ABSTRACT

MODELS AND ALGORITHMS FOR PROMOTING DIVERSE AND FAIR QUERY RESULTS

by
Md Mouinul Islam

Ensuring fairness and diversity in search results are two key concerns in compelling search and recommendation applications. This work explicitly studies these two aspects given multiple users' preferences as inputs, in an effort to create a single ranking or top-k result set that satisfies different fairness and diversity criteria. From group fairness standpoint, it adapts demographic parity like group fairness criteria and proposes new models that are suitable for ranking or producing top-k set of results. This dissertation also studies equitable exposure of individual search results in long tail data, a concept related to individual fairness. First, the dissertation focuses on aggregating ranks while achieving proportionate fairness (ensures proportionate representation of every group) for multiple protected groups. Then, the dissertation explores how to minimally modify original users' preferences under plurality voting, aiming to produce top-k result set that satisfies complex fairness constraints. A concept referred to as *manipulation by modifications is introduced, which involves making minimal changes to the original user preferences to ensure query satisfaction.* This problem is formalized as the margin finding problem. A follow up work studies this problem considering a popular ranked choice voting mechanism, namely, the Instant Run-off Voting or IRV, as the preference aggregation method. From the standpoint of individual fairness, this dissertation studies an exposure concern that top-k set based algorithms exhibit when the underlying data has long tail properties, and designs techniques to make those results equitable. For result diversification, the work studies efficiency opportunities in existing diversification algorithms, and designs a generic access primitive called DivGetBatch() to enable that. The contributions

of this dissertation lie in (a) formalizing principal problems and studying them analytically. (b) designing scalable algorithms with theoretical guarantees, and (c) extensive experimental study to evaluate the efficacy and scalability of the designed solutions by comparing them with the state-of-the-art solutions using large-scale datasets.

MODELS AND ALGORITHMS FOR PROMOTING DIVERSE AND FAIR QUERY RESULTS

by
Md Mouinul Islam

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

Department of Computer Science

August 2023

APPROVAL PAGE

MODELS AND ALGORITHMS FOR PROMOTING DIVERSE AND FAIR QUERY RESULTS

Md Mouinul Islam

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

Dr. Baruch Schieber, Committee Member — Date
Professor, Department of Computer Science, NJIT

Dr. Ioannis Koutis, Committee Member — Date
Associate Professor, Department of Computer Science, NJIT

Dr. Yi Chen, Committee Member — Date
Professor, Business Data Science, NJIT

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

BIOGRAPHICAL SKETCH

Author: Md Mouinul Islam

Degree: Doctor of Philosophy

Date: August 2023

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2023
- Bachelor of Science in Electrical and Electronic Engineering,
  Bangladesh University of Engineering and Technology, Bangladesh, 2013

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 2022).*

Md Mouinul Islam, Dong Wei, Baruch Schieber, and Senjuti Basu Roy, "Satisfying complex top-k fairness constraints by preference substitutions" *Proceedings of the VLDB Endowment (PVLDB 2023).*

Md Mouinul Islam, Mahsa Asadi, Sihem Amer-Yahia, and Senjuti Basu Roy, "A generic framework for efficient computation of top-k diverse results" *The International Journal on Very Large Data Bases (VLDB Journal 2023).*

Md Mouinul Islam, Mahsa Asadi, and Senjuti Basu Roy, "Equitable top-k results for long tail data" *(Under revision) Proceedings of the 2024 International Conference on Management of Data (SIGMOD 2024).*

Md Mouinul Islam, Soroush Vahidi, Baruch Schieber, and Senjuti Basu Roy, "Satisfying constrained multiple selection queries using IRV" *(Under submission) Proceedings of the 2024 International Conference on Management of Data (SIGMOD 2024).*

To my beloved family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude and appreciation to the following individuals who have contributed significantly to the completion of my PhD dissertation.

First and foremost, I extend my sincere gratitude to my dissertation advisor, Dr. Senjuti Basu Roy. Your guidance, expertise, and unwavering support throughout the research process have been invaluable. Your insightful feedback and encouragement have greatly shaped the outcome of this work, and I am truly grateful for the opportunity to have worked under your mentorship.

My sincere appreciation goes to the members of my dissertation committee, Dr. Baruch Schieber, Dr. Ioannis Koutis, Dr. Yi Chen, and Dr. Sihem Amer-Yahia. I am honored to have had the privilege of benefiting from your expertise and thoughtful feedback. Your invaluable guidance and constructive criticism have significantly shaped the development and quality of this research.

I am deeply grateful for the financial support provided by National Science Foundation (CAREER Award #1942913, IIS #2007935, IIS #1814595, PPoSS: Planning #2118458). These grants played a crucial role in supporting my research and was instrumental in enabling the completion of this dissertation. Without this valuable funding, the work presented in this thesis would not have been possible.

I would also like to thank my colleagues, Mahsa Asadi, Sepideh Nikookar, Dong Wei, and Sohrab Namazi Nia, for their collaboration, insightful discussions, and support throughout this journey. Your contributions and friendship have been invaluable, and I am grateful for the shared experiences and mutual support we have provided to one another.

Lastly, I want to express my deepest gratitude to my family, and in particular, to my wife, Khondoker Takia Zaman. Your unwavering support, love, and understanding

have been the bedrock of my academic journey. Your patience, encouragement, and belief in me have been instrumental in overcoming the challenges I faced throughout this endeavor. I am truly blessed to have you by my side, and I cannot thank you enough for your constant support and sacrifices.

To all those mentioned above, as well as to anyone else who has contributed to my academic and personal growth, I extend my heartfelt appreciation. Your support and encouragement have made this achievement possible, and I am forever grateful for the role each of you has played in my journey toward earning my Ph.D.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF TABLES
(Continued)

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Overview

Given a user query over a large database, the number of records that satisfy query constraints may be potentially large. Traditional user interfaces, on the other hand, are highly restrictive, and are designed to accommodate a small number of those results. This gives rise to a need to present users with a smaller set of results, known as top-k results [118]. To do this, a substantial amount of related works exist that explore how to design effective ranking functions and algorithms [73, 114, 118, 194]. It has been acknowledged recently that the existing ranking and top-k algorithms are to be revisited to enhance them with criteria that explicitly increase representation of historically disadvantaged populations, or to improve the overall representativeness of the selected set. Fairness and diversification are two such criteria [57, 58, 60, 206, 214], that we investigate extensively in this dissertation.

1.1.1 Background

Fairness. Fairness in query results is a critical objective in various applications, including electoral system [94], ranking and recommendation [101, 104, 134]. This dissertation explores the challenges and solutions associated with ensuring fair query results. Most approaches to algorithmic fairness interpret fairness as lack of discrimination [99] seeking *that an algorithm should not discriminate against its input entities based on attributes that are not relevant to the task at hand*. Such attributes are called protected, or sensitive, and often include among others gender, religion, age, sexual orientation, and race. So far, most work on defining, detecting, and removing unfairness has focused on classification algorithms [212, 216] used in decision-making. W.r.t ranking and top-k results, the algorithmic fairness literature deals with group fairness along the lines of statistical or demographic parity which is typically expressed by means of some fairness constraint requiring that the top-k results (for any k) to contain enough records from some groups that are protected [18, 94, 104, 123, 139, 163, 178, 197, 207, 214, 217]. Proportionate fairness (p-fairness) [26, 184] is another group fairness criteria that is close to statistical parity [67] studied in the context of group fairness. P-fairness is more stronger than statistical parity, because it ensures statistical parity for every position in the ranked order. Individual fairness, on the other hand, intends to ensure "similar individuals are treated similarly". A classifier is individually fair if the distance between probability distributions mapped by the classifier is not greater than the actual distance between the records [81]. In the context of item-fairness in ranking and top-k, it ensures that items should receive the amount of exposure proportional to their relevance [61]. This dissertation aims to return query results that ensure complex fairness constraints such as demographic parity, p-fairness, or individual fairness.

Preference aggregation. Preference of the individual users can be elicited as pairwise comparison [74], in form of a binary vector [173] known as Approval Voting [46], in an ordinal scale [13, 132], or considering Arrowian social choice, where users provide partial or complete preference order over the items [40, 52, 130, 177]. Similarly, The properties of social welfare functions for aggregating preferences have been studied by mathematicians since the 18th century [51, 63, 66]. Different preference aggregation methods are proposed, including majority voting, plurality voting [136, 152, 161], their weighted versions, as well as aggregation methods that consider positional preference [40, 52, 177], such as Kemeny rule [82, 127], Condorcet rule [69], Borda Count [84], or Instant Run-off Voting (IRV) [62, 149]. This dissertation studies the problems of finding the fair aggregated rank or top-k, considering three aggregation methods such as Kemeny, plurality, and IRV.

Top-k. This dissertation also identifies fairness and diversification concern in existing top-k algorithms, that return a "fixed" set of k results for a given query. Given a user query, a top-k result contains k records that have the highest scores [167]. Scores are computed based on relevance, diversity, newness, serendipity, etc. Designing effective scoring functions as well as efficient algorithms [1, 2] lend to numerous applications in recommendation and search [4, 50, 53, 86, 138, 192, 195] and is an active area of research. In this dissertation, we focus on both set based notion of top-k result as well as top-k defined w.r.t ranked order.

1.2 Motivations

In the context of fairness, this dissertation's first focus is on rank aggregation [9, 82, 186] considering Kemeny distance as an aggregation method with a specific emphasis on proportionate fairness or p-fairness [26, 184]. 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 [9, 82, 186]. The first goal in this work is to revisit the rank aggregation problem considering *proportionate fairness or p-fairness* [26, 184] that ensures proportionate representation of every group based on a protected attribute in every position of the aggregated ranked order. The problem 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 $f(p)$.

The second preference aggregation model studied in this dissertation is popular plurality voting [136, 152, 161], with an emphasis on adopting group fairness definitions [18, 94, 104, 123, 139, 163, 178, 197, 207, 214, 217]. The problem is studied 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. 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 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.* The process of minimizing initial preference modification, in general, is known as margin finding in the literature [41, 56, 176].

Finally, we study the margin finding problem considering a popular ranked choice voting mechanism as the underlying preference aggregation method, namely Instant Run-off Voting (IRV) [62,149]. IRV is chosen for its ability to promote fairness compared to other existing preference aggregation methods. One of the objectives of the research is to investigate techniques for modifying the original IRV ballots to adhere to query constraints while minimizing the number of required changes.

The inputs of this problem are made up of a set of *ballots*, one from each of the m users (voters), and each ballot is a ranked order of preference up to ℓ items from n items (candidates). The query comes with multiple (k) constraints. The goal is to recommend items that satisfy the query constraints and are most representative of the users' preferences. It is imperative that at times the original users' preferences may require further manipulation to meet query constraints. We consider manipulation by modifications, where a single modification amounts to changing any number of ℓ entries in an existing ballot. We formalize this as *the margin finding problem under modification* that minimizes the number of ballot modifications needed to guarantee that the results satisfy all the k query selection constraints.

It has been recognized that group fairness alone has its deficiencies [95]. In two independent efforts, Flanigan et al. [94] and Garcia-Soriano et al. [101] study how to enable equitable selection probability of the records under group fairness constraints and propose maxmin-fair distributions of ranking. Zemel et al. develop a learning algorithm for fair classification that ensures both group fairness and individual fairness [216]. [19] studies individual fairness in similarity search to ensure points within distance r from the given query have the same probability to be returned. However, none of these studied problems extend to top-k set based algorithms.

From the standpoint of individual fairness, we study how to redesign the existing set based top-k algorithms [167] such that the records returned to the users receive equitable exposure. This problem is studied in the context of long-tail data, where a small number of popular records that receive extensive user exposure, while there exist a long tail of niche records which may be equally desirable to the users but remain relatively unknown [203]. The work redesigns existing top-k algorithms to return multiple equivalent top-k sets to users, rather than a fixed set, with the goal of promoting equitable exposure of records (individual fairness). We adapt a political theory, namely, the *Sortition Act* [75, 180] and redesign existing top-k algorithms to have them compute *a set S of multiple top-k sets that are equivalent in utility* as opposed to a fixed top-k set. Given S, an end user still draws one of the sets at random. Hence, the goal is to assign a probability distribution over S, i.e., $PDF(S)$, such that after many such draws from many end users, the records returned inside the top-k sets have as uniform selection probability as possible. We formalize **θ-Equiv-top-k-MMSP** that produces $PDF(S)$ for a given query and a scoring function $\mathcal{F}$. Each set $s \in S$ contains k number of records whose score is at most $\theta\%$ (a tunable application dependent input parameter) smaller than the optimum top-k score, and the $PDF(S)$ is computed such that the selection probabilities of the records in it are as uniform as possible. Enabling equal selection probabilities promotes equal exposure of the records. **θ-Equiv-top-k-MMSP** is rooted on maxmin fairness theory that maximizes the minimum exposure.

Finally, this dissertation studies how to redesign existing result diversification algorithms to make them faster without having to compromise with their accuracy. Result diversification remains to be an active research topic with extensive applications in recommendation and search [1, 2, 4, 50, 141, 166, 167, 185, 188, 189, 193, 195]. Diversification algorithms aim to provide query results that are both relevant and cover a wide range of user intents. Previous research has studied returning top-k diverse results, with works proposing objective functions and algorithms to balance relevance and diversity [1, 2, 86, 166, 188, 193]. Traditional diversification algorithms, such as GMM [106], MMR [106], SWAP [211], rely on iterative processes and pairwise diversity computations, which can be computationally expensive for large databases. Although some works address this issue in geometric spaces [98, 156], the need for costly computations remains when diversity functions are arbitrary or non-metric. This research aims to enhance the efficiency of promoting diverse query results by reducing computational overhead and considering diversity functions as arbitrary or non-metric, thereby allowing for comprehensive and varied outcomes that align with user interests and preferences.

1.3 Contributions

This dissertation makes several non-trivial contributions in designing effective models, and principled algorithms for promoting diverse and fair query results.

Chapter 2 of the dissertation focuses on rank aggregation with proportionate fairness, explicitly addressing the concept of p-fairness for ensuring proportionate representation of different groups based on protected attributes in aggregated ranked orders. The Rank Aggregation under Proportionate Fairness (**RAPF**) problem aims to minimize disagreements among individual rankings while ensuring that the representation of each group in the aggregated ranking aligns with their representation in the original data for every position. The dissertation acknowledges that **RAPF** is NP-hard and introduces two computational frameworks: RandAlgRAPF, a highly scalable randomized algorithm, and AlgRAPF, a deterministic algorithm that provides a solution for **RAPF**. Both algorithms rely on achieving p-fair Kemeny optimized rankings for individual rankings, which is referred to as the Individual p-Fairness (**IPF**) problem. The dissertation presents several algorithmic contributions: (i) proving that when the protected attribute is binary, **IPF** can be solved exactly using a greedy technique; (ii) highlighting the non-triviality of solving **IPF** when the protected attribute is non-binary; (iii) introducing two solutions for **IPF**, ExactMultiValuedIPF (optimal) and ApproxMultiValuedIPF (2-approximation factor), resulting in 3 and 4 approximation factors, respectively, for the **RAPF** problem. (iv) demonstrating that ApproxMultiValuedIPF and AlgRAPF achieve an $\alpha + 2$ approximation factor if **IPF** can be solved with an approximation factor α; The proposed solutions are extensively evaluated using multiple real-world and large-scale synthetic datasets. Comparative experiments against state-of-the-art approaches demonstrate the effectiveness and efficiency of the studied problem and the proposed solutions.

Chapter 3 of the dissertation concentrates on the margin finding problem in the context of single ballot substitutions and explores various settings of protected group attributes to promote fairness. The objective is to optimize preference substitutions by minimizing the number of single ballot substitutions while satisfying complex top-k fairness constraints. The chapter formalizes several margin finding problems that consider different types of protected attributes: MFBinaryS for binary attributes, MFMultiS for multi-valued attributes, MFMulti2 for two different protected attributes, and MFMulti3+ for three or more protected attributes. The theoretical analysis of these problems is presented, accompanied by principled algorithmic contributions. The computational complexity of the defined problems is analyzed. MFBinaryS and MFMultiS are proven to be computationally easy. For MFMulti2, the decision version is shown to be (weakly) NP-hard by reducing it to the Partition problem. As for MFMulti3+, the satisfiability problem is proven to be (strongly) NP-hard through a reduction from the three-dimensional matching (3DM) problem. To evaluate the proposed solutions, rigorous large-scale experiments are conducted using real-world datasets related to election and movie applications, as well as synthetic datasets. Multiple state-of-the-art solutions are adapted and compared against the proposed approaches. The experimental results validate the effectiveness of the designed solutions and their relevance to practical scenarios.

Chapter 4 of this dissertation studies how to modify the original ballots of IRV to satisfy all k query constraints such that the total number of required ballot changes is minimized (**MqKIRV** in short). We prove that **MqIRV** is NP-hard, even when the ballot size is at most $\ell = 2$, by reducing an instance of the known NP-complete problem Exact Cover by 3-Sets (X3C) to an instance of **MqIRV**. Inspired by [42, 142], we then design an algorithmic framework AlgExact that considers all possible permutations over the candidates, where each permutation represents an elimination order simulating multiple run-off rounds of IRV. Solving AlgExact requires repeatedly solving a sub-problem DistTo, which finds the smallest number of ballot modifications to satisfy that order. Unfortunately, we prove that even the decision version of DistTo is NP-hard by reducing an instance of X3C to DistTo, even when $\ell = 3$. The basic idea behind AlgExact is to repeatedly invoke DistTo for every possible permutation and retain the permutation that requires the smallest number of ballot modifications overall as the answer. We further study efficiency opportunities of AlgExact by enabling early terminations. The aim is to avoid making expensive DistTo calls and instead compute a lower bound of margin for every possible suffix over all permutations. If the lower bound of margin for a permutation is not smaller than the current upper bound of margin over the instance of **MqIRV**, the permutation is eliminated entirely. To that end, we design a highly efficient lower bound computation algorithm DistToLB and an upper bound computation algorithm **MqIRVUB** that are both highly effective and computationally lightweight. In addition, we present an efficient exact solution, DistToAddAlg, for the DistTo problem, which focuses on adding the smallest number of ballots to the existing set of ballots to satisfy the query constraints. We also propose an integer programming formulation, IPEx, for **MqIRV** that is non-trivial. Lastly, we develop a highly scalable heuristics algorithm, AlgApprx, that demonstrates good performance in practice. Furthermore, this work includes experimental evaluations using real-world and synthetic datasets. The findings indicate that **MqIRV** yields significantly smaller anti-plurality index compared to alternative approaches, such as plurality voting based margin computation. The results also demonstrate that AlgExact is not only optimal but also more scalable than state-of-the-art solutions. Moreover, the experiments validate the optimality and scalability of DistToAddAlg, as well as the effectiveness and scalability of

AlgApprx, by varying relevant parameters and comparing them with appropriate baseline algorithms. These experimental evaluations provide empirical evidence of the efficiency, effectiveness, and scalability of the proposed solutions in practical scenarios.

Chapter 5 of this dissertation investigates how to promote equitable exposure to records that satisfy long tail criteria. Firstly, we formalize key definitions related to **θ-Equiv-top-k-Sets** and selection probabilities of records. We introduce the **θ-Equiv-top-k-MMSP** problem, which aims to generate **θ-Equiv-top-k-Sets** and maximize the minimum selection probability of a record. We prove that this problem is NP-Complete. Next, we propose the **OptTop-k-θ** algorithm, which is an exact solution for generating **θ-equivalent top-k sets**. It utilizes an efficient data structure based on item lattices to maintain candidate top-k sets and calculate their score bounds. We also present the **Opt-SP** algorithm, which provides an exact solution for generating the probability distribution function (PDF) of the **θ-equivalent top-k sets**. To address scalability, we introduce the **RWalkTop-k-θ** algorithm, which utilizes random walks to probabilistically generate θ-equivalent top-k sets. The algorithm leverages the Good Turing Test to determine when to stop the random walk and discover all **θ-Equiv-top-k-Sets** with high probability. We also present the **Gr-SP** algorithm, which produces a probability distribution over the generated sets. Additionally, we propose the **ARWalkTop-k-θ** algorithm, an adaptive random walk based approach that solves **θ-Equiv-top-k-Sets** and **MaxMinFair** at the same time. It lowers the probability of records already part of valid sets and boosts the probability of remaining records to generate θ-equivalent top-k sets. This adaptive random walk approach ensures a uniform probability distribution over the generated sets. We conduct extensive evaluations using real-world and synthetic datasets, comparing our designed solutions against baseline algorithms. The experimental results validate the quality and scalability of our proposed solutions and support our theoretical analyses.

Chapter 6 of this dissertation investigates the result diversification problem and proposes a computational solution to expedite existing top-k algorithms designed for result diversification. Firstly, we address a major computational bottleneck in existing diversification algorithms and introduce an accelerated process called **DivGetBatch()**. By grouping records instead of comparing record pairs, we significantly improve the computation speed. We develop a generic computation framework, including the I-tree index structure and other auxiliary data structures, to facilitate this improvement in speed. Our contribution lies in creating an indexing technique that is easily designed using popular record partitioning algorithms and is compatible with various diversification algorithms and functions. Secondly, we enhance the *MMR*, *GMM*, and *SWAP* [55, 106, 211] algorithms by incorporating the **DivGetBatch()** approach. By operating on pairs of groups of records instead of individual record pairs, these augmented algorithms achieve faster running times while maintaining identical top-k results. We provide theoretical analysis on the exactness and running time of these augmented algorithms, showcasing their improved efficiency. For example, the augmented SWAP algorithm (Aug-SWAP) exhibits a significantly faster running time compared to the original algorithm. Our third contribution focuses on the design and maintenance of the I-tree index structure. We address the computational bottleneck in updating the MinsimMatrixNode and MaxsimMatrixNode data structures and formulate an optimization problem to minimize the number of updates. We propose an exact solution, **OPTMn**, based on integer programming, and a scalable greedy heuristic, **GrMn**, for efficient index maintenance. Lastly, we conduct extensive experimental evaluations using large real-world datasets and a synthetic dataset. The results demonstrate that our augmented algorithms produce identical results to the original algorithms while achieving substantial speedups ranging from $3\times$ to $24\times$ on large datasets. We compare the I-tree index structure with existing indexing structures such as M-Tree [68], KD-Tree [34], and Ball-Tree [135] and find that

I-tree consistently outperforms them in terms of query processing speed and index construction time. These experiments provide empirical evidence of the efficiency and effectiveness of our proposed solutions in practical scenarios.

Chapter 7 summarizes the contributions of this research and outlines ongoing and future research problems. As an ongoing work, it proposes how to select top-k features for different subgroups (subgroups are defined based on different protected attribute value combinations) for datasets that are heavily incomplete. Traditional feature selection techniques fall short to estimate feature importance in such cases. This ongoing work investigates the applicability of machine learning models, such as, graphical neural network to estimate “importance” of features for different subgroups. Such a technique could be highly useful in many compelling applications, such as, personalized recommendation systems, targeted marketing, and group-based analysis, to name a few.

