journal*, 38(2):155–185, 2007.

[30] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, May 23-25, 1990*, pages 322–331. ACM Press, 1990.

[31] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.

[32] Julian H Blau. The existence of social welfare functions. *Econometrica: Journal of the Econometric Society*, pages 302–313, 1957.

[33] Michelle Blom, Peter J Stuckey, and Vanessa J Teague. Towards computing victory margins in stv elections. *arXiv preprint arXiv:1703.03511*, 2017.

[34] JC de Borda. Mémoire sur les élections au scrutin. *Histoire de l'Academie Royale des Sciences pour 1781 (Paris, 1784)*, 1784.

[35] Ria Mae Borromeo, Thomas Laurent, Motomichi Toyama, Maha Alsayasneh, Sihem Amer-Yahia, and Vincent Leroy. Deployment strategies for crowdsourcing text creation. *Information Systems*, 71:103–110, 2017.

[36] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In Dimitrios Georgakopoulos and Alexander Buchmann, editors, *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430. IEEE Computer Society, 2001.

[37] Steven J Brams and Peter C Fishburn. Approval voting. *American Political Science Review*, 72(3):831–847, 1978.

[38] Bryan Brancotte, Bo Yang, Guillaume Blin, Sarah Cohen-Boulakia, Alain Denise, and Sylvie Hamel. Rank aggregation with ties: Experiments and analysis. *Proceedings of the VLDB Endowment (PVLDB)*, 8(11):1202–1213, 2015.

[39] Gerth Stølting Brodal, Loukas Georgiadis, Kristoffer Arnsfelt Hansen, and Irit Katriel. Dynamic matchings in convex bipartite graphs. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 406–417. Springer, 2007.

[40] Hongyun Cai, Vincent W Zheng, Fanwei Zhu, Kevin Chen-Chuan Chang, and Zi Huang. From community detection to community profiling. *arXiv preprint arXiv:1701.04528*, 2017.

[41] Donald E Campbell and Jerry S Kelly. Information and preference aggregation. *Social Choice and Welfare*, 17(1):3–24, 2000.

[42] Donald E Campbell and Jerry S Kelly. Impossibility theorems in the arrovian framework. *Handbook of Social Choice and Welfare*, 1:35–94, 2002.

[43] David Cary. Estimating the margin of victory for instant-runoff voting. In Hovav Shacham and Vanessa Teague, editors, *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11, San Francisco, CA, USA, August 8-9, 2011*. USENIX Association, 2011.

[44] L Elisa Celis, Lingxiao Huang, and Nisheeth K Vishnoi. Multiwinner voting with fairness constraints. *arXiv preprint arXiv:1710.10057*, 2017.

[45] L Elisa Celis, Damian Straszak, and Nisheeth K Vishnoi. Ranking with fairness constraints. *arXiv preprint arXiv:1704.06840*, 2017.

[46] Christopher P Chambers and Takashi Hayashi. Preference aggregation under uncertainty: Savage vs. pareto. *Games and Economic Behavior*, 54(2):430–440, 2006.

[47] Danny Z Chen, Xiaomin Liu, and Haitao Wang. Computing maximum non-crossing matching in convex bipartite graphs. *Discrete Applied Mathematics*, 187:50–60, 2015.

[48] Yong-Gon Cho and Keun-Tae Cho. A loss function approach to group preference aggregation in the AHP. *Comput. Oper. Res.*, 35(3):884–892, 2008.

[49] Jan Chomicki, Paolo Ciaccia, and Niccolò Meneghetti. Skyline queries, front and back. *SIGMOD Rec.*, 42(3):6–18, 2013.

[50] Alexandra Chouldechova and Aaron Roth. The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810*, 2018.

[51] Elizabeth G Cohen. Restructuring the classroom: Conditions for productive small groups. *Review of educational research*, 1994.

[52] Marquis de Condorcet. Essay on the application of analysis to the probability of majority decisions. *Paris: Imprimerie Royale*, 1785.