CHAPTER 2

RANK AGGREGATION WITH PROPORTIONATE FAIRNESS

2.1 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 [9,82,186]. Our goal in this work is to revisit the rank aggregation problem considering a notion of fairness, namely *proportionate fairness or p-fairness* [26, 184] 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* [21, 184] 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. Subsection 2.1.1 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 [104] with weaker notion of fairness and the designed solutions are just heuristic. We investigate how a solution designed for **IPF** could solve **RAPF**.

2.1.1 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 2.1) 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 2.1 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 *[81] is somewhat similar to p-fairness, however, the best adapted version of top-k statistical parity studied in a recent paper [134] does not account for proportionate representation in every position of the top-k, limiting its applicability.*

2.1.2 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 2.2)** considering binary ($\ell = 2$) and multi-valued ($\ell > 2$) protected attributes.

Our second contribution is theoretical and algorithmic. 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 [104] for **IPF** are heuristics and do not ensure optimality (refer to Subsection 2.4.1 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 [48], that admits a 2 approximation factor.

Since rank aggregation problem under Kemeny Optimization is NP-hard for 4 or more lists $[9, 82, 186]$, **RAPF** is also NP-hard. 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 $[9, 82, 186]$ 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 ExactMultiValuedIPF

Table 2.2 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	$O(m^2 n \log n + mn^{2.5}\log n)$

admits a 3 approximation factor; whereas, it admits a 4 approximation factor when ApproxMultiValuedIPF is used instead. Table 2.2 summarizes our theoretical results.

Our third contribution is experimental (Section 2.3). 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 [104] for **IPF** and FairILP [134] 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 [104,134] - 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 2.3 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

2.2 Preliminaries and Formalism

Database. contains n items or candidates. These two terms will be used interchangeably in the paper. 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 [9, 82]. 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 [77] 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 paper, 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 2.4 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 [8, 9, 81]. 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 [134] work. Other distance measures are briefly described in Section 2.4.

Definition 4. Proportionate Fair or p-fair ranking [26, 184]. *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 2.4 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.

2.2.1 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 $m \geq 4$ [9].

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 2.4). These choices and other alternative ways of incorporating fairness inside rank aggregation are explored in Section 2.5.

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 Subsection 2.3.4.

2.3 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

Table 2.4 Rank Aggregation Results of Comparable Methods Using Subsection 2.1.1 Example Considering Gender as The Protected Attribute

Rank	**Rank aggregation (without fairness)**	**Rank aggregation (with statistical parity) [134]**	**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

with multiple state-of-the-art solutions [104, 134] 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].

2.3.1 Dataset description

We perform evaluations considering three real world datasets. (a) Fantasy players choose real athletes for their fantasy teams and generate scores based on the athlete's real 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 [207], 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 2.5 has further details.

[1] `https://github.com/MouinulIslamNJIT/Rank-Aggregation_Proportionate_Fairness.git`

Table 2.5 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 [134,207] using Mallows' Model [145]. The Kemeny rank aggregation has been shown to be a maximum likelihood estimator for this model [207]. 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 [134] for further details. The θ and p are set to 0.9 and 0.7 respectively in our experiments.

2.3.2 Implemented algorithms

DetConstSort [104] 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 Subsection 2.4.1, it neither guarantees smallest Kendall-Tau distance nor ensures p-fairness. We implement this for **IPF**.

FairILP [134] 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.

2.3.3 Summary of results

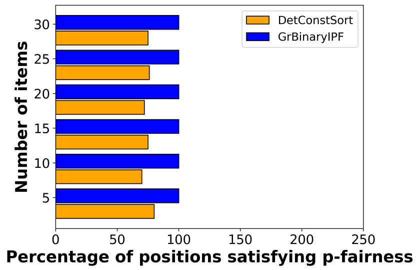
Our first observation is that, consistent with our theoretical analysis, p-fairness promotes stronger notion of fairness, by ensuring proportionate representation of each of the protected attribute values for every position in the ranked order. Naturally, incorporating p-fairness inside rank aggregation 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 [104,134] - 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.

2.3.4 Quality experiments

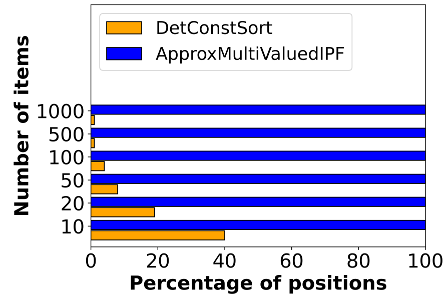
In this section we describe the results of our qualitative analysis.

BinaryIPF Results Figures 2.1a and 2.2a compare the fairness of GrBinaryIPF and DetConstSort. These results clearly indicate that GrBinaryIPF consistently satisfies p-fairness, whereas, DetConstSort does not.

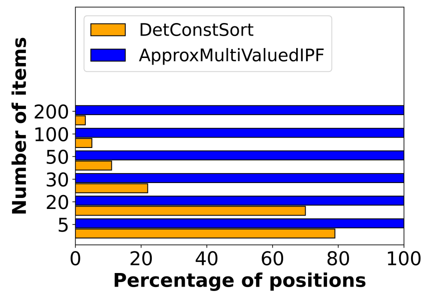
Figure 2.3a compares the Kendall-Tau distance between the input ranking and the ranking computed by OptIPF, GrBinaryIPF, and DetConstSort. Consistent with our theoretical analysis OptIPF and GrBinaryIPF always produce the same distance. At times DetConstSort computes a ranking with a smaller distance. This can indeed happen, as DetConstSort does not necessarily compute a p-fair ranking.



(a) Fantasy football: GrBinaryIPF vs DetConstSort



(b) German Credit: ApproxMultiValuedIPF vs DetConstSort

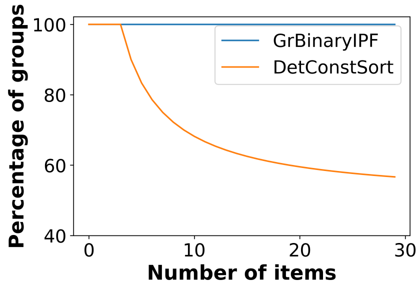


(c) MovieLens: ApproxMultiValuedIPF vs DetConstSort

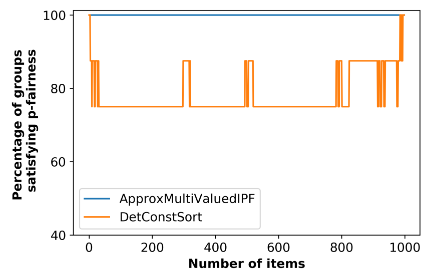
Figure 2.1 Percentage of positions satisfying p-fairness (IPF).

Figure 2.4a 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.

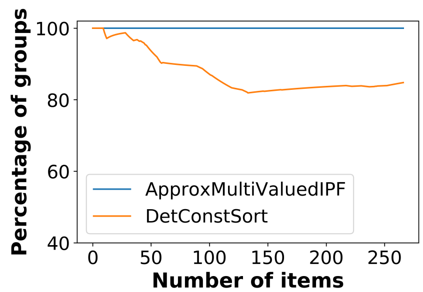
MultiValuedIPF Results We use the MovieLens and German Credit Score datasets to demonstrate the effectiveness of our proposed solution ApproxMultiValuedIPF and compare it with DetConstSort. Figures 2.1b, 2.1c, 2.2b, and 2.2c demonstrate that also in this case ApproxMultiValuedIPF consistently satisfies p-fairness whereas DetConstSort fails to satisfy p-fairness. Figures 2.3b, 2.3c compares the Kendall-Tau distance between the input ranking and the ranking computed by ApproxMultiValuedIPF and DetConstSort.



(a) Fantasy football: GrBinaryIPF vs DetConstSort



(b) German Credit: ApproxMultiValuedIPF vs DetConstSort



(c) MovieLens: ApproxMultiValuedIPF vs DetConstSort

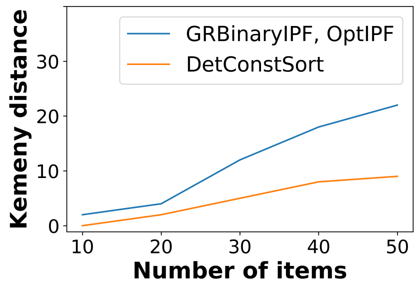
Figure 2.2 Percentage of groups satisfying p-fairness (**IPF**).

Again, at times DetConstSort computes a ranking with a smaller distance since DetConstSort does not necessarily compute a p-fair ranking.

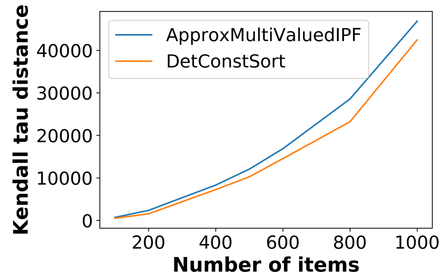
Figures 2.4b, 2.4c plot the Kendall-Tau distance of the rankings by ApproxMultiValuedIPF, as we relax the p-fairness using $\delta \geq 0$. Unsurprisingly, for large δ, the Kendall-Tau values become 0.

RAPF Results Next, we evaluate the **RAPF** problem by studying the effectiveness of our proposed AlgRAPF using GrBinaryIPF (Fantasy football) and ApproxMultiValuedIPF (MovieLens), and compare it with FairILP [134] and OptRAPF, whenever appropriate.

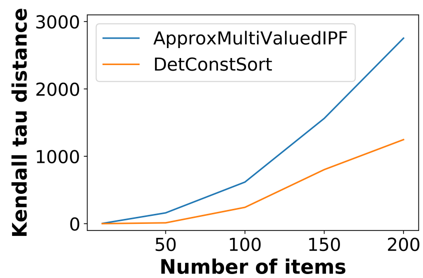
Figures 2.6a and 2.6b demonstrate that AlgRAPF consistently satisfies p-fairness whereas FairILP fails to satisfy p-fairness. Figures 2.7a and 2.7b compare the Kemeny distance between the input rankings and the aggregate ranking produced



(a) Fantasy football: OPTIPF, GRBINARYIPF , DETCONSTSORT



(b) German Credit: APPROXMULTIVALUEDIPF vs DETCONSTSORT



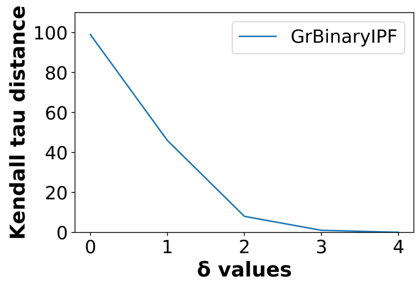
(c) MovieLens: APPROXMULTIVALUEDIPF vs DETCONSTSORT

Figure 2.3 Kendall-Tau distance **IPF**.

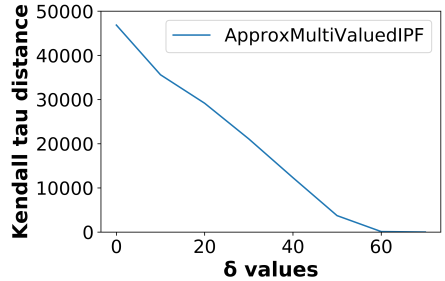
by ALGRAPF, RANDALGRAPF, FAIRILP, and OPTRA. As expected OPTRA achieves the smallest distance, followed by FAIRILP, since it does not require p-fairness, and then ALGRAPF and RANDALGRAPF. Algorithm RANDALGRAPF is inferior to ALGRAPF in practice, since its performance is same as the latter one only in expectation.

Figures 2.9b and 2.9a 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.

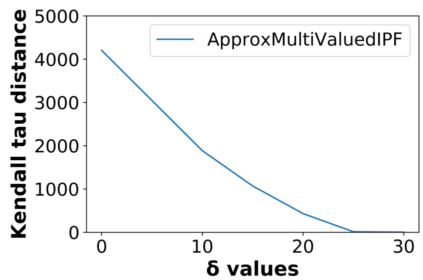
Finally, Table 2.6 presents the actual approximation factors of the different algorithms proposed in this work. Because of the exponential nature of the OPTIPF this comparison could be conducted only on small datasets. As evident from Table 2.6 the actual approximation factors are lower than the proven theoretical bounds.



(a) Fantasy Football: GrBinaryIPF



(b) German Credit : ApproxMultiValuedIPF



(c) Movie Lens: ApproxMultiValuedIPF

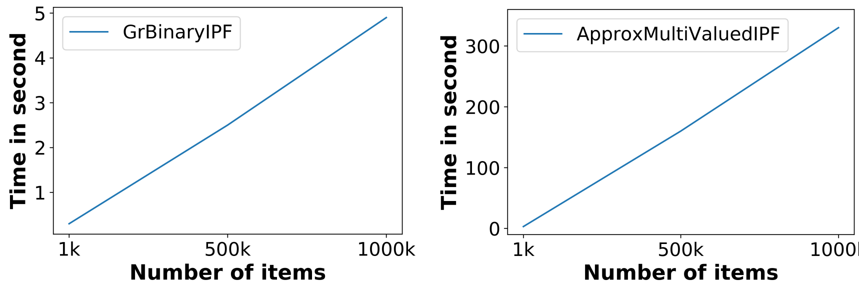
Figure 2.4 Varying δ analysis IPF.

2.3.5 Case study

For the case study, we use the ten popular movies based on five different IMDB users. All these movies belong to three 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 2.7 show the ranked order of the results based on FairILP [134] 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 ten ranked positions, thereby promoting improved user experience.

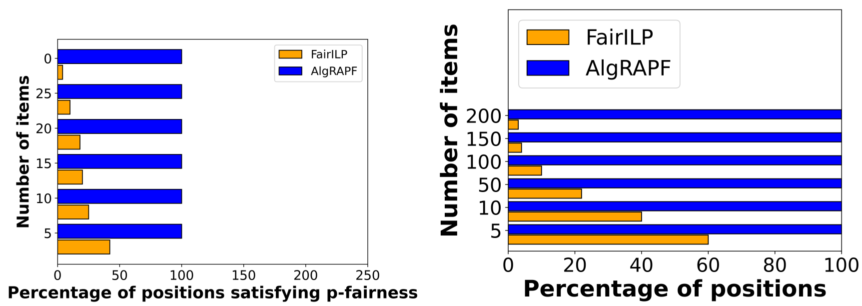
2.3.6 Scalability experiment

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 [104]



(a) GrBinaryIPF varying n (b) ApproxMulti-ValuedIPF varying n

Figure 2.5 Running time analysis of **IPF**.



(a) Fantasy football: p-fairness: AlgRAPF vs FairILP [134] (b) MovieLens: p-fairness: AlgRAPF vs FairILP [134]

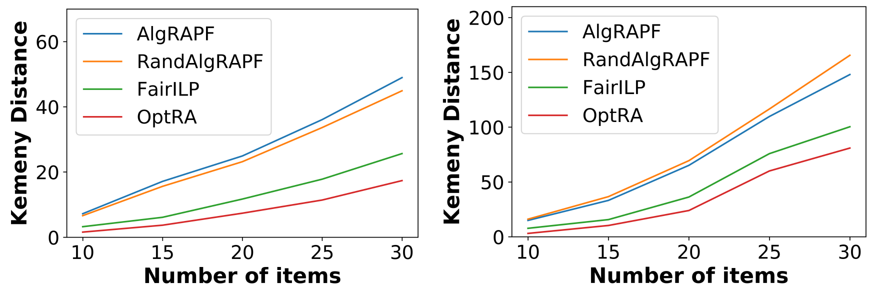
Figure 2.6 % of positions satisfying p-fairness (**RAPF**).

and FairILP [134] do not satisfy the p-fairness criteria; second, the baseline algorithm FairILP [134] is inherently not scalable. We use synthetically generated data using Mallows' model for this purpose. We vary n and m. Figures 2.5, and 2.8 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.

2.4 Related Work and Comparison

We primarily discuss four types of existing work that are related to our proposed problem.

Rank Aggregation. The rank aggregation study was initiated in the early 2000s by Dwork et. al. [82]. Since then, rank aggregation and several of its



(a) Fantasy football: AlgRAPF vs RandAlgRAPFvs OptRA vs FairILP [134]

(b) MovieLens: AlgRAPF vs RandAlgRAPF vs OptRA vs FairILP [134]

Figure 2.7 Kemeny Distance **RAPF**.

variants have been well studied, including rank aggregation considering different optimization functions, rank aggregation with partial ranking information, or with ties [8, 9, 24, 47, 88]. Kemeny optimal rank aggregation which minimizes the sum/average Kendall-Tau distances [127, 128] to the individually ranked lists is the most popular variant. In [9,24], 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 [9,186,187]. In [9], Ailon et al. introduced a randomized approximation algorithm with a $\frac{4}{3}$ approximation factor. In [186, 187], the authors propose deterministic pivoting algorithms with the same approximation factors. In [70] 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 [9, 82] 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 [82]. We note that finding an optimal rank aggregation using Spearman's Footrule based measure is computationally easy. However, it is *open*

Table 2.6 Approximation Factor of The Algorithms

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

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 [45] 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.

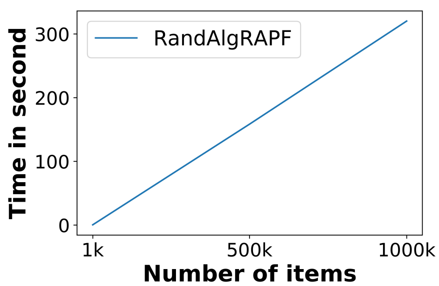
Proportionate Fairness. Based on the Chairman assignment problem [184], the idea of proportionate fairness (p-fairness) was studied in the context of resource scheduling [26]. The Chairman assignment problem simply studies how to select a chairman for a union from k states such that at any time the accumulated number of chairmen from each state is proportional to its weight. In [26], Baruha et al.

Table 2.7 Case Study Results on MovieLens Dataset

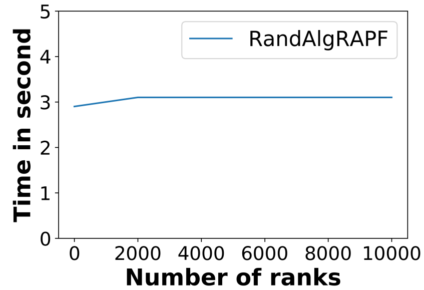
Movie	User1	User2	User3	User4	User5	OptRAPF	FairILP	Genre
Bad News Bears, The (1976)	9	7	7	7	4	7	3	Comedy
True Grit (2010)	7	5	1	9	3	9	6	Western
My Darling Clementine (1946)	2	3	3	3	10	4	4	Western
Last Picture Show, The (1971)	4	1	5	1	5	5	1	Drama
Man with the Golden Arm, The (1955)	6	8	4	10	6	8	10	Drama
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

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 [67] 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.*

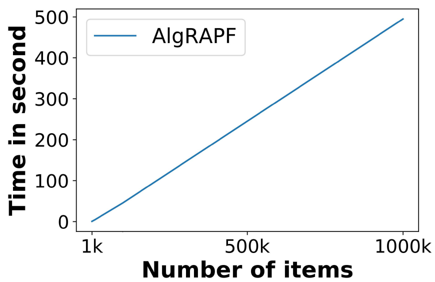
Social Choice Theory. Various ranking methods have been studied in the field of social choice theory [17, 93, 127, 148, 150, 210]. Early social choice theory literature considered rank aggregation in the context of preference aggregation methods [127, 150, 210]. The social choice theory papers [17, 93] 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 [148] 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.



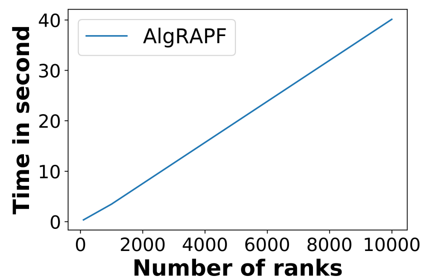
(a) Vary n, $m = 100$: RandAlgRAPF



(b) Vary m, $n = 1000$: RandAlgRAPF



(c) Vary n, $m = 100$: AlgRAPF



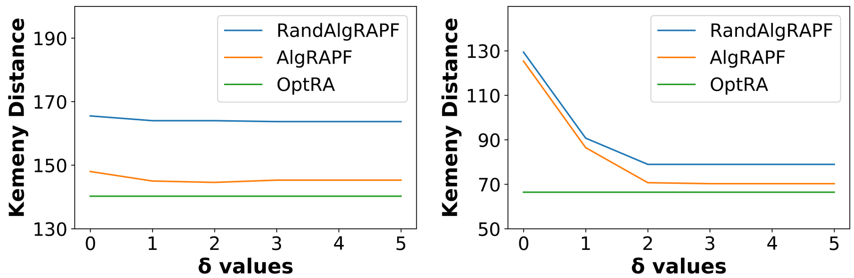
(d) Vary m, $n = 1000$: AlgRAPF

Figure 2.8 Running time analysis.

2.4.1 Fair ranking solutions

Several recent fair ranking studies focus on achieving fairness on a *single* rank [18,60, 104,214]. Celis et al. [60] 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 [214], 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 [104] that studies a variant of the **IPF** problem. The other



(a) Movie Lens : ALGRAPF vs RANDALGRAPF varying δ

(b) Fantasy Football : ALGRAPF vs RANDALGRAPF varying δ

Figure 2.9 Varying δ analysis RAPF.

one is FAIRILP [134], 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. [104] 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 2.4.1. Statement: DETCONSTSORT **[104] 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 2.1 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 2.4.2. Statement: DETCONSTSORT **[104] does not produce a p-fair ranking.** *The ranking produced by* DETCONSTSORT *in Example 2.4.1 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 [134] 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 2.4.3. Statement: FAIRILP **[134] is not necessarily p-fair even with** $\delta = 0$.

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

Table 2.4 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, 60, 104, 214], 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 *[134], as well as consider both binary and multi-valued protected attribute.*

2.5 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 two approximation factor. Next, we design two computational frameworks to solve **RAPF**: RandAlgRAPF and AlgRAPF that exhibit three and four 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 [8, 88]. We are also interested to study how to obtain an aggregate p-fair ranking when each member inputs only a partial ranking [8, 88].

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 [5]. For convex bipartite graphs, we currently explore if and how existing works that aim at finding a maximum matching without any crossing [64, 146] can adapt to crossing minimization of a perfect matching.

CHAPTER 3

SATISFYING COMPLEX TOP-K FAIRNESS CONSTRAINTS BY PREFERENCE SUBSTITUTIONS

3.1 Introduction

Preference aggregation is important in finding top-k outputs that represent plurality preference [151] and has wide variety of applications in recommender systems, search results listing [29], electoral systems [136, 152], or allocating resources among candidates, such as, in hiring or admission [218]. 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. 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. In voting theory [56], the concept of margin of victory (MOV) is designed to measure electoral competitiveness of the candidates, that we formalize as the smallest number of single ballot substitutions to promote a given set of k candidates as the top-k. To the best of our knowledge, we are one of the first to formalize the computational problem* - find margin via single ballot substitutions to promote a set of k candidates as top-k, considering multiple protected attributes of the candidates **(Section 3.6 contains details on related work)**.