[53] Vincent Conitzer, Andrew J. Davenport, and Jayant Kalagnanam. Improved bounds for computing kemeny rankings. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 620–626. AAAI Press, 2006.

[54] Thanasis Daradoumis, Montse Guitert, Ferran Giménez, Joan Manuel Marquès, and T. Lloret. Supporting the composition of effective virtual groups for collaborative learning. In *International Conference on Computers in Education, ICCE 2002, Auckland, New Zealand, December 3-6, 2002, Volume 1*, pages 332–336. IEEE Computer Society, 2002.

[55] Luca de Alfaro and B. Thomas Adler. Content-driven reputation for collaborative systems. In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *Lecture Notes in Computer Science*, pages 3–13. Springer, 2013.

[56] Persi Diaconis and Ronald L Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):262–268, 1977.

[57] Djellel Difallah, Elena Filatova, and Panos Ipeirotis. Demographics and dynamics of mechanical turk workers. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 135–143, 2018.

[58] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 214–226. ACM, 2012.

[59] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 613–622. ACM, 2001.

[60] Peter Emerson. The original borda count and partial voting. *Soc. Choice Welf.*, 40(2):353–358, 2013.

[61] Mohammadreza Esfandiari, Dong Wei, Sihem Amer-Yahia, and Senjuti Basu Roy. Optimizing peer learning in online groups with affinities. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 1216–1226. ACM, 2019.

[62] Sarah Evans, Katie Davis, Abigail Evans, Julie Ann Campbell, David P. Randall, Kodlee Yin, and Cecilia R. Aragon. More than peer production: Fanfiction communities as sites of distributed mentoring. In Charlotte P. Lee, Steven E. Poltrock, Louise Barkhuus, Marcos Borges, and Wendy A. Kellogg, editors, *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW 2017, Portland, OR, USA, February 25 - March 1, 2017*, pages 259–272. ACM, 2017.

[63] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal of Discrete Mathematics*, 20(3):628–648, 2006.

[64] David M Farrell, Malcolm Mackerras, and Ian McAllister. Designing electoral institutions: Stv systems and their consequences. *Political studies*, 44(1):24–43, 1996.

[65] Peter C Fishburn. Arrow's impossibility theorem: Concise proof and infinite voters. *Journal of Economic Theory*, 2(1):103–106, 1970.

[66] Bailey Flanigan, Paul Gölz, Anupam Gupta, Brett Hennig, and Ariel D Procaccia. Fair algorithms for selecting citizens' assemblies. *Nature*, 596(7873):548–552, 2021.

[67] Marc Fleurbaey, Kotaro Suzumura, and Koichi Tadenuma. The informational basis of the theory of fair allocation. *Social Choice and Welfare*, 24(2):311–341, 2005.

[68] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 61–72. ACM, 2011.

[69] David García-Soriano and Francesco Bonchi. Maxmin-fair ranking: Individual fairness under group-fairness constraints. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 436–446. ACM, 2021.

[70] Michael R Garey and David S Johnson. *Computers and intractability*. wh freeman New York, 2002.

[71] Susan Gauch and John B. Smith. Search improvement via automatic query reformulation. *ACM Trans. Inf. Syst.*, 9(3):249–280, 1991.

[72] Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2221–2231. ACM, 2019.

[73] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. SIMPATH: an efficient algorithm for influence maximization under the linear threshold model. In Diane J. Cook, Jian Pei, Wei Wang, Osmar R. Zaïane, and Xindong Wu, editors, *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 211–220. IEEE Computer Society, 2011.

[74] Adrien Guille, Hakim Hacid, Cécile Favre, and Djamel A. Zighed. Information diffusion in online social networks: a survey. *SIGMOD Rec.*, 42(2):17–28, 2013.

[75] Corinna Hertweck, Christoph Heitz, and Michele Loi. On the moral justification of statistical parity. In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 747–757. ACM, 2021.

[76] R Hertz-Lazarowitz, S Kagan, Shlomo Sharan, R Slavin, and Clark Webb. *Learning to cooperate, cooperating to learn*. Springer Science & Business Media, 2013.

[77] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[78] Oscar H Ibarra and Chul E Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468, 1975.

[79] C Kirabo Jackson and Elias Bruegmann. Teaching students and teaching each other: The importance of peer learning for teachers. *American Economic Journal: Applied Economics*, 1(4):85–108, 2009.

[80] Wen Jin, Martin Ester, Zengjian Hu, and Jiawei Han. The multi-relational skyline operator. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 1276–1280. IEEE Computer Society, 2007.

[81] Ouiame Ait El Kadi. Exploring crowdsourcing deployment strategies through recommendation and iterative refinement. *MS Research Report*, 2018.

[82] John G Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[83] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In Lise Getoor, Ted E. Senator, Pedro M. Domingos, and Christos Faloutsos, editors, *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 137–146. ACM, 2003.

[84] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[85] Craig W. Kirkwood and Rakesh K. Sarin. Ranking with partial information: A method and an application. *Oper. Res.*, 33(1):38–48, 1985.

[86] Yehuda Koren and Joe Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In Bamshad Mobasher, Robin D. Burke, Dietmar Jannach, and Gediminas Adomavicius, editors, *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011*, pages 117–124. ACM, 2011.

[87] Caitlin Kuhlman and Elke A. Rundensteiner. Rank aggregation algorithms for fair consensus. *Proceedings of the VLDB Endowment*, 13(11):2706–2719, 2020.

[88] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 467–476. ACM, 2009.

[89] Isabella Lari, Federica Ricca, Justo Puerto, and Andrea Scozzari. Partitioning a graph into connected components with fixed centers and optimizing cost-based objective functions or equipartition criteria. *Networks*, 67(1):69–81, 2016.

[90] Jean-François Laslier. And the loser is... plurality voting. In *Electoral systems*, pages 327–351. Springer, Berlin, Heidelberg, 2012.

[91] Eugene L Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 1979.

[92] Pedro G Lind, Luciano R Da Silva, José S Andrade Jr, and Hans J Herrmann. Spreading gossip in social networks. *Physical Review E*, 76(3):036117, 2007.

[93] Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: a crowdsourcing data analytics system. *arXiv preprint arXiv:1207.0143*, 2012.

[94] Anirban Majumder, Samik Datta, and K. V. M. Naidu. Capacitated team formation problem on social networks. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 1005–1013. ACM, 2012.

[95] Colin L Mallows. Non-null ranking models. i. *Biometrika*, 44(1/2):114–130, 1957.

[96] Federico Malucelli, Thomas Ottmann, and Daniele Pretolani. Efficient labelling algorithms for the maximum noncrossing matching problem. *Discrete Applied Mathematics*, 47(2):175–179, 1993.

[97] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.

[98] Adam Marcus, Eugene Wu, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *Fifth Biennial Conference on Innovative Data Systems Research, CIDR 2011, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 211–214. www.cidrdb.org, 2011.

[99] Arya Mazumdar and Barna Saha. Clustering via crowdsourcing. *arXiv preprint arXiv:1604.01839*, 2016.

[100] Christopher McComb, Kosa Goucher-Lambert, and Jonathan Cagan. Impossible by design? fairness, strategy, and arrow's impossibility theorem. *Design Science*, 3, 2017.

[101] Iain McLean. The borda and condorcet principles: three medieval applications. *Social Choice and Welfare*, 7(2):99–108, 1990.

[102] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[103] Reshef Meir. Plurality voting under uncertainty. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[104] Reshef Meir, Maria Polukarov, Jeffrey S. Rosenschein, and Nicholas R. Jennings. Convergence to equilibria in plurality voting. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[105] Samuel Merrill III. A comparison of efficiency of multicandidate electoral systems. *American Journal of Political Science*, pages 23–48, 1984.

[106] Chaitanya Mishra and Nick Koudas. Interactive query refinement. In Martin L. Kersten, Boris Novikov, Jens Teubner, Vladimir Polutin, and Stefan Manegold, editors, *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, volume 360 of *ACM International Conference Proceeding Series*, pages 862–873. ACM, 2009.