Our first contribution is to formalize several variants of the *margin finding problem via single ballot (preference) substitutions* considering complex fairness constraints **(Section 3.2)**. (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 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.

Our second contribution is to study the defined problems theoretically and make principled algorithmic contributions **(Sections 3.3 and 3.4)**. 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. 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 [102]. 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 [102]. 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 3.1. Our final contribution is experimental **(Section 3.5)**. 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 [94,181] 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. We conclude in **Section 3.7**.

3.2 Data Model and Problem Definitions

In this section, we describe the data model and illustrate that with a running example, following which we define the studied problems.

3.2.1 A toy running example

Table 3.2 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 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.*

Table 3.1 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	$\mathcal{O}(n^{c+1})$

An Example Complex fairness constraint. Imagine the goal is to select $k = 4$ candidates from the voting outcome described in Table 3.2 with the following fairness constraints described in Table 3.3.

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].

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 3.2.

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

[1]For the remainder of the paper, users and voters are synonymous, as well as items and candidates are used interchangeably.

Table 3.2 Twelve Voters, Six 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

Table 3.3 Fairness Constraints in The 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 3.4 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 from group G_A with at least t votes
C, c	set of candidates, $\Pi_{i=1}^{\ell}\ell_i$

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 [115, 134, 181] Fairness Constraints. A fairness constraint defined over a single protected attribute containing ℓ different groups G_1, G_2,.,G_ℓ, requires that the representation of each group G_i is a_i in a fair top-k, where $\sum_i^\ell a_i = k$. 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 3.3. Based on this, $\{C1, C3, C5, C6\}$ is a feasible top-4 outcome, as it satisfies all these requirements.

3.2.2 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 1. 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 3.2, 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 2. 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 3.2 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 3. 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 3.2 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 assign 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 $|R| > 2$, this problem instantiates to finding the margin when the fairness constraints are defined over three or more different attributes.

Consider Table 3.2 again and the fairness constraint presented in Table 3.3. 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.

3.3 Single Protected Attribute

We study two margin finding problems via single ballot substitutions, namely MFBinaryS and MFMultiS, the first one considers fairness constraint defined over a single binary protected attribute, and the second one for a single multi-valued protected attribute.

3.3.1 Binary protected attribute

The inputs 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).

Our algorithms use the notion of *threshold* defined as follows.

Definition 8. *The* threshold t *of an election outcome is the number of votes, such that each of the top-k candidates have got at least t votes and at least one such candidate got exactly t votes.*

Using the 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.

Intuitively speaking, our algorithms are based on the following two observations. First, for any given election outcome and a given threshold t we can compute the minimum number of single ballot substitutions that guarantee a fair outcome with threshold t; that is, after performing these substitutions the top-k candidates will consist of a candidates from G_A and b candidates from G_B, all these candidates will get at least t votes, and at least one of these k candidates will get t votes. This is shown in FINDBALLOTSUBB. Second, any optimal algorithm can be viewed as an algorithm that searches for the optimal value of the threshold t, that is, the threshold t that guarantees a fair outcome with the minimum number of ballot substitutions. This is proven in Lemma 2. This implies that to find the optimal solution we need to find the optimal threshold t. Naively, this can be done by checking all possible values of t. To make the algorithms more efficient we prove several properties that enable us to perform a binary search for the threshold in a relatively small space.

The pseudo code of FindBallotSubB and Alg1AttBOpt is shown in Algotithms 1 and 2.

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 4 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

Algorithm 1 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: $B_a = t(b - b_*(t)) - \sum_{i \in I_a \text{ \& } L_C(i) \in G_B} L_V(i)$

4: $I_r = \{(i_a + 1), \ldots, (a_*(t) + b_*(t))\}$

5: $B_r = \sum_{i \in I_r \text{ \& } L_C(i) \in G_A} L_V(i) - (t-1)(a_*(t) - a)$

6: $S = \max\{B_a, B_r\}$

7: Return S

threshold is in the interval $[L_V(i), L_V(i+1))$. In our Technical Report we show how the optimal threshold in this interval can be computed in constant time. However, $a_*(t)$ and $b_*(t)$ are the same for every $t \in [L_V(i), L_V(i+1))$, and thus the optimal threshold in this interval can be computed in constant time.

3.3.2 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

Algorithm 2 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 $\min\{S_a, S_b, M\}$

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 $b_*(t) < b$. 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.

Handling ties. The optimal solution may have a tie 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. Similarly, 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 $t-1$.

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. 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 1. 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. Similar arguments could be made for the other cases or for a tie. $\square$

Lemma 2. *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 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 (similarly from G_B) who got at least $t+1$ votes is less than a (b for G_B) but the number of candidates from G_A (similarly from G_B) who got at least t votes is equal to or greater than a (b for G_B). Thus, any optimal algorithm is essentially finding a threshold t that requires minimum number of ballot substitutions. $\square$

Lemma 3. *For a threshold t, subroutine* FINDBALLOTSUBB *returns the minimum number of ballot substitutions and satisfies fairness.*

Proof. 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 the 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. Similar arguments can be made for the other 2 cases and for tie. $\square$

Lemma 4. *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 $[L_V(i_b), L_V(i_a)]$. $\square$

Lemma 5. *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 $[L_V(i_a), L_V(i_b)]$.

$\square$

Theorem 1. 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 the optimality holds.

□

3.3.3 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 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 2. AlG1AttMOpt *always produces a fair outcome.*

Proof. The proof is similar to the proof of Theorem 1. □

3.4 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.

We begin the section by presenting a generic solution framework AlgCartOpt that is exact and exponential in general. 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. 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.

3.4.1 Exact solution AlgCartOpt

We propose AlgCartOpt by first converting multiple protected attributes to a single multi-valued attribute by enumerating all possible configurations. This step is exponential in the general case. Then, for each such configuration, we check the feasibility of the solutions. If the solution is feasible, then, AlG1AttMOpt is called to produce the margin for that case. Finally, we return that feasible configuration that has the smallest margin.

Suppose that $\Pi_{i=1}^{\ell} \ell_i = c$. This means that we have total number of 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:

$$\forall j \in [1..\ell] \; \forall r \in [1..\ell_j] \quad \sum_{i=1}^{c} n_i \cdot \mathbf{1}_{V[i,j]=A[j,r]} = a[j,r]. \tag{3.1}$$

After that, we call ALG1ATTMOPT that produces 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. Finally, we return that instance which has the smallest margin.

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 c attribute configurations by $\mathcal{O}(n^c)$ calls to the single attribute case and then choosing the call that produces the smallest margin. ALG1ATTMOPT has a running time of $\mathcal{O}(n)$. Overall running time is $\mathcal{O}(n^{c+1})$. Clearly, when c is a constant, ALGCARTOPT takes polynomial time to run.

Theorem 3. AlgCartOpt *finds the optimal set of single ballot substitutions.*

Proof. 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 (3.1). 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. □

3.4.2 MFMulti3+- 3 attributes 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 4. *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. In a nutshell, given a 3DM problem instance with vertex set $X_1 \cup X_2 \cup X_3$, each vertex in X_i corresponds to a distinct value of the i-th attribute. Each hyperedge $(x_{1,a}, x_{2,b}, x_{3,c})$ corresponds to a candidate with the attributes $A[1, a]$, $A[2, b]$, $A[3, c]$. An outcome with exactly one candidate for each attribute value is feasible iff the 3DM instance has a 3 dimensional matching. Our Technical Report contains further details. □

3.4.3 MFMulti2- 2 attributes 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 5. MFMULTI2 *is weakly NP-hard.*

Proof. To prove the hardness, we reduce the weakly NP-Hard Partition problem to our problem. The reduction is based on the fact that any solution with a ballot additions and r ballot removals implies a solution with $\max\{a, r\}$ ballot substitutions. For a given Partition problem instance we build an instance of the 2 attribute case in which the total number of ballot additions and subtractions is at least the sum of the n input integers in the Partition instance. The 2 attribute case instance has a solution with an equal number of ballot additions and removals each of which equals half of the sum of the n input integers iff a partition of the n integers exists. We refer to our Technical Report for details. □

3.4.4 Approximation algorithm for MFMULTI2

We show a 2 approximation algorithm for computing the margin in the 2 attribute case. For this we first show how to compute the minimum number of vote additions and removals that guarantee a fair outcome in the 2 attribute case.

Computing the Minimum Number of Ballot Additions and Removals We compute the minimum number ballot additions and removals that yield a fair outcome by enumerating all possible thresholds and for each threshold t calling the subroutine FINDBALLOTA+R that is shown in Algorithm 3. Subroutine FINDBALLOTA+R computes the minimum number of ballot additions and removals that yield a fair outcome with threshold t by casting the problem as a min-cost b matching problem.

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 $a[1,i]$ of them are adjacent to node $x_i \in X$, for $i \in [1..\ell_1]$, and exactly $a[2,j]$ of them 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. Let $M^* \subseteq E$ be a minimum cost matching.

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 $= 2+2 = 2+1+1 = 4 = k$. A minimum cost 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 $AplusR = -1+7 = 6$.

Theorem 6. *The number of ballot additions and removals needed to guarantee the election of the candidates corresponding to the edges of M^* with threshold t is minimum among all fair outcomes obtained with threshold t.*

Proof. Let $R = \sum_{e \in E} \max\{-w(e), 0\}$. By our definition of the b-matching there is one to one correspondence between the set of b-matchings and the set of fair outcomes. Consider a matching M. We claim that $w(M) + R$ is the number of ballot additions and removals needed to guarantee the election of the candidates corresponding to the edges of M with threshold t. To see this we consider the contribution of each edge to 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.

Summing over all edges yields our claim. 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 the minimum number of ballot additions and removals that guarantee a fair outcome we need to iterate the min cost matching over all possible threshold values. We show that it is enough to consider no more than $3n-2$ 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 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 the $3n-2$ threshold values $L_V(i), L_V(i)+1, L_V(i+1)-1$, for $i \in [1..n-1]$, and $L_V(n)$.

Approximating the Number of Single Ballot Substitutions Suppose that we are given a ballot additions and r ballot removals that guarantee a fair outcome. We show how to transform them to at most $a+r$ ballot substitutions that guarantee the same outcome. 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 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 it must be the case that less than k candidates received even one vote. Since we need to elect k candidates it is reasonable to assume at least k candidates received at least one vote.

Let O_{A+R} be the optimal number of ballot additions and removals that yield a fair outcome. It follows that we can find a set of at most O_{A+R} ballot substitutions that yield a fair outcome as shown in algorithm ALG2ATTAPX (Algorithm 4). The approximation ratio is proved in the following theorem.

Theorem 7. *The size of the set of ballot substitutions output by* ALG2ATTAPX *is at most twice the minimum number of ballot substitutions that yield a fair outcome.*

Proof. A ballot substitution can be viewed as a single ballot addition and a single ballot removal. Thus, any solution with $x \geq 0$ ballot substitutions can be converted to a solution with $2x$ ballot additions and removals. Let OPT_C be the minimum number of ballot substitutions that yield a fair outcome. It follows that there are $2OPT_C$ ballot additions and removals that yield a fair outcome. Hence, $OPT_{A+R} \leq 2OPT_C$, and the solution with at most OPT_{A+R} ballot substitutions output by ALG2ATTAPX is a 2 approximation. □

Running Time. The running time of ALG2ATTAPX is determined by the time complexity of subroutine FINDBALLOTA+R 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 [7]. The subroutine FINDBALLOTA+R is called $\mathcal{O}(n)$ times from ALG2ATTAPX. Thus, the running time of ALG2ATTAPX is $\mathcal{O}(n^2\ell \log m)$.

3.5 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+. The scalability measures the running time of the implemented algorithms by varying appropriate parameters.

3.5.1 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 real world datasets are described in Table 3.5. MovieLens3Star (similarly MovieLens5Star) datasets are created from the Movielens dataset by converting all user ratings of 3 or more (similarly 5) ratings as a vote, and selecting the movies accordingly.

Table 3.5 Real World Datasets

Dataset	# candidates(n)	# voters(m)	protected attributes (ℓ)
New South Wales (NSW) Senate Elections	105	$4,695,326$	2 attributes on the political parties and the election history
Bronx Justice of the Supreme Court Election in New York City	6	$343,071$	single binary - democrat and republican
MovieLens5Star	$2,926$	$382,323$	3 attributes on movie genre, production company and original language.
MovieLens3StarMore	$17,619$	$1,613,420$	3 attributes on movie genre, production company and original language.

Synthetic dataset. We generate large scale synthetic data for m voters and n candidates using normal distribution as voting outcomes tend to follow such distributions [154]. The process runs as follows: a loop is repeated m times to generate an id in the range $[0..n-1]$ (top candidate choice of a voter), by sampling an integer using the normal distribution with certain mean (*mean*) and standard deviation (*sd*), and then taking this integer modulo n to ensure that the id is in range. The *mean* and *sd* are integers chosen uniformly in the range $[0..n-1]$. Additionally, the protected attributes of the candidates are sampled uniformly within their range.

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.
(1) LEXIMIN [94] + Alg1AttMOpt . 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 Alg1AttMOpt to compute the margin.
(2) Fair-Topk-Set [181] + Alg1AttMOpt . This related work also does not solve the margin finding problem. The best use of this algorithm is to study it in the context of multiple protected attributes, where this algorithm first converts multiple

protected attributes to a single multi-valued protected attribute by computing joint distribution over the attributes assuming their independence. Given the resultant proportion, we run the ALG1ATTMOPT to compute the margin. FAIR-TOPK-SET is a heuristic, may not produce the smallest margin, or even a feasible solution, as we demonstrate empirically.

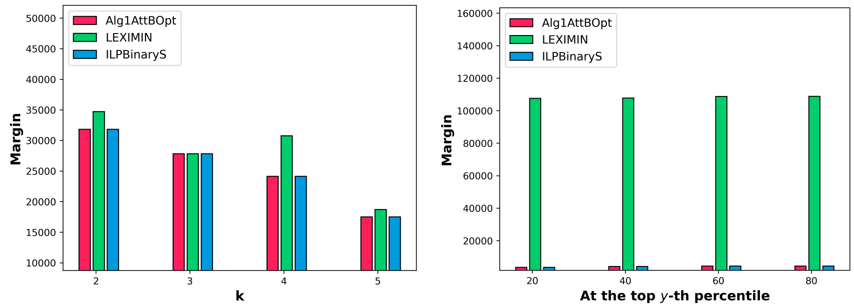
(3) **Integer Linear Programming.** We implement an exact algorithm for MFBINARYS, MFMULTIS, MFMULTI2, and MFMULTI3+ problems using ILP. We refer to these variants as ILPBINARYS, ILPMULTIS, ILPMULTITWO, ILPMULTITHREE, respectively.

3.5.2 Quality experiments results

Results for MFBINARYS Figure 3.1a shows the results from the election for 2021 Bronx Justice of the Supreme Court. There are six candidates (five Democrats and one 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.

In Figure 3.1b, we evaluate the effect of "vote gap" between the candidates to decide the margin. Here the x axis shows a particular candidate who is at is at the top y-th percentile (calculated as $y\% \times n$) after the initial vote outcome, and needs to be promoted in the final top-k ($k = 5$) to ensure fairness. The y-axis shows the margin value. Note that a large value of top y-th percentile produces higher vote gap between the current top-k and the candidate at the top y-th percentile that needs to be promoted to top-k.

Results for MFMULTIS We present the results of MFMULTIS in Figure 3.2. We consider political parties (36 different parties) of the candidates in 2019 NSW Senate Election, and the movie genres of the Movielens datasets (18 different genres)



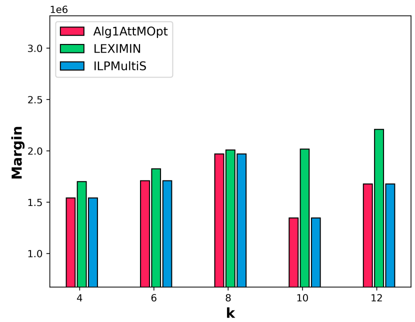
(a) 2021 Bronx Justice Election **(b)** 2019 NSW Senate Election

Figure 3.1 Results for MFBinaryS.

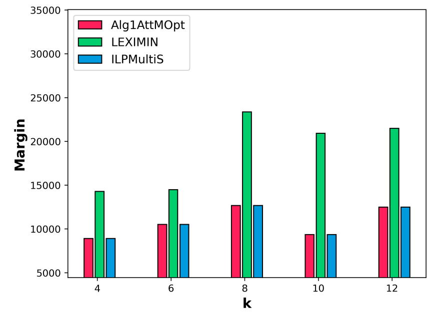
as the protected attribute. These results demonstrate similar pattern as that of MFBinarySresults.

Results for MFMulti2 Figure 3.3 shows the results for MFMulti2. For the NSW Senate Election dataset, the two protected attributes are: political party of the candidates and whether the candidate has been elected before or not. For the Movielens datasets, we use two protected attributes: genres (18 unique values), and language (English or not). The results show that Alg2AttApx has significantly lower margin compared to Fair-Topk-Set and LEXIMIN, and the margins produced by Alg2AttApx are bounded by 2 times the margins produced by ILPMultiTwo and AlgCartOpt that produce identical results. In fact, in many cases, Fair-Topk-Set produces infeasible results.

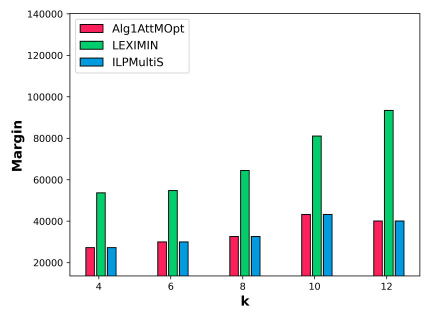
Results for MFMulti3+ This is run on the MovieLens datasets only. In addition of the two protected attributes genre and language, we consider the production company (American or not) as the third protected attribute. The results are presented in Figure 3.4, which is consistent with our previous observations.



(a) 2019 NSW Senate Election



(b) MovieLens5Star



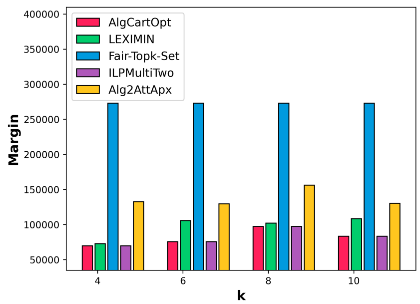
(c) MovieLens3StarMore

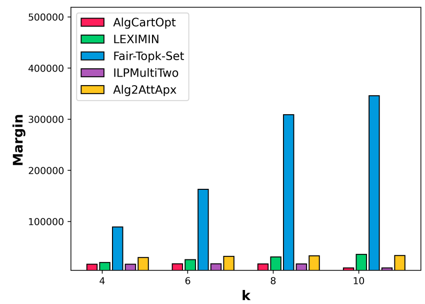
Figure 3.2 Results for MFMultiS.

3.5.3 Scalability results

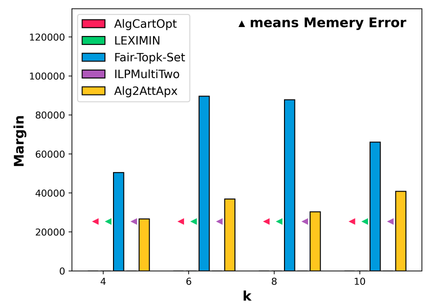
For these experiments, we use the synthetically generated normally distributed data to validate the effect of the parameters n, m, k on the running time of the proposed algorithms, considering 1, 2, or 3 and more protected attributes.

Results for MFBinaryS**,** MFMultiS Figure 3.5 shows that our proposed algorithms Alg1AttBOpt and Alg1AttMOpt are scalable up to millions of candidates and voters. In Figure 3.5a, 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 (Figures 3.5b, 3.5c). These results are consistent with our theoretical analysis. Fair-Topk-Set is the best choice scalability-wise (but fails to optimize margin or even gives infeasible results), whereas, LEXIMIN does not scale because of its exponential nature.





(a) 2019 NSW Senate Election **(b)** MovieLens5Star



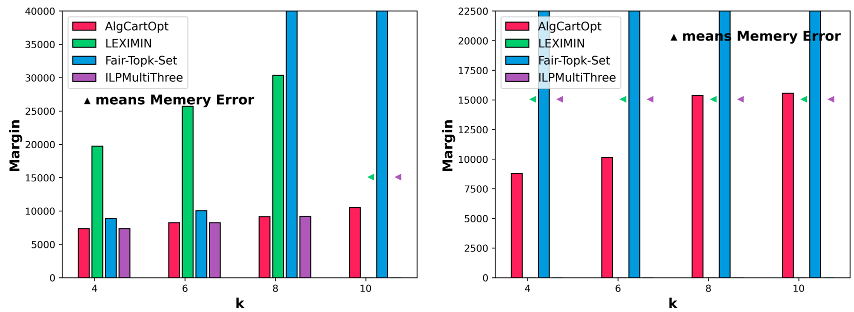
(c) MovieLens3StarMore

Figure 3.3 Results for MFMULTI2.

Figures 3.7a and 3.7b show running times varying standard deviation sd and $mean$. We observe non-significant change in running time due to varying $mean$ and sd.

Results for MFMULTI2 The running time of ALG2ATTAPX w.r.t n, m, k are shown in Figure 3.6. The running time of ALG2ATTAPX is sub-quadratic w.r.t number of candidates n as shown in Figure 3.6a, while it increases linearly w.r.t number of voters m (shown in Figure 3.6b) and does not change significantly with the size of the result set k (Figure 3.6c). As before, FAIR-TOPK-SET is the fastest, while LEXIMIN does not scale.

Results for MFMULTI3+ We present ALGCARTOPT as the solution of MFMULTI3+. It produces $c = \Pi_{i=1}^{\ell} \ell_i$ as the number of attribute value configurations.



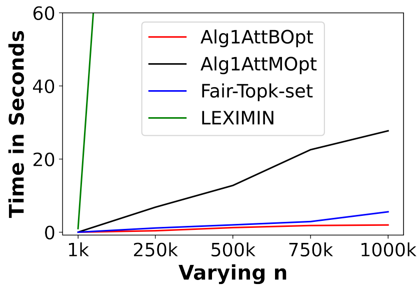
(a) Movielens5Star (b) Movielens3StarMore

Figure 3.4 Results for MFMULTI3+ .

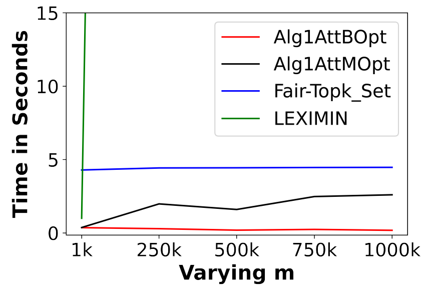
In Figures 3.8a and 3.8b, we see that the running time of ALGCARTOPT increases exponentially with n and c, as expected, whereas FAIR-TOPK-SET, LEXIMIN run faster.