[107] Tomomi Mitsuishi, Atsuyuki Morishima, Norihide Shinagawa, and Hideto Aoki. Efficient evaluation of human-powered joins with crowdsourced join pre-filters. In *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, pages 1–6, 2013.

[108] Vicente Rodríguez Montequín et al. Using myers-briggs type indicator (MBTI) for assessment success of student groups in project based learning. In *CSEDU*, 2010.

[109] Davide Mottin, Alice Marascu, Senjuti Basu Roy, Gautam Das, Themis Palpanas, and Yannis Velegrakis. A probabilistic optimization framework for the empty-answer problem. *Proceedings of the VLDB Endowment*, 6(14):1762–1773, 2013.

[110] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 635–646. ACM, 2006.

[111] Kyriakos Mouratidis and Bo Tang. Exact processing of uncertain top-k queries in multi-criteria settings. *Proceedings of the VLDB Endowment*, 11(8):866–879, 2018.

[112] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664–667, 2007.

[113] Lionel S Penrose. The elementary statistics of majority voting. *Journal of the Royal Statistical Society*, 109(1):53–57, 1946.

[114] Julien Pilourdault, Sihem Amer-Yahia, Dongwon Lee, and Senjuti Basu Roy. Motivation-aware task assignment in crowdsourcing. In Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß, editors, *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, pages 246–257. OpenProceedings.org, 2017.

[115] Habibur Rahman et al. Worker skill estimation in team-based tasks. *PVLDB*, 2015.

[116] Habibur Rahman, Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Task assignment optimization in collaborative crowdsourcing. In Charu C. Aggarwal, Zhi-Hua Zhou, Alexander Tuzhilin, Hui Xiong, and Xindong Wu, editors, *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 949–954. IEEE Computer Society, 2015.

[117] Habibur Rahman, Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Optimized group formation for solving collaborative tasks. *The VLDB Journal*, 28(1):1–23, 2019.

[118] Syama Sundar Rangapuram, Thomas Bühler, and Matthias Hein. Towards realistic team formation in social networks based on densest subgraphs. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo Baeza-Yates, and Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1077–1088. International World Wide Web Conferences Steering Committee / ACM, 2013.

[119] Syama Sundar Rangapuram, Thomas Bühler, and Matthias Hein. Towards realistic team formation in social networks based on densest subgraphs. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo Baeza-Yates, and Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1077–1088. International World Wide Web Conferences Steering Committee / ACM, 2013.

[120] Christopher Re, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 886–895. IEEE, 2007.

[121] Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Aggregating preferences cannot be fair. *Intelligenza Artificiale*, 2(1):30–38, 2005.

[122] Senjuti Basu Roy and Kaushik Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 361–372. ACM, 2011.

[123] Senjuti Basu Roy, Laks V. S. Lakshmanan, and Rui Liu. From group recommendations to group formation. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1603–1616. ACM, 2015.

[124] Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, Gautam Das, and Cong Yu. Exploiting group recommendation functions for flexible preferences. In Isabel F. Cruz, Elena Ferrari, Yufei Tao, Elisa Bertino, and Goce Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 412–423. IEEE Computer Society, 2014.

[125] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In Ignac Lovrek, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems, 12th International Conference, KES 2008, Zagreb, Croatia, September 3-5, 2008, Proceedings, Part III*, volume 5179 of *Lecture Notes in Computer Science*, pages 67–75. Springer, 2008.

[126] Anand D Sarwate, Stephen Checkoway, and Hovav Shacham. Risk-limiting audits and the margin of victory in nonplurality elections. *Statistics, Politics, and Policy*, 4(1):29–64, 2013.

[127] Amartya Sen. Social choice theory. *Handbook of mathematical economics*, 3:1073–1181, 1986.

[128] Ivan Srba and Maria Bielikova. Dynamic group formation as an approach to collaborative learning support. *TLT*, 2015.