3.5.4 Summary of results

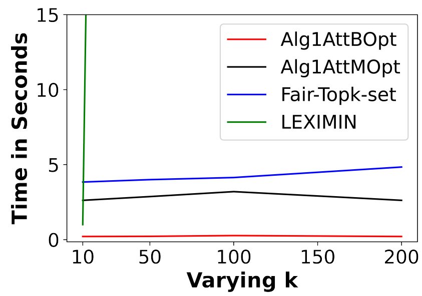
Our first and foremost observation is, consistent with our theoretical analysis ALG1ATTBOPT, ALG1ATTMOPT, ALGCARTOPT produce exact solutions of the underlying problems, i.e., they satisfy the fairness constraints, while minimizing the margin values, whereas, ALG2ATTAPX demonstrates better approximation factors compared to the theoretical bound 2. Our second observation is that the implemented state-of-the-art solutions LEXIMIN [94] and FAIR-TOPK-SET [181], despite adapting them non-trivially to our problem, fail to optimize margin values and do not turn out to be effective.As expected, FAIR-TOPK-SET [181] is highly scalable but produces sub-optimal margin values or even infeasible results for MFMULTI2, and MFMULTI3+, as it just becomes a heuristic for those problems. 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 observe that the value of margin depends on both k and the vote gaps between candidates: as k increases, the margin generally decreases, while the margin increases with larger vote gaps. The ILP based baseline



(a) Vary n, $k = 50, m = 10000k$



(b) Vary m, $k = 50$, $n = 100k$



(c) Vary k, $n = 100k, m = 10000k$

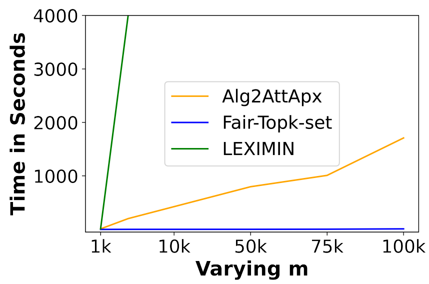
Figure 3.5 Running time for ALG1ATTBOPT & ALG1ATTMOPT.

solutions as well as LEXIMIN [94] give memory error when run on very large dataset. Overall, our designed solutions turn out to be the unanimous choice.

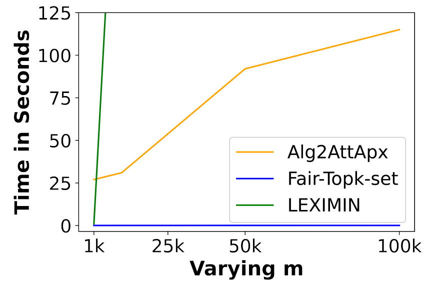
3.6 Related Work

We primarily discuss three types of existing work that are related to our proposed problem.

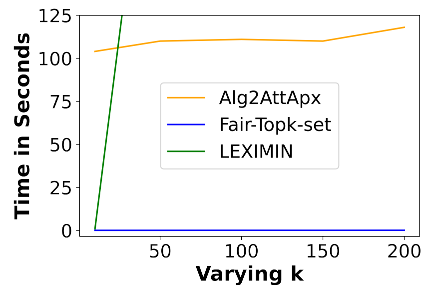
Preference Elicitation and Aggregation. Preference of the individual users could be elicited as pairwise comparison [74], in form of a binary vector [173] known as Approval Voting [46], in an ordinal scale [13, 132], or considering Arrowian social choice, where users provide partial or complete preference order over the items [40, 52, 130, 177]. Similarly, The properties of social welfare functions for aggregating preferences have been studied by mathematicians since the 18th century [51, 63, 66]. Different preference aggregation methods are proposed, including majority voting, plurality voting [136, 152, 161], their weighted versions, as well as



(a) Vary n, $m = 1000k, k = 50$

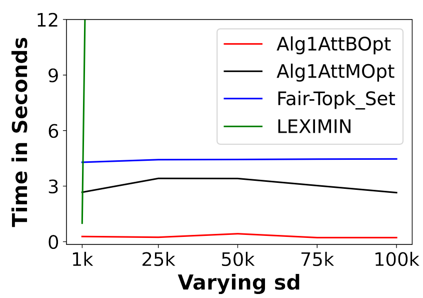


(b) Vary m, $n = 10k, k = 50$

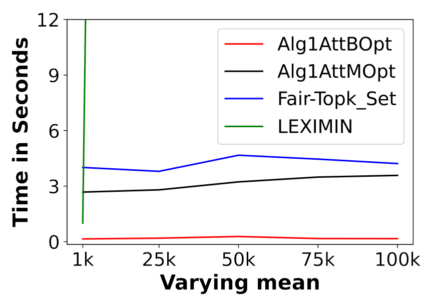


(c) Vary k, $n = 10k$, $m = 100k$

Figure 3.6 Running time for ALG2ATTAPX.



(a) Vary sd, $m = 1000k, n = 100k, mean = 50k$

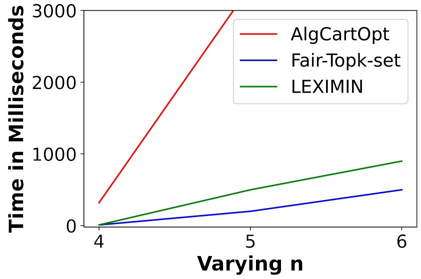


(b) Vary $mean$, $m = 1000k, n = 100k, sd = 30k$

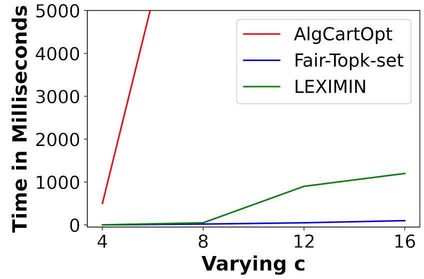
Figure 3.7 Varying distribution ALG1ATTBOPT and ALG1ATTMOPT.

aggregation methods that consider positional preference [40,52,177], such as Kemeny rule [82, 127], Condorcet rule [69], or Borda Count [84]. 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 [96,171], authors study the fairness of preference aggregation in the context of Arrow's Impossibility Theorem. In a very recent work of ours [197], we study how to ensure proportionate fairness [26,27] in aggregating preferences that are provided as ranked orders. Earlier, existing works



(a) Vary n, $c = 8, m = 1000$



(b) Vary c, $n = 5, m = 1000$

Figure 3.8 Running time for MFMULTI3+.

study proportionate representation considering group fairness in the top-k ranked order [104, 134]. Authors in [101] 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 [94, 181] to maximize fairness or diversity. In a recent paper [59], the authors maximize a monotone submodular function given only upper bound of fairness constraints. Unlike our work, it does not study ballot substitution as well as does not consider exact fairness constraints. Kearns, Michael, et al. prove that achieving subgroup level fairness is np-hard [126]. They propose an approximate solution (FairFictPlay) for achieving subgroup level fairness based on two-player zero-sum game between a Learner (finds best classifier) and an Auditor (finds best subgroup distribution).

Preference Substitution. Preference substitution could done 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 has received significant interests in electoral systems, in particular, to understand the mechanism of Single Transferable Vote (STV) [91, 183]. In [202], the authors study *margin*, to introduce a different election result for different voting systems, including approval voting, all positional scoring rules (which include Borda, plurality, and veto). In [41, 56, 176] 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.

While we adopt popular preference aggregation models and group fairness definitions, we are the first to formally study the margin finding problem under single ballot substitutions considering complex fairness constraints and present principled solutions.

3.7 Conclusion

We initiate the study of *the margin finding problem of a top-k preference aggregation model 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 to demonstrate the effectiveness of our proposed solutions.

Algorithm 3 FINDBALLOTA+R

Inputs: L_C, L_V, A, t

Outputs: The minimum number of ballot additions and removals required to yield a fair outcome with threshold t, and the set of elected candidates in the resulting fair outcome

$X = \{x_i : x_i \in A_1\}$

$Y = \{y_j : y_j \in A_2\}$

$E = \{e_c = (x_i, y_j) : c \in L_C \text{ with attributes } A[1,i]\ A[2,j].\}$

for $i = 1$ to n **do**

  $c = L_C(i)$

  if $L_V(i) < t$ **then**

    $w(e_c) = t - L_V(i)$

  else

    $w(e_c) = (t-1) - L_V(i)$

  end if

end for

Construct the graph G $= (X, Y, E, w)$

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

Find M^* a min cost b-matching in G subject to the constraints on the number of adjacent edges

$R = \sum_{e \in E} \max\{-w(e), 0\}$

$AplusR = w(M^*) + R.$

$C_t^* =$ the set of candidates corresponding to the edges in M^*

Return $(AplusR, C_t^*)$

Algorithm 4 Alg2AttApx

Inputs: L_C,L_V,A

Output: A set of at most OPT_{A+R} ballot substitutions that yield a fair outcome

```
OPT_{A+R} = k · m
U = {L_V(i), L_V(i) + 1, L_V(i + 1) − 1 : i ∈ [1 . . . n − 1]} ∪ {L_V(n)}
for t ∈ U do
    (AplusR, C*_t) = FindBallotA+R( L_C, L_V, A, t)
    if OPT_{A+R} > AplusR then
        OPT_{A+R} = AplusR
        C* = C*_t
    end if
end for
Transform the OPT_{A+R} ballot additions and removals needed to guarantee the
election of the candidates in C* to at most OPT_{A+R} ballot substitutions that
guarantee the same outcome
```

CHAPTER 4

SELECT-K WINNERS BY SATISFYING QUERY CONSTRAINTS USING IRV

4.1 Introduction

The task of finding the winner, i.e., the most favorable item or candidate from a given a set of m users' (voters') preferences over n items (candidates), has found a wide variety of applications in the search results listing, recommender systems, and even in electoral voting systems. Compelling examples include, the problem of finding the top choice movie, tweet, song, or news article, or even the winning candidate in electoral voting systems. IRV (Instant Run-off Voting) is a ranked choice voting mechanism, and has been gaining popularity lately as an electoral system in the US [42, 56, 116, 142, 176]. In this paper, we study the applicability and computational implications of adapting IRV in recommender systems and demonstrate its advantages over existing positional [172] preference aggregation mechanism, such as plurality voting. We study a settings where a query comes with multiple constraints on the permissible set of the winning candidates.

Example 4.1.1. *(***Running example.)**

Underlying preference database. *Table 4.2 represents ranked order of up to top-*5 *preferences over* 7 *movies (items or candidates in general, where each item is represented by a unique id) provided by* 10 *users (voters). Each of these ranked orders of preferences constitutes a ballot.*

Query and Constraints. *An example query comes with* $k = 3$ *constraints (Table 4.1), one constraint per day, with the requirement of selecting a thriller movie on Monday, a sci-fi movie on Tuesday, and a horror movie on Wednesday.*

Goal. *The goal is to return 1 movie each day that satisfies the constraint and is most representative of the users' preferences (in Table 4.2). The process has to recommend one of the two thrillers:* Inception *and* Fargo *for Monday.*

Preference aggregation. *We propose to repeatedly invoke IRV to aggregate the ballots and select the recommendation for each day.*

The IRV process. The IRV process [113, 149] is multi-stage process [172] that simulates $n - 1$ run-off rounds, where in each such round one item is eliminated. The single item that survived the eliminations after all rounds is the winner. More specifically, given the original preferences of the users (voters), an initial tally of the first choice votes of every candidate is performed in round 1. The item that has the lowest number of first choice votes is eliminated. Ties are broken arbitrarily. After the elimination all the ranked orders that include the eliminated item are updated, and the items following this eliminated item in the ranked order are advanced one place up. This concludes round 1. This is iterated $n-1$ times, namely, the tally is recomputed, and the item that has the lowest number of first choice votes is eliminated, where the ties are broken arbitrarily.

Using the running example, the movie *Scream* has the highest number (3) of first choice votes at the initial tally. Then, as shown in Table 4.3, the IRV process eliminates *Fargo* in round 1, *American Psycho* in round 2 (and the respective vote gets transferred to *The Last Jedi*), and *Return of Jedi* in round 3. This process continues further making *The Last Jedi* the winner after 6 rounds.

Motivation for using IRV in recommendation. The resurgence of IRV is motivated by a range of expected benefits, including, ensuring majority support for the winner, reducing conflict within the electorate, reducing strategic voting, and increasing diversity of the winners [149]. IRV is amenable to incomplete ranked order, making the process further suitable to applications, where users are not obligated to provide full order. A recent work [62] suggests IRV as a way to aggregate preferences in recommendation applications, demonstrated the superiority of IRV over plurality voting according to multiple fairness criteria, namely proportional representation of solid coalition and anti-plurality, as we shall demonstrate in Subsection 4.2.2. Referring to Table 4.2, note that plurality voting will choose *Inception* as the winner among the movies in the Thriller genre, even though it is clear that between *Inception* and *Fargo*, the latter is more preferred by the users. As we will demonstrate later our IRV based process will indeed choose *Fargo*. Finally, it is known that finding the *margin* (the number of ballots that must be substituted in order to change the original winner [76,121,176,202]) for IRV is NP-hard [42], making IRV less susceptible to manipulation.

Margin computation using IRV (Section 4.2). Recall that the IRV process chooses *The Last Jedi* as the winner of the ballots given in Table 4.2. Clearly, *The Last Jedi* is not a Thriller, i.e., it does not satisfy the query constraint for Monday. Hence, some ballot modifications are needed. Table 4.4 shows one such modification, where user Jack's ballot is changed by replacing *Scream* with *Fargo*. After this, a series of 6 run-off rounds are simulated, as listed in Table 4.5, which makes *Fargo* the winner. If instead *Inception* is to be made the winner, this will require at least 3 ballot modifications, for example, by replacing the top choice of Alice, Monica, and John with *Inception*. Intuitively too, *Fargo* is a better choice because it is liked *as the second choice for* 3 *out of* 7 *original users*. A similar process must also be carried out for Tuesday and Wednesday independently. Our goal is to study this problem - how to modify the original ballots of IRV to satisfy all k query constraints such that the total number of required ballot changes is minimized. Motivated by a recent work studying this problem for plurality voting [121], we refer to this problem as a margin computation to satisfy k query constraints using IRV (or **MqKIRV** in short). To the best of our knowledge, we are the first to initiate a principled study on this topic.

Table 4.1 Query Constraints

Day	**Genre**	**Movies**
Monday	Thriller	Fargo, Inception
Tuesday	Sci-Fi	Return of Jedi, Star Wars, The Last Jedi
Wednesday	Horror	Scream, American Psycho

Table 4.2 Preferences Over 7(n) Movies by 10 Users(m) Upto 5-th Position (ℓ)

User	**1st choice**	**2nd choice**	**3rd choice**	**4th choice**	**5th choice**
Jack	Scream	American Psycho			
Emma	Inception	Return of Jedi	Fargo	Star Wars	Scream
Monica	Scream	Fargo	Star Wars	Return of Jedi	The Last Jedi
Daniel	Scream	Fargo	The Last Jedi	Return of Jedi	
Max	American Psycho	Fargo	The Last Jedi	Star Wars	Scream
John	The Last Jedi	Return of Jedi	Star Wars	Scream	
Amy	Return of Jedi	The Last Jedi	Star Wars	American Psycho	Scream
Alice	The Last Jedi	Return of Jedi	Star War	Fargo	Scream
Bob	Star Wars	Return of Jedi	The Last Jedi	Fargo	Scream
Steve	Star Wars	Return of Jedi	The Last Jedi		

Table 4.3 IRV Rounds: *The Last Jedi* Winner

MovieId	**Movie Name**	**Initial tally**	**End of round 1**	**End of round 2**	**End of round 3**	**End of round 4**	**End of round 5**	**End of round 6**
0	**Scream**	3	3	3	3	3	3	
1	**The Last Jedi**	2	2	3	4	5	7	9 (winner)
2	**Star Wars**	2	2	2	2	2		
3	**Inception**	1	1	1	1			
4	**Return of Jedi**	1	1	1				
5	**American Psycho**	1	1					
6	**Fargo**	0						

Table 4.4 A single Ballot Modification for Monday

User	1st choice	2nd choice	3rd choice	4th choice	5th choice
Jack	**Fargo**	American Psycho			
Emma	Inception	Return of Jedi	Fargo	Star Wars	Scream
Monica	Scream	Fargo	Star Wars	Return of Jedi	The Last Jedi
Daniel	Scream	Fargo	The Last Jedi	Return of Jedi	
Max	American Psycho	Fargo	The Last Jedi	Star Wars	Scream
John	The Last Jedi	Return of Jedi	Star Wars	Scream	
Amy	Return of Jedi	The Last Jedi	Star Wars	American Psycho	Scream
Alice	The Last Jedi	Return of Jedi	Star War	Fargo	Scream
Bob	Star Wars	Return of Jedi	The Last Jedi	Fargo	Scream
Steve	Star Wars	Return of Jedi	The Last Jedi		

Table 4.5 IRV Rounds After Ballot Modification:*Fargo* Winner

MovieId	Movie	Initial tally	End of round 1	End of round 2	End of round 3	End of round 4	End of round 5	End of round 6
0	**Scream**	2	2	2	2			
1	**The Last Jedi**	2	2	3	3	3	5	
2	**Star Wars**	2	2	2	2	2		
3	**Inception**	1	1	1				
4	**Return of Jedi**	1	1					
5	**American Psycho**	1						
6	**Fargo**	1	2	2	3	5	5	7 (winner)

Technical Contributions (Sections 4.3 and 4.4). We make multiple technical contributions in terms of analyzing the studied problems as well as designing solutions for them. We prove that **MqIRV** is NP-hard, even when the ballot size is at most $\ell = 2$ by reducing an instance of the known NP-complete problem Exact Cover by 3-Sets (X3C) to an instance of **MqIRV**. Inspired by [42, 142] we then design an algorithmic framework AlgExact that needs to consider all possible permutations over the candidates (i.e., $n!$), where each permutation represents an elimination order simulating multiple run-off rounds of IRV, such that the last candidate in the order is the winner. Solving AlgExact thus requires repeatedly solving a sub-problem DistTo, which, given a permutation, finds the smallest number of ballot modifications to satisfy that order. Unfortunately, we prove that even the decision version of DistTo is NP-hard by reducing an instance of X3C to DistTo, even when $\ell = 3$. The basic idea behind AlgExact is to repeatedly invoke DistTo for every possible permutation and retain the permutation that requires the smallest number of ballot modifications overall as the answer. We further study efficiency opportunities of AlgExact by enabling early terminations - basically, the idea is to avoid making expensive DistTo calls, rather compute a lower bound of margin for every possible suffix over all permutations, and eliminate a permutation in its entirety if its lower bound of margin is not smaller than the current upper bound of margin over the instance of **MqIRV**. To that end, we design a highly effective lower bound computation algorithm DistToLB and an upper bound computation algorithm **MqIRVUB** that are highly effective and computationally lightweight. We also study the DistTo problem under different preference manipulation model - for example, we study *how to only add the smallest number of ballots* to the existing set of ballots, such that the query constraints are satisfied. We refer to this as the DistToAdd problem. Clearly, this model varies the number of ballots, unlike the modification model that we closely study in this work, which keeps this number as an invariant. However, some applications may be flexible enough not to require this invariance assumption. We present an efficient exact solution DistToAddAlg for the DistToAdd problem. Moreover, we present an integer programming formulation for **MqIRV** that is non-trivial. We finally design a highly scalable heuristics AlgApprx that works well in practice.

Experimental Evaluations (Section 4.5). Our final contribution is experimental - we use four real world large scale datasets motivated by different recommender systems and electoral voting applications, as well as one synthetically generated very large datasets. Our experimental evaluations have the following findings: **(a)** we empirically show that **MqIRV** results in significantly smaller anti-plurality index (meaning does not select candidates that are disliked by the majority of voters), compared to alternative approaches, such as plurality voting based margin computation. **(b)** Our results demonstrate that AlgExact is optimal, yet more scalable compared to state-of-the-art solutions [142], [42] that we implement after appropriate adaptation. **(c)** We empirically demonstrate the optimality of DistToAddAlg and its scalability, as well as the effectiveness of our designed approximate solution AlgApprx qualitatively and scalability wise by varying several pertinent parameters and comparing with appropriate additional baseline algorithms. We present the discussion of related work in Section 4.6 and we conclude in Section 4.7.

4.2 Data Model and Problem

In this section, we describe the data model, following which we formally define the problem, and prove its hardness.

4.2.1 Data model

Ballot/preference. Preference of a user is elicited using a ballot b containing a ranking up to position at most ℓ, where c_i is the i-th preferred candidate. Using the running example, c_1 and c_5 are *Return Of Jedi*, and *Scream*, respectively of user *Amy's* ballot.

Database of ballot profiles. The database contains the preferences/ballots $\mathcal{B}$ of m users/voters over a set C of n items/candidates. Using the running example, $m = 10$, $n = 7$. The columns in Table 4.2 show $\mathcal{B}$.

Preference aggregation satisfying query constraints. A preference aggregation method $\mathcal{F}$ takes $\mathcal{B}$ as input and selects a winner from the candidates/items. A query Q comes with k constraints, and the goal is to make use of $\mathcal{B}$ and $\mathcal{F}$ multiple (k) times to select k different winners. Table 4.1 shows $k = 3$ such constraints for recommending a thriller, a sci-fi, and a horror movie on Monday, Tuesday, and Wednesday, respectively. We use IRV as $\mathcal{F}$, as discussed in more detail in Subsection 4.2.2.

Preference manipulation models. We consider two different preference manipulation models, where only the first one satisfies the number of ballot invariance property and is our primary focus in this work.

1. By modification. Given a ballot b with ranking up to position j ($j \leq \ell$) positions, update any number of entries in b considering candidates from C. As an example, in Table 4.4 Jack's ballot is changed from *Scream, American Psycho*→*Fargo, American Psycho*. Contrarily, if *Scream, Fargo, Return Of Jedi, The Last Jedi*→*American Psycho, Star Wars*, that also constitutes to a single ballot modification.

2. By addition. Add a new ballot b with ranking up to position at most ℓ with candidates from C.

4.2.2 IRV properties

IRV is known to satisfy properties [62] that other preference aggregation measures are unable to accommodate. We briefly discuss why these properties may be important in the recommendation application settings that we study.

IRV promotes proportionality for solid coalitions. Consider a scenario in which two candidates with similar ideologies compete over the same pool of voters, resulting in divided votes and potentially allowing a third candidate with a different ideology that has fewer overall votes to win. In social choice theory, a solid coalition for a set of candidates is defined as a set of voters who all rank every candidate in that set higher than any candidates outside that set. This criterion requires that if the number of such voters is at least half of the total number of voters, then one of that candidates from that set must win. IRV fulfills this criterion, whereas plurality voting [108] fails to do so. To demonstrate this property, notice that in our running example, there exists a solid coalition of voters who like sci-fi (refer to Table 4.2 which shows 5 of the 10 users, John, Amy, Alice, Bob, and Steve rank the three sci-fi movies *Return Of Jedi, The Last Jedi, Star Wars* higher than any other movie). Clearly, if user preferences are aggregated using plurality voting, none of the sci-fi movies will be returned as the winner since *Scream* has the highest number of first place votes, and will be selected as the winner. On the contrary, IRV will select *The Last Jedi* as the winner, and hence it is resistant to the ballot splitting problem.

IRV promotes anti-plurality. In social choice theory, the majority loser criterion was proposed to evaluate single-winner elections. It states that if a majority of voters prefer every other candidate over a given candidate, then that candidate must not win. IRV fulfills this criterion [62] (as there is a solid coalition for the rest of the candidates). Indeed, the movie *Scream* is the last choice of 6 out of the 10 users (Table 4.2), and thus IRV will not select it. Contrarily, plurality voting will select *Scream* as the winner. In [62] this criterion is extended to require that no candidate among the bottom $x\%$ of the ranked choices for the majority of the voters should be selected. While this is not guaranteed by IRV, it is empirically shown in [62] that IRV fulfills this extended criterion frequently.

Handling ties in IRV Recall that according to our definition ties during the IRV process are broken arbitrarily. It is not difficult to see that the way these ties are broken may impact the value of the margin. Indeed, in our example of ballot modification movie *Fargo* is the winner after just a single modification only in case the ties are broken in a very specific way. We made the design decision to compute the margin assuming the "best case" of tie breaking, namely, compute the margin under the assumption that ties are broken in an optimal way to ensure a low margin. Certainly, we could also choose the "worst case" of tie breaking. We postulate that any consistent choice would be effective in our case since we use the margin to distinguish among choices and are not interested in the actual value of the margin. (This is in contrast to worst case complexity analysis where the value of the complexity measure is important.)

4.2.3 Problem definitions and hardness

Problem Definition 1. MqIRV *(IRV Margin satisfying a single query constraint). Given a set of ballots $\mathcal{B}$ eliciting m voters ranked preferences of up to ℓ positions over a given set C of n candidates, and a query constraint that specifies a subset of the candidates, find the minimum number of ballots that need to be modified in order to ensure that the winner of the IRV election belongs to the subset specified in the query constraint.*

Running Example. Referring to Table 4.1, if the query constraint on Monday specifies selecting a thriller, then the minimum number of ballot modifications required to ensure that is 1, where *Scream* in Jack's ballot is swapped by *Fargo*. If instead *Inception* is to be made the winner, this will require 3 ballot modifications. Hence, the margin to satisfy the query constraint of Monday is 1.

Theorem 8. MqIRV *is NP-Complete, even when $\ell = 2$.*

Proof. (sketch). Consider an election in which m voters need to elect $k = 1$ candidate out of n candidates. In the election, each voter casts his/her ballot for two candidates in ranked order. The final candidate is determined using the IRV process. For a given instance of the election, we define the margin as the number of ballot changes required to ensure that a specific candidate wins.

We prove that computing the margin is NP-Complete. Our proof is inspired by the NP-Hardness proof of [25]. It is straightforward that the problem is in NP since the outcome of an IRV election can be computed in polynomial time. The NP-hardness is proved by reduction from the 3-Exact Cover problem (3XC). In this problem, we are given a universal set $\{e_1, \ldots, e_{3n}\}$, and $m \geq n$ subsets $S_1, \ldots, S_m$ of size 3 each. We need to determine whether there are n sets whose union is the universal set.

Suppose that we are given an instance of the 3XC problem. We show how to define a related IRV margin problem and then prove that the IRV has a margin n if and only if the answer to the respective 3XC problem is Yes.

The IRV problem has $2m + 3n + 2$ candidates $b_1, \ldots, b_m$, $c_1, \ldots, c_m$, $e_1, \ldots, e_{3n}$ and u, v. We must ensure that u wins the election. There are $6m + m^2 + m(m+5) + 3n(2m+10) + 2m + 11 + 2m + 13 = 2m^2 + 6mn + 15m + 30n + 24$ ballots as follows:

- For every $i \in [1..m]$, let $S_i = \{e_x, e_y, e_z\}$. There are 6 ballots that we call "cover ballots". These ballots are two of each $[b_i, e_x]$, $[b_i, e_y]$, and $[b_i, e_z]$
- For every $i \in [1..m]$ there are m ballots $[b_i, c_i]$
- For every $i \in [1..m]$ there are $m + 5$ ballots $[c_i, b_i]$
- For every $i \in [1..3n]$ there are $2m + 10$ ballots $[e_i, v]$
- There are $2m + 11$ ballots $[v, u]$
- There are $2m + 13$ ballots $[u, v]$

Suppose that the 3XC instance has an exact cover. Let the indices of the sets in the cover be $j_1 \ldots, j_n$. We change n ballots as follows. For every $i \in [1..n]$ change a ballot $[b_{j_i}, c_{j_i}]$ to $[c_{j_i}, b_{j_i}]$.

We successively eliminated all candidates who got the least number of votes, which is initially $m + 5$. There are m candidates with this number of votes: $m - n$

candidates c_x, for $x \in [1..m] \setminus \{j_1 \ldots, j_n\}$, and n candidates b_x, for $x \in \{j_1 \ldots, j_n\}$. As a result of eliminating the $m-n$ candidates c_x, the number of votes of the candidates b_x, for $x \in [1..m] \setminus \{j_1 \ldots, j_n\}$ jumps to $2m+11$. As a result of eliminating the n candidates b_x, the number of votes of the candidates c_x, for $x \in \{j_1 \ldots, j_n\}$, jumps to $2m+5$. Also, since the union of the n sets S_x, $x \in \{j_1 \ldots, j_n\}$, is the universal set, the elimination of b_x in the $6n$ "cover ballots" causes the number of votes of *every* e_i to jump to $2m+12$.

Next, the n remaining candidates c_x, for $x \in \{j_1 \ldots, j_n\}$, with $2m+5$ votes are eliminated. This does not change the vote of any other candidate. Lastly, the $m-n$ candidates b_x, for $x \in [1..m] \setminus \{j_1 \ldots, j_n\}$, and v each with $2m+11$ votes are eliminated. None of the e_i is eliminated at this point because all of them have $2m+12$ votes. Then, all e_is will be deleted, each with $2m+12$ votes, and, finally, u wins with $2m+11+2m+13 = 4m+24$ votes.

We need to prove the other direction. Namely, if the margin is n then there is an exact cover. Suppose that the outcome of the elections can be changed to be u by at most n ballot changes. Since candidate v has one more vote than each of the $3n$ candidates $e_1, \ldots, e_{3n}$, we need to increase the votes of all the candidates $e_1, \ldots, e_{3n}$ by at least 2 so that none of the e_i is eliminated before v is eliminated. Because if any of e_is is eliminated before v is eliminated, then the second choice of e_i's ballot goes to v and the votes of v increase to $4m+21$. Then all e_i and u will be eliminated, and v wins the election, and u loses. The only way to ensure that none of e_is is eliminated before v is by eliminating some of the candidates b_j. This can be done by ballot changes that reduce the number of votes of some of the candidates b_j by 1 and increase the number of votes of the respective candidates c_j. This will cause some candidates b_j to be eliminated and thus increase the votes of the resulting elements e_i in the "cover ballots" corresponding to these candidates b_j. Since we can make only n

ballot changes and since the cover ballots of any candidate b_j change the votes of only the 3 candidates from $\{e_1, \ldots, e_{3n}\}$ that correspond to the set S_j, the n candidates b_j eliminated first must correspond to an exact set.

□

Problem Definition 2. MqKIRV *(IRV Margin satisfying k query constraints.) Given a set of ballots $\mathcal{B}$ eliciting m voters ranked preferences of up to ℓ positions over a given set C of n candidates, and a query with k constraints, each specifies a subset of the candidates, find the minimum number of ballots that need to be modified in order to ensure that the winners of k independent invocations of the IRV election (each starting from the original ballots) belong to the respective subsets specified k query constraints.*

Theorem 9. MqKIRV *is NP-Complete, even when $\ell = 2$.*

Proof. (sketch.) Follows trivially from Theorem 8. □

Running Example. Considering the running example again (Table 4.1), $k = 3$ and the ballots are shown in Table 4.2. The winner for thriller is *Fargo* (margin = 1), for sci-fi it is *The Last Jedi* (margin = 0), for horror it is *Scream* (margin = 1). The minimum number of ballot modifications (margin) required to ensure all three constraints is 1+0+1 = 2.

4.3 Algorithms for MqKIRV

In this section, we first show the relationship between **MqKIRV** and **MqIRV**. Following this, we first focus on designing exact solutions for **MqIRV**.

Theorem 10. *An optimal solution for* **MqKIRV** *corresponds to solving* **MqIRV** *optimally k times.*

Proof. **(sketch.)** It follows from the problem definitions that an optimal solution of **MqKIRV** can be expressed as a sum of k independent **MqIRV** instances. Therefore, solving **MqKIRV**optimally is equivalent to solving each of the k **MqIRV** instances optimally.

□

We therefore turn our focus to designing exact solutions for **MqIRV**. In Subsection 4.3.2, we discuss AlgExact, a branch-and-bound algorithm that is capable of pruning effectively, following which we present a non-trivial integer programming formulation of **MqIRV** in Subsection 4.3.3.

4.3.1 Required definitions

We first give some definitions that will be useful when discussing our algorithms.

Signature. Let $\mathcal{S}$ be the set of all possible partial or total rankings over C (including those that do not appear in $\mathcal{B}$). A signature $s \in \mathcal{S}$ is one such partial or total ranking. The total number of possible signatures is $|\mathcal{S}| = \sum_{x=1}^{\ell} \binom{n}{x} \cdot x!$. For example, both $\{$*Fargo, The Last Jedi*$\}$ and $\{$*Scream, Fargo, The Last Jedi, Return Of Jedi*$\}$ are valid signatures even though the former is not present in Table 4.2.

Tally $t_r(c)$ or first choice votes. The tally or first choice votes of a candidate c at round r, denoted as $t_r(c)$, is defined as the number of votes in round r, where c is the first choice candidate. Using the running example, tally of *The Last Jedi*, *Scream*, and *Star Wars* at the beginning of round 5 are: t_5(*The Last Jedi*) $= 5$, t_5(*Scream*) $= 3$, and t_5(*Star Wars*) $= 2$.

4.3.2 AlgExact for MqIRV

Next, we propose an algorithmic framework AlgExact that is an exact solution of the **MqIRV** problem. The algorithmic solution is developed by creating a branch and bound tree, akin to two prior works [42, 142].

For a given winner w, the solution considers all possible permutations of the candidates that need to be eliminated (i.e., $(n-1)!$), where each permutation represents an elimination order simulating $n-1$ run-off rounds of IRV. The height of the tree is at most n. Each node of the tree contains two values: (a) an elimination order π, (b) a score that represents the number of ballot modifications to realize π (we formalize that as DistTo below). Each edge of the tree represents the next candidate to be eliminated. An artificial root node connects the branches of the subtree, where each subtree represents a $w \in W$ as the winner, where W is the constrained winner set specified by the query. Except for the fake root node, the relationship between any parent and child nodes in the tree is as follows: (i) At any parent node with elimination order π, the child node has elimination order $\pi' = c + \pi$ where $c \in C - \pi$, (ii) and DistTo(π) $\leq$ DistTo(π') [142]. The leaf nodes end with a full permutation, where the last candidate is from W. The maximum number of possible leaf nodes is $= |W| \times (n-1)!$. To that end, we formalize a sub-problem DistTo that AlgExact needs to solve repeatedly.

Problem Definition 3. DistTo*. Given an elimination order over the candidates π (complete or partial order, $|\pi| \leq |C|$) and a database of ballot profiles $\mathcal{B}$, find the minimum number of ballots that must be modified to achieve π.*

Theorem 11. DistTo *is NP-hard, even when $\ell = 3$.*

Proof. (sketch). First, we prove that DistTo is NP-hard when instead of ballot modifications we consider ballot deletions. The proof is by reduction from the 3-Exact Cover problem (3XC) described earlier. In the 3XC problem we are given a universal set $\{v_1, \ldots, v_{3n}\}$, and $m > n$ subsets $S_1, \ldots, S_m$ of size 3 each. We need to determine whether there are n subsets whose union is the universal set. Given an instance of the 3XC problem, we show how to reduce it to an instance of DistTo. The instance of DistToconsists of $3n+1$ candidates $v_1, v_2, \ldots, v_{3n+1}$, and the elimination order $\pi = v_1, v_2, \ldots, v_{3n+1}$ ($\pi[1] = v_1$ is eliminated first, and $\pi[3n+1] = v_{3n+1}$ is the winner). We show that this elimination order can be achieved with n ballot deletions iff the 3XC instance has a positive answer. The polynomial number of ballots in the instance varies in size from 3 to 1 and is described below.

Ballots of size 3: There are m ballots of size 3, one per every subset S_i, $1 \leq i \leq m$. Consider a subset $S_i = \{v_x, v_y, v_z\}$. From now on, we assume that the subset is "ordered", that is, $1 \leq x < y < z \leq 3n$. For every such subset S_i, the ballot (v_x, v_y, v_z) is added, namely v_x is the top candidate in the ballot, v_y is the second candidate, and v_z is the bottom candidate.

Ballots of size 2: For $1 \leq x < y \leq 3n$, let c_{xy} be the sum of the following 2 numbers: (1) number of ballots of size 3 in which v_x is the top candidate and v_y is the second candidate and (2) the number of ballots of size 3 in which v_x is the second candidate and v_y is the bottom candidate (note that the index of the top candidate in this case is lower than x). Let $maxc_x = \max_{y=x+1}^{3n}\{c_{xy}\}$. For every $x < y \leq 3n$, there are $maxc_x - c_{xy}$ ballots of size 2 consisting of candidate v_x as the top candidate and v_y as the second candidate. There are also $maxc_i$ ballots consisting of candidate v_i as the top candidate and candidate v_{3n+1} as the second candidate.

The total number of size 2 ballots is bounded by $6nm - 2m$ since there are at most $(3n-1) \cdot maxc_x$ size 2 ballots with v_x as the top candidate for $1 \leq i \leq 3n$, and $\sum_{x=1}^{3n} maxc_x \leq 2m$.

Ballots of size 1: For $1 \leq x \leq 3n$, let d_x be the total number of ballots of size 3 and size 2 in which v_x is the top candidate. Let $maxd = \max_{y=1}^{3n}\{d_y\}$. There are $maxd - d_x$ ballots of size 1 consisting only of candidate v_x as the top candidate and the only candidate. There are also $maxd - 1$ ballots consisting of only candidate v_{3n+1} as the top and only candidate. The number of ballots of size 1 is bounded by $18n^2m - 3nm$ since at most $3n$ candidates have single ballots, and for each of these candidates, there are at most $m + 6nm - 2m$ ballots of size 1, since this is an upper bound on the number of ballots of size 2 and 3 per candidate.

We prove that if there is an exact cover, then the margin is n. Suppose that the 3XC instance has an exact cover consisting of n sets. Each such set corresponds to a ballot of size 3. We call these ballots the "cover ballots". For $1 \leq x \leq 3n$, let $b(x)$ be the unique cover ballot containing x. We prove below that after deleting the n cover ballots the IRV process will result in the elimination order $v_1, v_2 ... v_{3n+1}$.

By our construction, before the deletion of the cover ballots, each of the candidates $v_1, \ldots, v_{3n}$ is the top candidate on the $maxd$ ballots and v_{3n+1} is the top candidate on the $maxd - 1$ ballots. Since the candidates on every ballot are ordered, v_1 must be the top candidate in ballot $b(1)$ and thus after the removal of this ballot, v_1 is the top candidate in $maxd - 1$ ballots. Also, since no candidate appears more than once in the cover ballots, after their removal, each of the candidates $v_2, \ldots, v_{3n}$ is the top candidate on either $maxd - 1$ or $maxd$ ballots. Recall that ties are broken arbitrarily, and thus we can eliminate v_1. As a result of the elimination of v_1 the top candidate in all ballots that included v_1 (and are not of size 1) is updated. By our construction, there are exactly $maxc_1$ such ballots for each of the candidates $v_2, \ldots, v_{3n+1}$. After the elimination of v_1, v_2 must be the top candidate in ballot $b(2)$ and therefore after the removal of this ballot v_2 is the top candidate in $maxc_1 + maxd - 1$ ballots. Again, no candidate can be the top candidate in less than $maxc_1 + maxd - 1$ ballots and thus v_2 can be eliminated. Continuing in the same manner, after the elimination of $v_1, \ldots, v_{x-1}$, candidate v_x must be the top candidate in ballot $b(x)$ and thus after the removal of this ballot v_x is the top candidate in $\sum_{y=1}^{x-1} maxc_y + maxd - 1$ ballots and can be eliminated as dictated by the required elimination order.

In the other direction, we prove that if the margin is n then there is an exact cover. To achieve this goal, we show that any set of ballots whose removal results in the elimination order $v_1, v_2 ... v_{3n+1}$ must include the candidates $v_1, v_2 ... v_{3n}$. We prove this by contradiction. Assume that this is not the case and that there exists a set of ballots that do not include a candidate v_x whose removal results in the required elimination order. Let v_x be the candidate with the minimum index that is not included in the deleted ballots. In this case, by our construction, when v_x is about to be eliminated, it is the top candidate of $\sum_{y=1}^{x-1} maxc_y + maxd$ ballots, while v_{3n+1} is the top candidate of $\sum_{y=1}^{x-1} maxc_y + maxd - 1$ ballots. A contradiction. Clearly, the only way to delete n ballots that include all $3n$ candidates $v_1, v_2...v_{3n}$ is by choosing ballots of size 3 that correspond to an exact cover.

Next, we extend this proof to the case of ballot modifications. We use the same ballot profile as before with only one difference: candidate v_{3n+1} has $maxd - n - 1$ ballots, that is, $n + 1$ fewer ballots than any other candidate (instead of having 1 ballot less than the others). By a similar reduction, it can be shown that in this scenario, the 3XC problem instance has an exact cover iff the optimal solution to the DistToinstance consists of n ballot modifications where the ballots removed in these modifications include candidates $v_1, v_2...v_{3n}$ and each of the added n ballots includes candidate v_{3n+1} as the top and only candidate.

□

AlgExact explores the tree level by level (refer to Figure 4.1) and makes an attempt to prune part of the tree based on the lower bound of a branch (which corresponds to an elimination order) and the **MqIRVUB** of the **MqIRV** instance.

Definition 9. ***Upper bound of an instance*** **MqIRVUB.** *Given an* **MqIRV** *instance,* **MqIRVUB** *is defined as an upper bound of the number of ballots that must be modified to satisfy the query constraint, namely,* **MqIRV** $\leq$ **MqIRVUB**.

Definition 10. ***Lower bound (*DistToLB*) of* DistTo*(*π*).*** *Given* π*,* DistToLB *is the number of ballots that must be modified so that each* $c[i] \in \pi$ *is eliminated before each* $c[j] \in \pi$ *(where* $i < j$*) and* DistToLB(π) $\leq$DistTo(π).

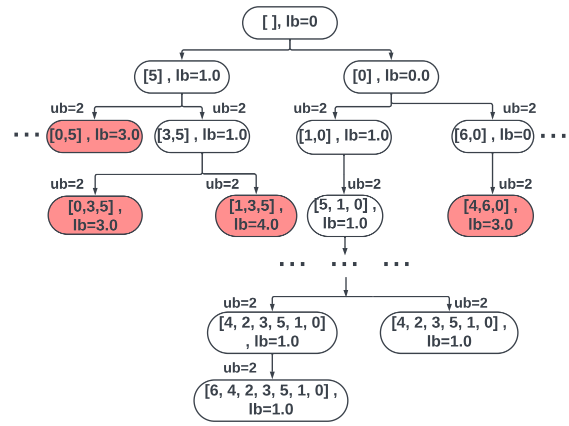


Figure 4.1 Partially explored tree for ALGEXACT , the movies are represented with their ids, where red nodes and their subtrees are pruned.

Running Example. Figure 4.1 shows one such partially constructed tree Example 4.1.1. The movies are represented by their unique ids, and any red node and the sub-tree under them are fully pruned. Each such red node has DISTTOLB(DISTTO) that is not smaller than the **MqIRVUB** of the **MqIRV** instance (e.g., DISTTOLB(DISTTO$[1, 3, 5]$) $= 4$ is larger than **MqIRVUB** $= 2$). Compared to prior works [42, 142], we propose both effective as well as computationally efficient **MqIRVUB** and DISTTOLB solutions, as we discuss in Section 4.4.

4.3.3 IP for MqIRV

MqIRV can be solved using integer linear programming. The objective function of such a linear program measures the number of ballot modifications required such that the winner is the preferred candidate. Next, we describe how to construct an integer linear program to solve this problem.

For each ballot signature $s \in \mathcal{S}$, let m_s denote the number of ballots with signature s in the original ballot profile. Define $m = \sum_{s \in \mathcal{S}} m_s$, so that m counts the total number of ballots in the original election profile. Note that the values of m_s and m are determined by the original election profile. Let a_s denote the number of ballots that are modified to s from a different ballot signature, d_s denote the number of ballots that are modified from s to another ballot signature, and y_s denote the

Algorithm 5 ALGEXACT

Input: Ballot profile $\mathcal{B}$, set of Candidates C, set of preferred candidates W.

Output: **MqIRV**

$ub = \infty$

$lb = 0$

initialize priority $queue$ with tuples $(w, 0)$ where $w \in W$

while $queue$.notEmpty() **do**

$\pi, lb = queue$.pop()

for $c \in C \setminus \pi$ **do**

$\pi' = c + \pi$

$lb =$ DISTTOLB$(\mathcal{B}, C, \pi')$

if $lb > ub$ **then**

prune π'

else

$queue$.add(π', lb)

end if

if $|\pi'| == n$ **then**

$ub = \min(ub, \text{DISTTO}(\mathcal{B}, C, \pi'))$

end if

end for

end while

MqIRV $= ub$

Return **MqIRV**

total number of ballots with signature s. The basic inequalities of IP formulation are as follows:

$$m_s + a_s - d_s = y_s \tag{4.1}$$

$$m \geq y_s \geq 0 \tag{4.2}$$

$$m_s \geq d_s \geq 0 \tag{4.3}$$

$$a_s \geq 0 \tag{4.4}$$

Equation (4.1) requires that the number of ballots with a new signature s be equal to the number of ballots that originally had the signature s, plus the number that changed from something else to s, minus the number that changed from s to something else. Equation (4.2) constrains that the number of ballots that end with signature s cannot be more than the total number of ballots that were cast in the election. The next two equation mandates: the number of ballots that are modified to have signature s must be nonnegative, and one cannot change more ballots of signature s than that were originally reported. The next constraint is that the total number of ballots changed *from* any signature is equal to the total number of ballots changed *to* any signature.

$$\sum_{s \in \mathcal{S}} a_s = \sum_{s \in \mathcal{S}} d_s \tag{4.5}$$

The next two constraints correspond to the elimination order. Assume C is the set of all candidates which is given to us. For every pair $\{c_i, c_j\} \subseteq C$, define u_{c_i,c_j} as a binary variable that is 1 iff candidate c_j is eliminated before candidate c_i. The following constraints guarantee that the variables u_{c_i,c_j} define an order. Equation (4.6) constrains it to be antisymmetric and Equation (4.7) constrains it to satisfy the triangle inequality.

$$u_{c_i,c_j} + u_{c_j,c_i} = 1 \qquad \forall \{c_i, c_j\} \subseteq C \tag{4.6}$$

$$u_{c_i,c_j} + u_{c_j,c_r} + u_{c_r,c_i} \geq 1 \qquad \forall \{c_i, c_j, c_r\} \subseteq C \tag{4.7}$$

Let signature $s = c_1, c_2, \ldots, c_\ell$, where c_x is the x-th candidate on the ballot, c_1 is the top choice while c_ℓ is the bottom. For a signature s and candidates c_i and c_j (which may be equal), define the bit v_{s,c_i,c_j} to be 1 iff signature s is a ballot in which c_i is the top candidate when candidate c_j is eliminated. Bit v_{s,c_i,c_j} is trivially 0 if c_i does not appear in $\mathcal{S}$. Bit v_{s,c_1,c_1} is trivially 1. Otherwise, we have the following constraint that is written below as a product of bits. (Later, we show how to convert it to linear constraints.) Let $y_{i,j}$ be u_{c_i,c_j} if $i \neq j$ and 1, otherwise.

$$v_{s,c_i,c_j} = y_{i,j} \cdot \Pi_{x=1}^{i-1} u_{c_j,c_x} \tag{4.8}$$

This constraint ensures that all the candidates $c_1, c_2, \ldots, c_{i-1}$ are eliminated before c_j is eliminated, and in case $i \neq j$ c_i is eliminated after c_j is eliminated. Thus signature s contributes to c_i's tally when c_j is eliminated.