[129] Julia Stoyanovich, Ke Yang, and H. V. Jagadish. Online set selection with fairness and diversity constraints. In Michael H. Böhlen, Reinhard Pichler, Norman May, Erhard Rahm, Shan-Hung Wu, and Katja Hose, editors, *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, pages 241–252. OpenProceedings.org, 2018.

[130] Junhua Tang, Sisi Dai, Jianhua Li, and Shenghong Li. Gossip-based scalable directed diffusion for wireless sensor networks. *International Journal of Communication Systems*, 24(11):1418–1430, 2011.

[131] James Tanton. *Encyclopedia of mathematics.* Facts On File, inc., 2005.

[132] Qian Tao, Yuxiang Zeng, Zimu Zhou, Yongxin Tong, Lei Chen, and Ke Xu. Multi-worker-aware task planning in real-time spatial crowdsourcing. In *International Conference on Database Systems for Advanced Applications*, pages 301–317. Springer, 2018.

[133] Nicolaus Tideman. The single transferable vote. *Journal of Economic Perspectives*, 9(1):27–38, 1995.

[134] Robert Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32(3):323–330, 1980.

[135] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon M. Kleinberg. Structural diversity in social contagion. *Proc. Natl. Acad. Sci. USA*, 109(16):5962–5966, 2012.

[136] Anke van Zuylen and David P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In Christos Kaklamanis and Martin Skutella, editors, *Approximation and Online Algorithms, 5th International Workshop, WAOA 2007, Eilat, Israel, October 11-12, 2007. Revised Papers*, volume 4927 of *Lecture Notes in Computer Science*, pages 260–273. Springer, 2007.

[137] Anke Van Zuylen and David P Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009.

[138] Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545–576, 2012.

[139] Noreen M Webb. Peer interaction and learning in small groups. *International journal of Educational research*, 13(1):21–39, 1989.

[140] Dong Wei, Md Mouinul Islam, Schieber Baruch, and Senjuti Basu Roy. Rank aggregation with proportionate fairness. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, 2022.

[141] Dong Wei, Ioannis Koutis, and Senjuti Basu Roy. Peer learning through targeted dynamic groups formation. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 121–132. IEEE, 2021.

[142] Hyeongon Wi, Seungjin Oh, Jungtae Mun, and Mooyoung Jung. A team formation model based on knowledge and collaboration. *Expert Systems with Applications*, 36(5):9121–9134, 2009.

[143] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. A survey of human-in-the-loop for machine learning. *arXiv preprint arXiv:2108.00941*, 2021.

[144] Lirong Xia. Computing the margin of victory for various voting rules. In Boi Faltings, Kevin Leyton-Brown, and Panos Ipeirotis, editors, *Proceedings of the 13th ACM Conference on Electronic Commerce, EC 2012, Valencia, Spain, June 4-8, 2012*, pages 982–999. ACM, 2012.

[145] Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*, pages 22:1–22:6. ACM, 2017.

[146] Peyton Young. Optimal voting rules. *Journal of Economic Perspectives*, 9(1):51–64, 1995.

[147] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. Fa*ir: A fair top-k ranking algorithm. In Ee-Peng Lim, Marianne Winslett, Mark Sanderson, Ada Wai-Chee Fu, Jimeng Sun, J. Shane Culpepper, Eric Lo, Joyce C. Ho, Debora Donato, Rakesh Agrawal, Yu Zheng, Carlos Castillo, Aixin Sun, Vincent S. Tseng, and Chenliang Li, editors, *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 1569–1578. ACM, 2017.

[148] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. Crowdsourced top-k algorithms: An experimental evaluation. *Proceedings of the VLDB Endowment*, 9(8):612–623, 2016.

[149] Haichao Zheng, Dahui Li, and Wenhua Hou. Task design, motivation, and participation in crowdsourcing contests. *International Journal of Electronic Commerce*, 15(4):57–88, 2011.

[150] ARMEN Zzkarian and Andrew Kusiak. Forming teams: an analytical approach. *IIE transactions*, 31(1):85–97, 1999.