The next constraint is for every ordered pair of candidates $c_i \neq c_j$. It guarantees that if $u_{c_i,c_j} = 1$, namely c_i is eliminated after c_j, then in the round in which c_j is eliminated the number of ballots in which c_i is the top candidate is at least the number of ballots in which c_j is the top candidate. The constraint is written as a product of bits and an integer (later, we show how to convert it to linear constraints).

$$\sum_s \left(y_s \cdot v_{s,c_i,c_j}\right) \geq u_{c_i,c_j} \cdot \sum_s \left(y_s \cdot v_{s,c_j,c_j}\right) \tag{4.9}$$

If we want to force candidate c_i to be the winner we need to add the constraints $u_{c_i,c_j} = 1$, for every $c_j \neq c_i$. Alternatively, if we want to force candidate c_i *not* to be the winner we need to add the constraint $\sum_{j \neq i} u_{c_j,c_i} \geq 1$. Also, we can change the objective function to count only additions or only deletions or any linear combination of additions, deletions, and modifications. Finally, we set the objective function to be: minimize $\sum a_s$, which is the number of ballots modifications.

In the last two constraints, we used (i) product of bits, and (ii) product of a nonnegative number and bits. We show how to linearize these two constraints. Let $u_1, \ldots, u_x$ be x bits. The constraints that replace $z = \Pi_{i=1}^{x} u_i$ are as follows.

$$\begin{aligned} & z \le u_i && \text{for } i \in [1, x] \\ & z \ge \sum_{i=1}^{x} u_i - (x-1) \\ & z \ge 0 \end{aligned}$$

Similarly, we can linearize the product of a nonnegative number and a bit as long as we have an upper bound on the number. Let A be a non-negative number, and let u be a bit. Assume that M is an upper bound on A. The constraints that replace $Z = A \cdot u$ are as follows.

$$\begin{aligned} & z \le A \\ & z \le uM \\ & z \ge A + (u-1)M \end{aligned}$$

We can extend this method to linearize also the product of a nonnegative number and several bits.

4.4 Efficient Algorithms

This section is dedicated to further investigation of computational efficiency. In Subsection 4.4.1, we first discuss an improved DistToLB calculation that AlgExact makes use of. In Subsection 4.4.2, we discuss an improved **MqIRVUB** algorithm that is computationally efficient. Interestingly, this algorithm also serves as an efficient heuristic for the **MqIRV**problem. In Subsection 4.4.3, we discuss that when only

Table 4.6 Efficiency Improvement Using **MqIRVUB** and DistToLB For The Running Example

Algorithm	Efficient AlgExact	Blom
Number of IP calls	Horror: 1 Sci-Fi: 1 Thriller: 2	Horror: 143 Sci-Fi: 108 Thriller: 107
Runtime (s)	0.057	0.626

ballot additions are allowed, DistTo becomes a computationally easy problem, for which an efficient algorithm could be designed.

4.4.1 An improved DistToLB algorithm

In this section, we discuss an improved lower bound calculation algorithm for DistTo(π). The intuition is the following: given π and two candidates c and c', if c needs to be eliminated before c' in round i, where $t_i(c)$ and $t_i(c')$ are the number of first choice votes of c and c' in round i, respectively, then at least $\left\lceil \frac{t_i(c)-t_i(c')}{2} \right\rceil$ number of first choice votes from c needs to go to c'. That is, lb, the lower bound of round i is calculated as the half of the difference of tally between c and c'. Finally, the maximum over all of these is returned as the output of the algorithm. Algorithm 6 has the pseudocode.

Running example. Lets assume, π = [*Return Of Jedi*, *Fargo*, *Scream*] = $[4, 6, 0]$ where 4 is eliminated first. Initially, t_1(*Return Of Jedi*) = 6, t_1(*Scream*) = 3, t_1(*Fargo*) = 1. To ensure *Return Of Jedi* is eliminated, at least $\max(\left\lceil \frac{6-1}{2} \right\rceil, \left\lceil \frac{6-3}{2} \right\rceil) = 3$ ballot modifications are required. After *Return Of Jedi* is eliminated, t_2(*Scream*) = 5, t_2(*Fargo*) = 4. Required modifications of the ballot to ensure that *Scream* wins $= \left\lceil \frac{5-6}{2} \right\rceil = 0$. Therefore, $lb = \max(3, 0) = 3$.

Using the running example, Algorithm 6 reduces a significant number of DistTo (which is solved using IP) calls. For example, lb = DistToLB($[4, 6, 0]$) =

$3 \leq$ DistTo($[4, 6, 0]$). Hence AlgExact prunes the branch $[4, 6, 0]$ without having to make an expensive DistTo call (this is because lb for this branch $> ub$). Table 4.6 shows efficiency improvement using DistToLB and **MqIRVUB** inside AlgExact over prior work [42].

Algorithm 6 Algorithm for DistToLB

```
Input: Set of ballots 𝓑, an elimination order π
Output: DistToLB(DistTo(π))
lb = 0
while |π| > 1 do
    c = π.pop_front()
    for c′ ∈ π \ e do
        lb = max(lb, ⌈(t_i(c) − t_i(c′))/2⌉)
    end for
end while
Return lb
```

Theorem 12. *Algorithm 6 returns a valid lower bound for* DistTo*(π).*

Proof. **(sketch.)** Each round of the algorithm calculates the half of the difference of the first choice votes between the eliminated candidate and other standing candidates based on π. Notice that the eliminated candidate must have fewer or equal votes in its tally than any of the standing candidates. For any pair of candidates, the minimum number of ballot modifications required to ensure that the eliminated candidate has less or equal votes than the standing candidate could be achieved by reducing $lb \left\lceil \frac{t_i(c)-t_i(c')}{2} \right\rceil$ number of votes from the eliminated one and adding that to the standing one. This is true for all pairs of eliminated and standing candidates across all rounds. Hence, the maximum of all lb's serves is indeed DistToLB(DistTo(π)). $\square$

Running Time. The running time of Algorithm 6 has two components: (i) time for calculating the tally (ii) time for finding maximum lb (line 4-6). Tally can be calculated efficiently as follows: for each candidate, maintain the number of ballots in which this candidate is the top choice as well as a linked list of all these ballots. In every elimination round pick a candidate that appears as a top candidate in the minimum number of ballots, and eliminate this candidate by going over its linked list and adding each ballot in the linked list to the next surviving candidate (and update this candidate's number of ballots). While finding the next surviving candidate delete the ones who are already eliminated from the ballot. This way the number of operations done on a single ballot during the tally calculation is $O(\ell)$. Hence, the running time for calculating the tally is $O(m\ell)$. To find the maximum of lb in each of the n rounds (line 4-6) the algorithm iterates over the remaining $O(n)$ candidates. This totals to $O(n^2)$ time. Hence, running time for Algorithm 6 is $O(n^2 + m\ell)$.

4.4.2 Algorithm AlgApprx

In this section, we discuss a highly scalable Algorithm AlgApprx which could be used as a subroutine inside AlgExact to calculate **MqIRVUB**, as well as, could serve as a standalone algorithm to solve **MqIRV**.

The basic idea of AlgApprx simply leverages the fact that for every possible winner $w \in W$, w must have more first choice votes (tally) than the rest of the candidates ($e \in C \setminus w$). An upper bound of ballot modification to ensure the winning of candidate w is thus the maximum difference in the first choice votes (tally) between w and each e. Finally, given W, **MqIRVUB** is the smallest (minimum) over these bounds considering $w \in W$.

Algorithm 7 has the pseudocode, which simulates $n-1$ rounds of IRV run-offs for each $w \in W$. In round i, the candidate e with the smallest tally is removed from C. After that the remaining first choice votes of e are redistributed and the tally of

Algorithm 7 Algorithm AlgApprx: An Improved **MqIRVUB**

Input: $\mathcal{B}$, candidate set C, winners W.

Output: **MqIRVUB**or margin

MqIRVUB$= \infty$

for $w \in W$ **do**

  $ub = 0, C' = C$

  $i = 1$

  while $i \leq n - 1$ **do**

    $e = \arg\min_{c \in C \setminus w} t_i(c)$

    C.remove(e)

    Distribute e's vote following IRV rules and update tally of the remaining candidates

    $ub = \max(ub, [t_i(e) - t_i(w)])$

  end while

  $C = C'$

end for

MqIRVUB $= min($**MqIRVUB**$, ub)$

Return **MqIRVUB**or margin

the remaining candidates is updated. The current upper bound ub is updated by the difference $t_i[e] - t_i[w]$ of tally between the eliminated candidate e and w (indeed, if $t_i[e] - t_i[w]$ number of extra votes could be added to w, it will never get eliminated before e). Finally, if $|W| > 1$, Algorithm 7 runs for all $w \in W$ and the minimum of the ub's is returned as the output of **MqIRVUB** problem.

Running example. Consider that *American Psycho* is the preferred winner ($w =$ *American Psycho*). Initially, it has 1 ballot in its tally. The movies *Fargo*, *Return Of Jedi*, *Inception* (having ballot 0, 1 and 1 respectively) are eliminated in the first three rounds. To ensure that *Star Wars* with ballot 2 gets eliminated, 1 ballot needs to be added to *American Psycho.* Similarly, to ensure that *Scream* is eliminated next, 2 ballots must be added to *American Psycho.* In the last round, *The Last Jedi* will have 8 ballots in its tally and *American Psycho* will have 2 ballots. As a result, 6 more ballots are required for *American Psycho* to avoid elimination. Hence, the **MqIRVUB**(*American Psycho*) $= \max(0, 0, 0, 1, 2, 6) = 6$. Using the running example in Figure 4.1, for $W =${*American Psycho*, *Scream*}, **MqIRVUB**(*American Psycho*, *Scream*) =2, which is a tighter upper bound than ∞ and saves expensive DistTo calls.

Theorem 13. *Algorithm 7 returns a valid upper bound of* **MqIRV**.

Proof. **(sketch.)** Each round of the algorithm calculates the difference of tally between the eliminated candidate in that round and w. Let us assume that ub is the maximum of those differences after $n-1$ rounds. Indeed, if the tally of w increases by ub, w will be the surviving candidate after $n-1$ rounds of elimination. Modifying a single ballot amounts to adding a new ballot and removing an existing ballot. This could be facilitated starting from the candidate who is eliminated first, then repeat the process for the next eliminated candidate, and so on, until ub number of ballot additions has been accounted for. Similarly, the **MqIRVUB** will be the smallest of ub's for each candidate $w \in W$. □

Theorem 14. *Algorithm* ALGAPPRX*is an approximate solution for* **MqIRV**.

Proof. (Sketch). Per Theorem 13, ALGAPPRX is an upper bound of **MqIRV**. Therefore, ALGAPPRX also solves an instance of **MqIRV** approximately. □

Running Time. The running time of Algorithm 7 has two components: (i) time for calculating the tally (ii) time for finding the candidate with minimum tally. Tally can be calculated efficiently in $O(m\ell)$ time as explained in the analysis of Algorithm 6. Finding the candidate with a minimum tally can be done using two methods depending on the value of n and m. *Method 1:* Perform a linear search over all remaining candidates to find the one with minimum tally in every round. The linear search requires $O(n)$ time per round, and thus total $O(n^2)$ time in n rounds. *Method 2:* The candidate with minimum tally can be found using a min heap to store the tally of the remaining candidates. Creating the heap takes $O(n)$ time. Finding the initial candidate with the smallest tally takes constant time. A single update of the heap takes $O(\log n)$ time. The number of times heap needs to be updated is bounded by the number of ballots that need to be redistributed when a candidate is eliminated. Since we eliminate the candidate with the minimum tally, if a round has x surviving candidates then the minimum tally is no more than m/x. So summing over all elimination rounds we get that the number of heap updates is upper bounded by $m(1/n + 1/(n-1) + \cdots + 1/2)$ which is $O(m \log n)$ (Harmonic number). Hence, the total time for updating the heap is $O(m(\log n)^2)$. and the running time for Algorithm 7 is $O(m\ell + \min\{n^2, n + m(\log n)^2\})$.

4.4.3 DISTTOADDALG for DISTTOADD

Algorithm DISTTOADDALG (Pseudocode in Algorithm 8) takes $\mathcal{B}$,C,ℓ as inputs, and returns the minimum number of ballot additions to ensure π. The algorithm has two main procedures: **Add** and **Merge**. **Add** finds the number of size 1 ballot needed to ensure π. **Merge** merges multiple size 1 ballots and produces ballots up to size n.

Algorithm 8 first calls Subroutine **Add** which returns *addone*. Then it passes *addone* to Subroutine **Merge** which returns the output of DISTTOADDALG.

Algorithm 8 DISTTOADDALG

Input: $\mathcal{B}$,C, l, $\pi = \{c_1, \ldots, c_n\}$

Output: DISTTOADD

1: $addone = \mathbf{Add}(\mathcal{B}, C, \pi)$

2: $\text{DISTTOADD} = \mathbf{Merge}(\mathcal{B}, C, \pi, addone, l)$

3: Return DISTTOADD

Subroutine **Add** (Algorithm 9) returns a two dimensional array *addone*. Each element $addone[c][r]$ represents the number of ballots of size one added to candidate c's tally at round r. It repeats in $|\pi|$ rounds. In round r, it computes the tally $t_r(c)$ of candidates $c \in \pi$, as well as keeps track of the sum of ballot additions upto round $r-1$ in $t'_{r-1}[c] = \sum_x addone[c][x]$ $(x \in 1, \ldots, r-1)$. To ensure c is not eliminated in round r, $max(0, t_r(e) + t'_{r-1}(e) - t_r(c) - t'_{r-1}(c))$ number of ballots of size one ballot additions is required for c. $addone[c][r]$ is updated based on that. Finally when all the candidates in π is popped, *addone* is returned.

Subroutine **Merge** (Algorithm 10) reduces the number of ballots by merging the ballots of size 1 into ballots of size at most n. The intuition behind this subroutine is as follows. A ballot of signature (c_{x_1}) corresponding to $addone[c_{x_1}][r_{y_1}]$ can be merged with a ballot (c_{x_2}) corresponding to $addone[c_{x_2}][r_{y_2}]$ into a new ballot of signature (c_{x_1}, c_{x_2}) if $\pi^{-1}[c_{x_1}] < \pi^{-1}[c_{x_2}]$ and $\pi^{-1}[c_{x_1}] \leq r_{y_2}$. Since, first (c_{x_1}, c_{x_2}) will contribute to c_{x_1} at round r_{y_1}, and then after c_{x_1} is eliminated, this ballot will contribute to c_{x_2} at round r_{y_2}. After the merge we can reduce $addone[c_{x_1}][r_{y_1}]$ by one, keeping value of $addone[c_{x_1}][r_{y_1}]$ the same. We can keep merging ballots this way as long as it is feasible. The size of a merged ballot $(c_{x_1}, c_{x_2}, \ldots c_{x_n})$ is at most n since $\pi^{-1}[x_1] < \pi^{-1}[x_2] < \cdots < \pi^{-1}[x_n]$.

Subroutine **Merge** runs in n rounds. We maintain two variable $mergeFrom$ and $mergeTo$, initially they are 0 (line 1). In each round r, the sum of the *addone* entries in the row corresponding to candidate $\pi[r]$ is added to $mergeFrom$, and $mergeTo$ is set to the sum of the column $r+1$ (line 2-4). If we merge ballots from $mergeFrom$ with the ballots counted in $mergeTo$ then the resulting ballot will always satisfy the conditions specified above. As we are merging in n rounds, the merged ballot length will never be more than n. After the merge we reduce the $mergeFrom$ by $mergeTo$, making sure $mergeFrom$ is not negative (line 5). Finally, we return $mergeFrom$.

Running example. Consider an elimination order $\pi = \{$*Fargo*, *American Psycho*, *Return of Jedi*, *Star Wars*, *Inception*, *Scream*, *The Last Jedi*$\}$. To make sure *Inception* is not eliminated before *Star Wars* at round 4, we need to add 1 ballot of signature (*Inception*). Similarly, to make sure *Scream* is the winner, 4 ballots of signature (*Scream*) have to be added at round 6. Total ballots of size one equals $4 + 1 = 5$. We can merge (*Inception*) and (*Scream*) to (*Inception*, *Scream*). When *Inception* is eliminated this ballot counts towards *Scream*. Hence, required ballot additions = 4.

Lemma 6. *The minimum number of ballots of size one required to be added to ensure elimination order* π *is* $\sum_{c \in C} \sum_{r=1}^{|\pi|} addone[c][r]$.

Proof. **(sketch.)** Consider a round r where e is the eliminated candidate and c is a standing candidate. To ensure c is not eliminated in round c, it must satisfy: $t_r(e) + t'_{r-1}(e) \leq t_r(c) - t'_{r-1}(c)$. For a candidate c and round r, $addone[c][r]$ is the number of ballots of size one that are required to ensure c is not eliminated before e. Hence, $\sum_{c \in C} \sum_{r=1}^{|\pi|} addone[c][r]$ is the minimum number of ballots of size one required to ensure π. □

Theorem 15. DistToAddAlg *returns an exact solution.*

Proof. **(sketch.)** Using Lemma 6, *addone* (returned by Subroutine **Add**) represents all the ballots of size one required to be added to ensure π. Next, we show that Subroutine **Merge** merges maximum number of size one ballots of *addone*.

Algorithm 9 Subroutine: **Add**

Input: $\mathcal{B}$, C, π

Output: *addone*

$addone[c][x] = 0$, $\forall\ c \in C, x \in \{1, \ldots, |\pi|\}$

$r = 1$,

while $|\pi|.notEmpty()$ **do**

  $t_r(c)$ = determine tally of c at round r, $\forall c \in \pi$

  $t'_{r-1}(c) = \sum_{x=1}^{r-1} addone[c][x], \forall c \in C$

  $e = \pi.pop\_front()$

  e's first choice votes are redistributed according to IRV

  for $c \in \{\pi - e\}$ **do**

    $addone[c][r] = max(0, t_r(e) + t'_{r-1}(e) - t_r(c) - t'_{r-1}(c))$

  end for

  $r = r + 1$

end while

Return *addone*

Algorithm 10 Subroutine: **Merge**

Input: $\mathcal{B}$, C, π, *addone*

Output: *margin*

$mergeFrom = 0$, $mergeTo = 0$

for $r = 1$ to n **do**

  $mergeFrom = mergeFrom + \sum_{i=1}^{r-1} addone[\pi[r]][i]$

  $mergeTo = \sum_{j=r+1}^{n} addone[\pi[j]][r]$

  $mergeFrom = \min(0, mergeFrom - mergeTo)$

end for

Return $mergeFrom$

Subroutine **Merge** always produces a merged ballot such that after replacing the original ballots with the merged ballot, the resulting elimination order of the election does not alter. In each round the algorithm 10 merges the maximum number of ballots possible. Repeating this process n times produces minimum number for $mergeFrom$. Hence, DistToAddAlg returns optimum value of DistToAdd. □

Runtime. (a) **Add**: the runtime for counting tally is $O(m\ell)$, and for calculating *addone* is $O(n^2)$. (b) **Merge**: each cell of *addone* is visited a constant number of times, hence it takes $O(n^2)$ time. It follows that the total running time for DistToAddAlg is $O(m\ell + n^2)$.

Extension to ballots of bounded size. We remark that Subroutine **Merge** can be generalized also to the case of ballots of bounded size $\ell < n$. In this case we need to optimize the way we merge ballots as it may not be beneficial to merge a ballot (c_x) corresponding to $addone[c_x][r_y]$ where $\pi[x] = n$ and $r_y << n$ to a ballot of length $<< \ell$ as this will block us from using this ballot in future rounds (after round r_y). One way to compute the best way to merge the ballots is by modeling this problem as a min cost flow problem where the (negative) cost rewards merged ballots and the flow value is the total number of ballots of size 1.

4.5 Experimental Evaluations

In this section, we present experimental analysis and corresponding results. All 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 [15]. For brevity a subset of results are presented that are representative.

Table 4.7 Real World And Synthetic Datasets

Dataset Name	m	n
MovieLens	100k	100k
Adressa News	100k	100k
Book Crossing	278k	217k
Restaurant Reviews	1k	100
Synthetic	1m	1m

4.5.1 Experiment design

We have three goals. (a) Assess the effectiveness of **MqKIRV** in recommendation systems in comparison with margin finding based on plurality voting (Subsection 4.5.2). (b) Evaluate the quality of our designed algorithms for **MqIRV** and **MqKIRV** problems (Subsection 4.5.3).(c) Evaluate their scalability (Subsection 4.5.4).

Dataset Description There are 4 real world datasets and one synthetic dataset used for comparison. (a) The Adressa Dataset is a news dataset, where users preferences are constructed based on $m = 100k$ users' ratings over $n = 100k$ news articles. (b) MovieLens dataset contains a set of movie ratings from the MovieLens website, containing preferences of of $m = 100k$ users over $n = 100k$ movies. (c) Book crossing dataset contains $m = 278k$ users providing $1,149k$ ratings (explicit / implicit) about $n = 271k$ books. (d) The restaurant review dataset contains ratings of 1000 users over 100 restaurants. (e) A large scale synthetic dataset is generated by randomly generating $m = 1M$ ballots over $n = 1M$ candidates. The overview is presented in Table 4.7.

Baseline Algorithms The following algorithms are implemented.

1. ***Blom*** [42]. Magrino et al. [142] propose a simple lower bound based on the DistTo of any π of length n. Given two elimination orders if one is the suffix of another, then, the DistTo of the suffix could be used as the DistToLB of DistTo for the longer elimination order. Blom. et. al. [42] propose an improved lower bound over [142] based on the last round margin $l(c', c)$ between any pair of candidates c and c' (to ensure c' is eliminated before c), which is the half of the difference in their respective tallys (first choice votes). This idea is generalized to generate lower bound of margin to ensure an elimination order ending in π, which is $\max\{l(c', c)\}$, where $c' \in C - \pi$, $c \in \pi$.

2. *Random.* We implement an algorithm that runs iteratively. In the first iteration, it randomly selects a ballot and modifies it. In the next iteration, it doubles the number of selected ballots to be modified (and selects those ballots randomly), and repeats the process until query constraints are satisfied.

3. *IP for* DistToAdd. We implement an integer programming based solution for the DistToAdd problem.

These algorithms are compared against our proposed DistToLB and **MqIRVUB** solutions inside AlgExact. We also compare AlgApprx against these solutions and the implemented IP for **MqIRV**. Finally, we compare our designed solution DistToAddAlg with its corresponding IP implementation.

Measures To evaluate Goal (1), we measure the anti-plurality index [62] (smaller is better) of items selected by **MqIRV** and margin computation based on plurality voting [90, 108]. We consider an item to be disliked by a user if it appears among bottom 10 choices of the users, and then aggregate this over all the users to calculate anti-plurality index for an item. To evaluate Goal (2), we compare approximation factors of margins produced by different algorithms (margin produced by the proposed algorithm/ exact margin), as well as compare the original margin values. We finally compare the effectiveness of the proposed algorithms based on the number of expensive DistTo calls they make (smaller is better). To evaluate Goal (3), we evaluate pruning effectiveness of the algorithms and overall running time.

Query and Parameters Queries are generated randomly with varying constraints for evaluating **MqKIRV**. For evaluating **MqIRV**, we vary the size of the ballot (ℓ), number of users (m), and the number of candidates (n). We consider a various combinations over these parameters to cover a wide range of recommendation settings. The default values are $n = 10$, $\ell = 4$ and $m = 1000$.

4.5.2 Goal 1: Analyzing anti-plurality

For these experiments, Movielens dataset is used, where we choose ballots such that voters express their choices at least for 50 candidates. Set W is selected as follows: 10 randomly hated candidates form top 20 most hated candidates, plus 10 any other randomly selected candidates. Anti-plurality index of **MqKIRV** and plurality voting of the the winning candidates are presented in Figure 4.2. These results clearly indicate that **MqKIRV** results in significantly smaller anti-plurality compared to that of plurality voting.

4.5.3 Goal 2: Analyzing quality

Approximation factor. In Table 4.8, we present the approximation factors of the **MqIRV** problems solved using different algorithms. The results are shown for four real datasets. Two of the exact solutions are compared against the IP formulation of **MqIRV** and exhibit approximation ratio of 1, as expected. AlgApprx has an approximation ratio between 1.91 to 3.15. On the other hand, *Random* has an approximation ratio between 3.61 to 4.21. As analyzed analytically, DistToAddAlg is an exact solution of DistToAdd and has an approximation ratio of 1.

Margin. Figure 4.3 shows the box plot of difference in margin for AlgApprx and AlgExact varying n for all four real datasets over ten different queries. These results corroborate that AlgApprx is an effective solution across all four datasets.

We also analyze the margin difference between AlgApprx and *Random* using one synthetic dataset and three real datasets varying n up to one million. For each run, we keep the number of ballots $m = n$. Figure 4.4 shows AlgApprx always returns smaller margin than *Random*. Using MovieLens data, *Random* margin is about 20 times larger than AlgApprx.

Number of DistTo **IP calls.** Finally, we show that AlgExact requires significantly less number of IP calls compared to *Blom* (Figure 4.5). On Adressa News dataset on $n = 10$, AlgExact invokes about 17 times less number of IP calls than what *Blom* does. These results demonstrate the effectiveness of our proposed DistToLB and **MqIRVUB** solutions, compared to the state-of-the-art.

4.5.4 Goal 3: Analyzing scalability

Running time. In these experiments (Figure 4.6), we compare running time in seconds for AlgExact, AlgApprx, and *Blom* on four real world datasets by varying n, while keeping ℓ and m fixed. The exact algorithms show that running time increases exponentially with increasing n. AlgApprx is almost 24333 times faster than Blom for $n = 12$ using MovieLens dataset. While AlgExact is 7.6 times faster than Blom for $n = 12$ using MovieLens dataset.

Figure 4.7 presents effect of varying ℓ and m on running time of AlgExact, AlgApprx, and Blom on two real world datasets. As expected, running time AlgExact does not significantly vary with increasing m and ℓ, as it is mostly driven by exponential 2^n cost of branch & bound tree exploration.

Running time in very large scale data. For these experiments, we compare running time of our efficient solution AlgApprx and compare that with *Random*.

Table 4.8 Approximation Factor of The Algorithms

Dataset	AlgExact	DistToAdd	AlgApprx	*Random*
MovieLens	1	1	1.99	3.42
Adressa News	1	1	3.15	4.21
Book Crossing	1	1	1.91	3.61
Restaurant Reviews	1	1	1.94	3.67

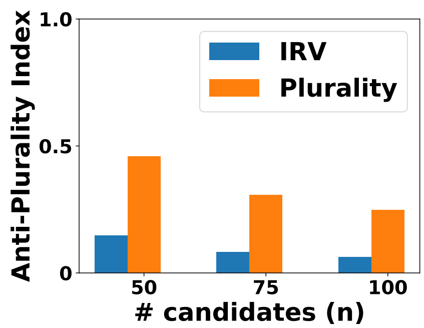


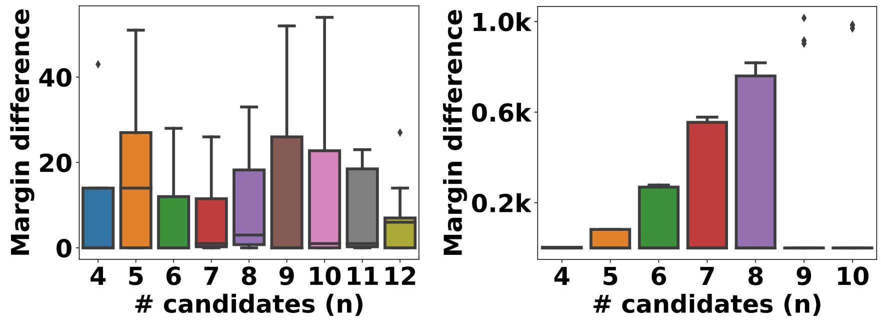
Figure 4.2 Anti-plurality index for **MqKIRV**and plurality voting.

Figure 4.8 shows that the running time of AlgApprx is significantly smaller than *Random*. Using Adressa News dataset with $n = 100k$, $m = 100k$ and $l = 4$, the runtime for *Random* is about 100 times larger than AlgApprx.

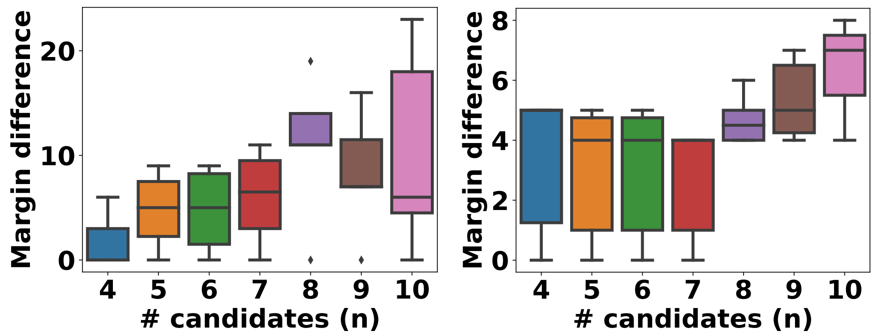
Running time of DistToAddAlg. Figure 4.9 compares the running time between our exact solution DistToAddAlg for DistToAdd with IP based implementation (DistToIPADD). DistToIPAdd runtime increases exponentially with n as expected, whereas, DistToAddAlg runs in n^2. For MovieLens dataset with $n = 10$ DistToAddAlg is 53 times faster than DistToIPAdd.

4.5.5 Summary of results

Our first observation is that, **MqKIRV** significantly enables lower anti-plurality, whereas, plurality voting does not. It indeed is an important observation, which showcases that plurality voting may end up choosing less preferred candidates unlike IRV, which is truly undesirable in recommendation settings. Our second major obser-



(a) MovieLens **(b)** Adressa News



(c) Book Crossing **(d)** Restaurant Reviews

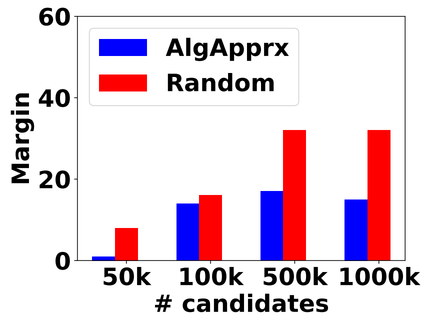
Figure 4.3 Margin difference between ALGAPPRX and ALGEXACT varying n.

vation is that our designed ALGEXACT enabled by effective lower bound DISTTOLB and upper bound **MqIRVUB** algorithm is highly effective as well as computationally efficient compared to their counterparts *Blom*. Third, ALGAPPRX exhibits empirical approximation factor around 2 (for 3 of the datasets) and runs significantly faster than the exact solutions (order of magnitude faster) and the *Random* baseline. Finally, consistent with our theoretical analysis, DISTTOADDALGreturns an exact solution for DISTTOADD, runs in polynomial time, and is significantly faster (about 53 times for some datasets) than the IP based solution.

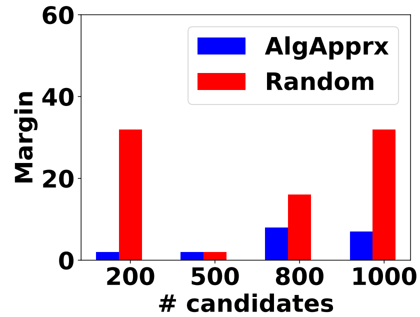
4.6 Prior Work

We present three types of related work in this section.

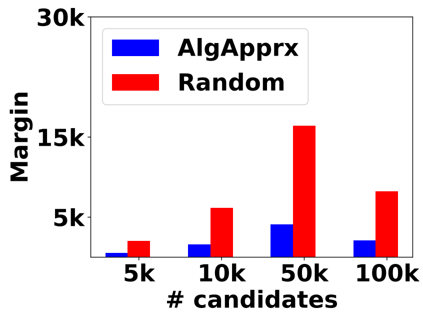
Preference aggregation in recommender systems. Preference aggregation is closely studied in the context of group recommendation [12, 13, 23, 30, 54, 129, 165, 173, 173], with the goal of selecting one or top-k items that are most suitable to the



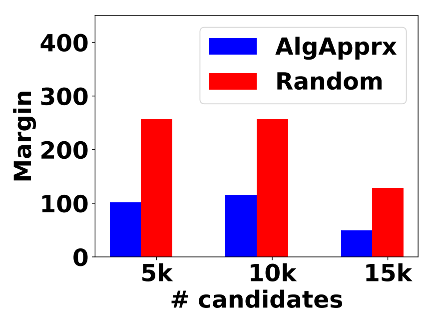
(a) Synthetic



(b) MovieLens



(c) Adressa News

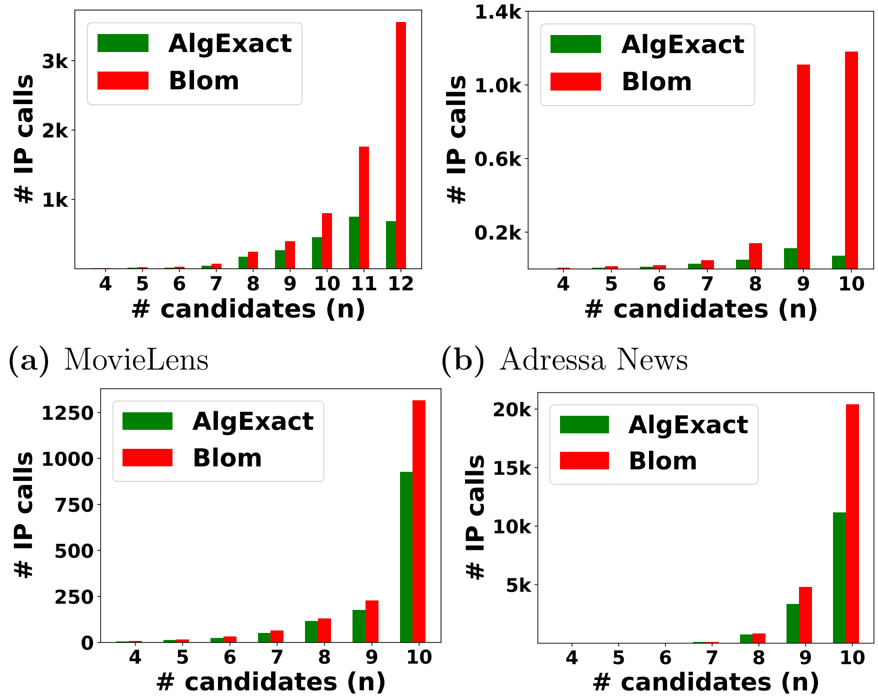


(d) Book Crossing

Figure 4.4 Margin for AlgApprx and Random.

preference of all users in the group. Preference aggregation models from social choice theory [83] are adapted and related works study their computational implications, and investigate efficiency opportunities. Related work also exists on modeling evolution of users' preference over time and adapting existing group recommendation models to accommodate that [12]. In [28], the authors present a preference aggregation algorithm designed for situations in which a limited number of users write review over a large (but finite) set of candidates. [160] has illustrated a correspondence between collaborative filtering (CF) and social choice theory. In [62], the authors empirically demonstrate that multi-stage voting methods, such as STV and IRV offer benefits over positional preference aggregation methods (e.g., plurality voting, approval voting) in the recommendation contexts (recommending tweets, movies, hashtags), by handling hyperactive users in a more equitable and fair way.

Changing original preferences. The second line of related work exists in how to minimally update original preferences of the users so that the produced outputs



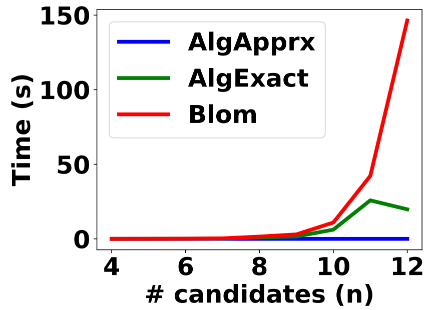
(a) MovieLens (b) Adressa News

(c) Book Crossing (d) Restaurant Reviews

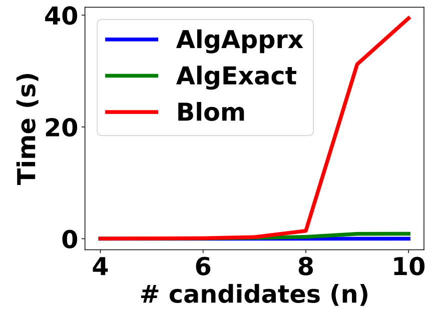
Figure 4.5 Number of IP calls for AlgApprx & AlgExact varying n.

satisfy additional criteria. Some leading criteria include, maximizing satisfaction of some specific users considering rating based preference aggregation methods in top-k recommendation [173], changing the original winner, that is, compute margin, or produce Margin of victory (MoV), or satisfy fairness criteria, [121,197], to name a few. Among these, the most relevant to this work is the prior work on computing MoV. There exists two kinds of MoV: constructive and destructive. In the constructive (destructive) version, the goal is to find the minimum number of changes to the ballots which is needed so that a special candidate is (not) elected. [202] has investigated the computational complexity and (in)approximability of computing MoV for various voting rules, including approval voting, all positional scoring rules, etc. [37] has introduced a sampling based probabilistic algorithm for finding the margin of victory, which can be used for many voting rules.

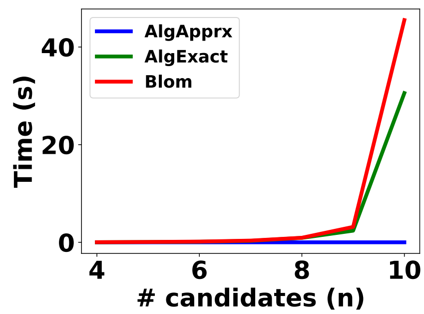
Margin of victory of multi-stage preference aggregation methods. Multi-stage methods, such as, STV and IRV, were introduced in the 19th century in electoral



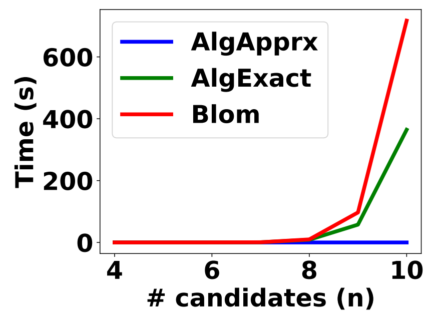
(a) MovieLen



(b) Adressa News



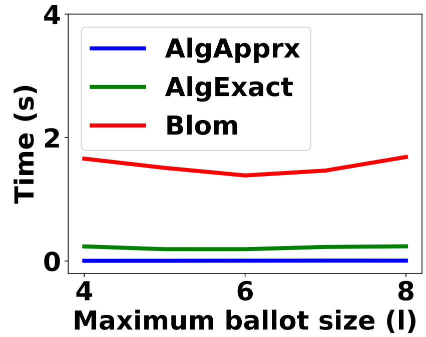
(c) Book Crossing



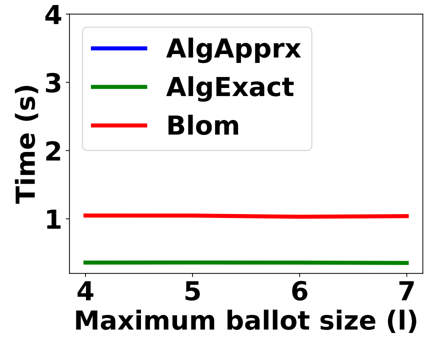
(d) Restaurant Reviews

Figure 4.6 Runtime for AlgApprx , AlgExact and *Blom* varying n.

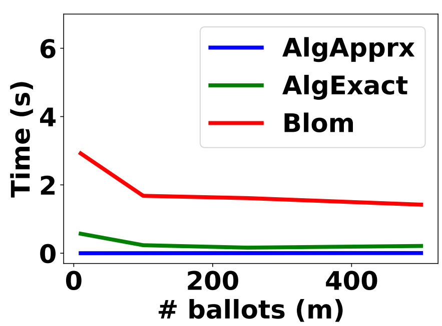
voting systems. [25] demonstrated that determining whether the MoV in an IRV election is at most 1 is NP-hard for both constructive and destructive versions. Moreover, there is no 2-approximation algorithm for it unless $P = NP$. In [71], the coalitional weighted manipulation is investigated. In the coalitional weighted manipulation, given a set of weighted ballots as the input such that each ballot with weight w can be replaced by w votes with weight 1, and also a set of blank weighted votes, the goal is to see whether it is possible to fill out the blank votes such that a special candidate is selected (or not selected). It is shown that when the number of candidates is 2, this problem is in P for both destructive and constructive versions is in P; otherwise, it is $NP-$complete. In [122], the authors have shown a branch and bound algorithm that calculates possible winners when only some part of the ballots are accessible, not all. The usage of [122] is to generate information on the result of an election and to announce it on election night, when there are still some ballots



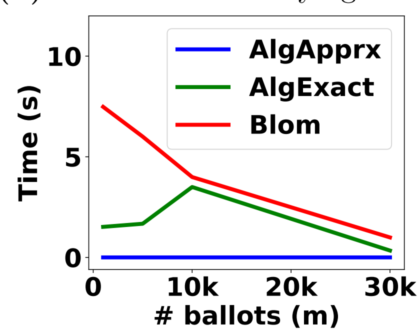
(a) MovieLens varying l



(b) Adressa News varying l



(c) MovieLens varying m



(d) Adressa News varying m

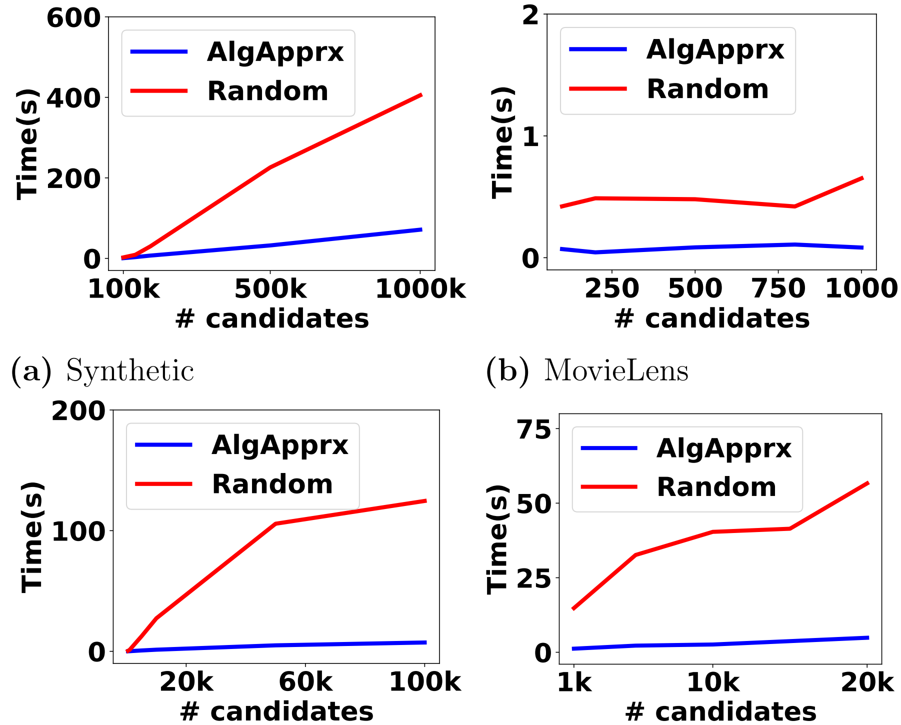
Figure 4.7 Runtime for AlgApprx , AlgExact and *Blom* varying l and m.

that have not arrived at the specified place to count the votes. MoV of IRV [149] and STV [113] are studied in many related works [142], [42], [20], [41].

In contrast, we study IRV, which is a multi-stage preference aggregation procedures [172] that uses a choice function iteratively on diminishing sets of alternatives. **MqKIRV** *is different from MoV problem, we present non-trivial hardness results and algorithmic solutions.*

4.7 Conclusion

We study the suitability of Instant Run-off Voting (IRV) as a preference aggregation method for selecting k different winners that satisfy the query constraints. We formalize an optimization problem that aims at finding the margin, i.e., the smallest number of modifications of original users' preferences (ballots) so that the selected k winners satisfy all these query constraints. We present principled models and several non-trivial algorithmic and theoretical results. Our experimental analyses

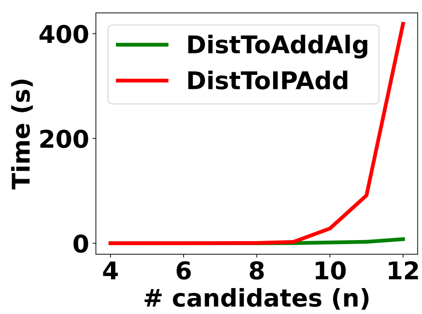


(a) Synthetic (b) MovieLens (c) Adressa News (d) Book Crossing

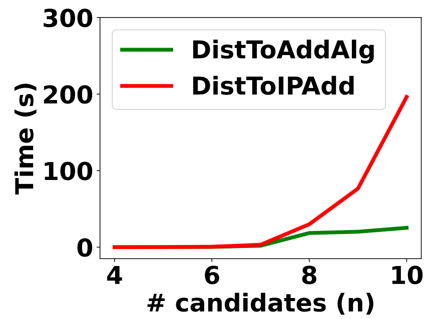
Figure 4.8 Runtime for AlgApprx and Random.

demonstrate suitability of IRV as a preference aggregation method over plurality voting, as well as corroborate our theoretical analysis.

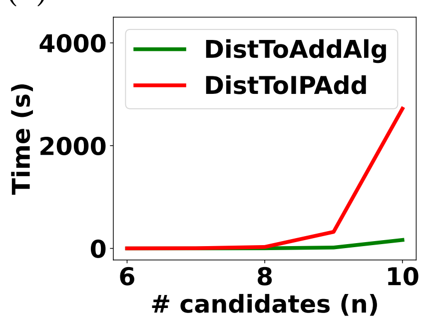
This work opens up many interesting directions - as an ongoing work, we are investigating how to design approximation algorithms with theoretical guarantees for **MqIRV**. We are also studying how our proposed solution AlgExact could be adapted to compute margin for single transferable voting (STV) schemes.



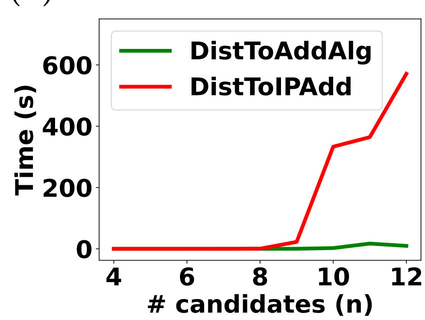
(a) MovieLens



(b) Adressa News



(c) Book Crossing



(d) Restaurant Reviews

Figure 4.9 Runtime for DistToAddAlg and DistToIPAdd.

CHAPTER 5

EQUITABLE TOP-K RESULTS FOR LONG TAIL DATA

5.1 Introduction

The proliferation of e-commerce platforms such as Amazon.com, Netflix, and Spotify.com has given rise to the so-called "infinite-inventory", which offer an order of magnitude more records (products, movies, songs) than their brick-and-mortar counter-parts [16]. This results in a long-tail market, where a handful of records get heavily exposed to the end users and a long tail of "niche" records remain relatively unknown. As a concrete example, the top-1000 highest rated movies in IMDB [124] follow a long tail distribution in terms of number of views (refer to Y-axis in Figure 5.1), even though they all have highly similar (average rating between 8.34 and 7.9) "utility" (IMDB ratings).

In Subsection 5.2.1 we describe the current process with a running example on the aforementioned IMDB-1000 datasets, how it leads to inequitable exposure of movies, and how we intend to redesign existing top-k algorithms to circumvent that. Our proposed solution advocates to return one of the *equivalent* top-k sets to the end users in a probabilistic manner, such that, after many such draws by many end users, the exposure of the records are as equitable as possible. The same static answer could still be returned if the application warrants - but when users pose generic queries [153] (e.g., top-3 movies, books) on long tail data, this will unveil interesting movies, songs, and products, that the users will not experience otherwise. *To the best of our knowledge, we are the first to study this aspect of unequal exposure inside top-k algorithms that is agnostic to any specific scoring functions.*

Problem Motivation and Models. We adapt a political theory, namely, the *Sortition Act* [75, 180] and redesign existing top-k algorithms to have them compute *a set S of multiple top-k sets that are equivalent in utility* as opposed to a fixed top-k set. Given S, an end user still draws one of the sets at random. Hence, the goal is to assign a probability distribution over S, i.e., $PDF(S)$, such that after many such draws from many end users, the records returned inside the top-k sets have as uniform selection probability as possible. We formalize **θ-Equiv-top-k-MMSP** that produces $PDF(S)$ for a given query and a scoring function $\mathcal{F}$. Each set $s \in S$ contains k number of records whose score is at most $\theta\%$ (a tunable application dependent input parameter) smaller than the optimum top-k score, and the $PDF(S)$ is computed such that the selection probabilities of the records in it are as uniform as possible. Enabling equal selection probabilities promotes equal exposure of the records. **θ-Equiv-top-k-MMSP** is rooted on maxmin fairness theory that maximizes the minimum exposure. We are aware of a few related works that we borrow inspirations from. [19] studies how to enable equal exposure in similarity search by returning points within distance r from the given query with the same probability. The bulk of the algorithmic fairness literature deals with group fairness along the lines of demographic parity [134, 197]: this is typically expressed by means of some fairness constraint requiring that the top-k results (for any k) to contain enough records from some groups that are protected from discrimination based on sex, race, age, etc. In practice these group fairness constraints hurt equitable exposure [39, 94, 101] owing to differential participation rates across sub population. Both [94, 101] study how group fairness alone can hurt equitable exposure of the records and thus define computational frameworks to promote equal selection probability in group fairness. These existing works do not have any easy extension to top-k algorithms. We study how **θ-Equiv-top-k-MMSP** alleviates exposure based fairness concerns that demographic parity based group fairness (e.g., top-k parity [134], proportionate fairness [197]) give rise to.

Technical Contributions. We formalize key definitions, such as, θ-equivalent top-k sets, selection probability of records, and present **θ-Equiv-top-k-MMSP** that has two steps **(Section 5.2)**. (A) **θ-Equiv-top-k-Sets** generates S, the set of θ equivalent top-k sets (where θ is a tunable parameter that can control how much change is desirable across different top-k sets for different applications), (B) **MaxMinFair** computes $PDF(S)$ such that the minimum selection probability of a record is maximized. We prove that the counting problem involved in **θ-Equiv-top-k-Sets** is #P-hard, which makes **θ-Equiv-top-k-MMSP** an NP-Complete problem. In **Section 5.3**, we first present an exact algorithm **OptTop-k-θ** that produces S, all θ-equivalent top-k sets and is exact in nature. We also study efficient alternatives later, which only computes a few θ equivalent top-k sets (as opposed to all). The exact algorithm is inspired by the celebrated NRA algorithm [89] but *not an easy adaptation, because of the exponential nature of* ***θ-Equiv-top-k-Sets***. At the heart of the process, **OptTop-k-θ** intends to maintain a set of candidate top-k sets, efficiently compute and maintain their best and worst possible scores through upper and lower bounds, and decide if it is safe to terminate and produce the exact S without having to read any more records. However, because the number of possible size-k sets increases exponentially with new records being read, **OptTop-k-θ** leverages an efficient data structure based on the concept of item lattice that allows efficient computation of the possible size-k sets and incremental updates of their score bounds by reusing previously calculated scores. For producing $PDF(S)$, we present a linear programming-based exact solution **Opt-SP**. For **OptTop-k-θ** , the storage space and computational cost of this lattice is $\mathcal{O}(\binom{N}{k})$, which is the theoretical lower bound, but the same structure could be made significantly lightweight, if approximation is allowed, as we discuss in Section 5.4.

In **Subsection 5.4.1**, we present **RWalkTop-k-θ** that is highly scalable to solve both **θ-Equiv-top-k-Sets** and **MaxMinFair**. It makes use of the same item lattice

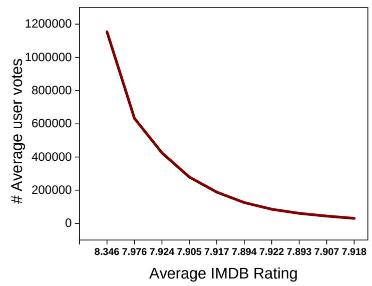


Figure 5.1 Viewership distribution of top-1000 IMDB movies.

structure described above, but builds it only partially on the go, making it significantly lightweight. **RWalkTop-k-θ** is a *probabilistic algorithm* based on random walk on the lattice that is backed by the Good Turing Test [100]. Good Turing Test is often used in population studies to estimate the number of unique species in a large unknown population [100], which we use to determine when **RWalkTop-k-θ** could stop and still discover all θ-equivalent top-k sets with high probability. Given S, **RWalkTop-k-θ** calls a highly efficient greedy solution **Gr-SP** to produce a probability distribution over it.

In **Subsection 5.4.2**, we finally design **ARWalkTop-k-θ**, an adaptive random walk based approach that solves **θ-Equiv-top-k-Sets** and **MaxMinFair** at the same time. The intuition comes from the fact (that we formally prove in the paper) that if S contains records that only appears in one and exactly one set $s \in S$, then $PDF(S)$ is a uniform probability distribution which ensures equal selection probabilities for all records. **ARWalkTop-k-θ** is similar to the random walk described in **RWalkTop-k-θ**, except it performs the random walk adaptively, by lowering the probability of the records that are already part of some valid s, and boosting the probability of the remaining records that have not been part of any valid s yet. After that, $PDF(S)$ becomes a uniform probability distribution over the sets produced during the adaptive random walk.

Experimental Evaluations (Section 5.5). Our final contributions are empirical. As discussed above, equal exposure is orthogonal to demographic parity based group fairness, such as, top-k-parity [134], proportionate fairness [197], or group exposure [178]. We empirically demonstrate **θ-Equiv-top-k-MMSP** further alleviates exposure biases of individual items in long tail data that group fairness alone gives rise to by comparing it with two related works [134, 178]. We use 4 different large scale real world datasets and two synthetic datasets to extensively evaluate our designed solutions and compare them against several intuitive baseline algorithms. Our experimental evaluations also corroborate our theoretical analysis, it terms of the quality and the scalability of the designed solutions.

Section 5.6 contains the related work and we conclude in Section 5.7, giving future research directions.

5.2 Data Model and Problem Definition

5.2.1 Running example

Consider the IMDB-1000 datbase D. The attributes are movie name, IMDB rating, year, genre, and director. Assume that a user writes a query (q) to search for top-3 movies ($k = 3$) released in year 2022. Key notations are described in Table 5.1.

Imagine only 5 movies as described in Table 5.2 are released in 2022 and they have highly similar IMDB ratings. Let the scoring/utility function $\mathcal{F}$ be the weighted relevance and max sum diversity (WRMSD in short), as proposed below (with λ = 0.5). Let IMDB ratings reflect the relevance scores of the records and diversity be computed considering genre and director values. The sorted pairwise diversity is given in Table 5.3.

In the set $s1 = \{r_2, r_3, r_5\}$, the utility score of r_2, r_3, r_5 are $6.75, 6.65, 6.45$, leading to the maximum utility score of top-3 movies to be 19.85, as shown in 5.4. Static top-k algorithms will always return $\{r_2, r_3, r_5\}$, whereas, $s2 = \{r_1, r_2, r_3\}$, $s3 =$

$\{r_2, r_3, r_4\}$, $s4 = \{r_1, r_3, r_5\}$, may also be equally desirable (all have items with high utility, leading to high set score above 19). However, if only $s1$ is always returned, this leads to little to no exposure of movies r_1, r_4.

We advocate for an alternative process, where, there exists a tunable parameter θ, which will empower the application designer to introduce variability in the top-k results to the end users (if the application warrants the same static answer, θ could be set to 0). For long tail data with generic queries [153], this process may bring forth additional interesting movies, products, songs to the end users. If $\theta = 0.03$, the goal is to create a set S of top-k sets, such that each $s \in S$ has utility score $\geq (19.85 - [0.03 \times 19.85]) = 19.25$. It is easy to notice that even with only 5 records, there are three additional sets $\{s2, s3, s4\}$ that satisfy this condition (Table 5.4).

The top-k interface however still allows users to see only one set of k results. Thus, given S, our goal is to create a probability distribution over it, $PDF(S)$. A user draws one s from S corresponding to its associated probability, such that, after many draws from many end users, the movies in S have as uniform selection probabilities as possible. Creating $PDF(S)$ is non-trivial - if one associates uniform probability (0.25) to each of the 4 sets, then, r_3 will always be over exposed (quantified by its selection probability, which is also formalized in this section), as it will always be returned to the end users, leading to 1 selection probability, whereas, r_4 will be heavily underexposed. The selection probabilities of $r_1 = 0.5$, $r_2 = 0.75$, $r_5 = 0.5$, and that of r_4 is only 0.25, as r_4 is present in only $s3$ out of the 4 sets. Our effort here is thus to produce $PDF(S)$ such that the movies in S have as uniform selection probabilities as possible.

5.2.2 Data model

Database. A database D contains N records, where each record is represented as r.

Top-k Query. A top-k query q intends to return k answers from D. We are especially interested in generic queries (e.g., top vacation spots, top movies, good books, etc).

Utility Based Scoring Functions. Given a query q and D, a utility based scoring function $\mathcal{F}$ scores each record with utility value $\mathcal{F}(r, q)$ and produces $\mathcal{F}(s, q), r \in s, |s| = k$, which is the the aggregated utility score of set s with k records.

- Relevance: $\mathcal{F}(r, q) = Rel(r, q)$, where Rel is the *relevance* between record r and query q.
- Diversity: Diversity is the dissimilarity between any two records, $Div(r_i, r_j)$ that is used to capture results that are representative of the population.

The attributes of the records could be used to calculate these values. Tables 5.2, 5.3 have some of those for Example 5.2.1.

Representative $\mathcal{F}$. Some representative utility functions appear as follows.

- Sum-relevance. $\mathcal{F}(s, q) = \Sigma_{r \in s} Rel(r, q)$
- Weighted relevance and max sum diversity (WRMSD).
$\mathcal{F}(s, q) = \lambda \times \Sigma_{r \in s} Rel(r, q) + (1 - \lambda) \times \Sigma_{r \in s} Max_{r, r_j \in \{s-r\}}$
$Div(r, r_j)$, where λ is a weight between $[0, 1]$.
- Maximal marginal relevance [55] or MMR. $\mathcal{F}(s, q) = \lambda \times \Sigma_{r \in s} Rel(r, q) + (1 - \lambda) \times \Sigma_{r \in s} Min_{r, r_j \in \{s-r\}} Div(r, r_j)$

The proposed framework is generic and extensible to any utility function, however, as we shall see later that the exact solution **θ-Equiv-top-k-Sets** requires the function to be monotonic.

Top-k Algorithms Given D, q, and an integer k, return a set s of k records from D that has the highest $\mathcal{F}(s, q)$, i.e., $|s| = k$; and s has the highest utility score, i.e., for any other set of k records s', $\mathcal{F}(s, q) \geq \mathcal{F}(s', q)$.

Promoting Fairness inside Top-k Algorithms It is easy to see that there could be more than one set of k-records that have highly similar utility score. To that end, we define the notion of equivalent size-k sets.

Definition 11. Equivalent size k sets. *Given a threshold θ, a query q and size k, two sets s_i and s_j each with k records are equivalent if the score of the set with lower score is not smaller than* a predefined threshold $\theta\%$ *of that with the higher score, i.e.,*

$$s_i \equiv s_j \text{ if } \mathcal{F}(s_i, q) \geq (1-\theta) \times \mathcal{F}(s_j, q), \text{ when } \mathcal{F}(s_i, q) < \mathcal{F}(s_j, q)$$

Running Example. In the context of example 5.2.1, when WRMSD is considered as the scoring function and $\theta = 0.03$, two equivalent size k sets with scores 19.85 and 19.7 are $s1 = \{r_2, r_3, r_5\}$, $s2 = \{r_1, r_2, r_3\}$, respectively.

Definition 12. Probability Distribution over size k sets. *Given a set S of sets, each with k records, a probability distribution $PDF(S)$ assigns a probability $P(s)$ to each $s \in S$, such that $\sum_{s\in S} P(s) = 1$.*

Definition 13. Selection probability of a record. *Given a probability distribution $PDF(S)$ of a set S containing many size k sets, the selection probability [94] of a record r is the sum of probability values of all the sets that contain r.*

$$\mathcal{P}(r) = \sum_{r\in s, s\in S} P(s) \tag{5.1}$$

Running Example. Considering the running example, uniform probability distribution $P(s_1) = P(s_2) = P(s_3) = P(s_4) = 1/4$, leads to selection probability $\mathcal{P}(r_4) = P(s3) = 1/4$, whereas, $\mathcal{P}(r_3) = P(s1) + P(s2) + P(s3) + P(s4) = 1$. Indeed, no matter which set the end users draw, r_3 will always be returned, whereas, r_4 will be returned only 1/4 of the time.

5.2.3 Problem definition and hardness

Our overarching goal is to produce top-k set of sets that are "equivalent" in utility w.r.t. the set with the highest utility (i.e., the optimum top-k set), and ensure that all records present in any of the equivalent top-k sets have an equal selection probability.

Problem Definition 4. (θ-Equiv-top-k-MMSP) Maximize Minimum Selection Probability in θ-Equivalent Top-k Sets.
Given a database D with N records, scoring function $\mathcal{F}$, threshold θ, query q, and integer k, produce a set S of equivalent top-k sets and a probability distribution $PDF(S)$ over S, such that, the minimum selection probability of a record present in any $s \in S$ is maximized. Specifically, we define the following two sub-problems.

- **θ-Equiv-top-k-Sets.** *Produce a set S of all θ-equivalent top-k sets, such that, $s \in S$ satisfies:*
  $\mathcal{F}(s, q) \geq (1 - \theta) \times argmax_{s' \in S} \mathcal{F}(s', q)$
- **MaxMinFair.** *Compute probability distributions S such that the smallest selection probability $\mathcal{P}(r)$ of a record $r \in s, s \in S$ is maximized. That is:*

$$Maximize\ Min\ \mathcal{P}(r), r \in s, s \in S, \tag{5.2}$$

We note that equal selection probability of all the records is perhaps too ideal to achieve in real world, resulting in finding no feasible solutions to many problem instances. Thus, the reasons to strive for equality also motivate a more gradual version of this goal: making selection probabilities as equal as possible [85], which adapts to the Egalitarian Social Welfare notion from the optimization standpoint - maximizing the lowest selection probability of the records present in any top-k sets. Indeed, our proposed definition **MaxMinFair**accommodates the properties of equal selection probabilities in the most generic way, and is also used in related literature [94, 101].

θ-Equiv-top-k-MMSP compared to existing fairness criteria. **θ-Equiv-top-k-MMSP** is not designed to promote group fairness - it is more aligned to individual fairness by promoting equitable exposure to records that satisfy long tail property. Related works on item-side fairness in recommendation systems [38, 61, 133, 196] is defined wrt ranked order of the top-k items, whereas, **θ-Equiv-top-k-MMSP** focuses on a set based notion (if an item is present in top-k, it has exposure, else not) which is suitable to only long tail data. As we empirically demonstrate in Section 5.5, **θ-Equiv-top-k-MMSP** complements multiple group fairness criteria that are suitable for ranking and top-k, such as top-k statistical parity [134] and group exposure [178], Overall, **θ-Equiv-top-k-MMSP** is fundamentally different from existing fairness criteria. Section 5.6 has further details.

In general, our proposed framework can accommodate any scoring function. However, when the scoring function is non-monotone, such as, MMR [55], the designed solutions become approximation.

Theorem 16. *The problem of finding the number of θ-**Equiv-top-k-Sets** is #P-hard.*

Proof. We show a polynomial time reduction from the problem of computing all maximal frequent itemsets of size at most t [109, 204] to the problem of computing all θ-equivalent top-k sets, that has a simple mapping between the number of solutions. This suffices since the problem of finding the number of σ-frequent maximal itemsets (threshold $\sigma \in [0, 1]$) with at most t items of a given 0-1 database D is known to be #P-hard [204].

We take an instance of such 0-1 database with m transactions over N items. The σ is set to be $1/m$. Given one such instance of a 0-1 database, we create an instance of our problem as follows: each item becomes a unique record r, such that $\mathcal{F}(r, q) = 1$, for an arbitrary query q. $\mathcal{F}(s, q) = \Sigma_{\forall r \in s} \mathcal{F}(r, q)$. θ is set to be any number between $[0, 1]$. A set of items is σ-frequent maximal itemset of size at most k, iff the set of records corresponding to the itemset forms a set s with score $\mathcal{F}(s, q) = k$.

Table 5.1 Table of Notations

Symbol	Definition
N	# records in D
k, q	size of result sets, query
θ,s, S	equivalence threshold, a top-k set, θ-equivalent top-k sets
$\mathcal{C}$, $\mathcal{L}$, $\mathcal{F}$	candidate set, sorted input lists, scoring function
$\mathcal{P}(r)$	selection probability of record r

Table 5.2 Records With Sorted Relevance (Example 5.2.1)

Record	Movie Name	IMDB Score
r1	Top Gun: Maverick	8.6
r2	K.G.F: Chapter 2	8.5
r3	Everything Everywhere All at Once	8.3
r4	RRR	8.1
r5	The Batman	7.9

Therefore, the number of θ-equivalent top-k sets is at least as many as the number of σ frequent maximal itemsets of size at most k. This completes the reduction. □

Theorem 17. *The θ-**Equiv-top-k-MMSP** problem is NP-Complete.*

Proof. (sketch) We omit the details for brevity. Intuitively, the hardness comes from the fact that θ-**Equiv-top-k-MMSP** needs to enumerate all θ-equivalent top-k sets, which is at least as hard as counting all such sets that is proved to be #P-hard. □

5.3 Exact Algorithms

We first describe an exact solution that solves both the sub-problems θ-**Equiv-top-k-Sets** and **MaxMinFair** exactly, thereby ensuring exact solution for θ-**Equiv-top-k-MMSP**.

The framework is described in Algorithm 11. To solve θ-**Equiv-top-k-Sets**, it runs in a loop and finds the i-th best top-k set in the i-th iteration - that is, $\mathcal{F}(s,q) =$ **TopkSets**$(i) \geq \mathcal{F}(s',q) =$ **TopkSets**(j), where $i < j$. It maintains all records that are seen throughout. This process continues until the utility score of a top-k set falls $\theta\%$ below from the optimum top-k. After that, it calls the **MaxMinFair** S to produce $PDF(S)$.

In Subsection 5.4.2, we will show how these two steps could be combined to design a highly scalable solution.

Table 5.3 Sorted Diversity List Based on Example 5.2.1

Pair of records	**(r2,r3)**	**(r3,r5)**	**(r1,r3)**	**(r3,r4)**	**(r1,r4)**	**(r4,r5)**	**(r1,r2)**	**(r2,r4)**	**(r2,r5)**	**(r1,r5)**
Diversity Score	5	5	4	4	2	2	2	2	1	1

Table 5.4 WRMSD Scores of All Set of Sets, Each With Three Movies

sets	**s2:{r1,r2, r3}**	**{r1,r2, r4}**	**{r1,r2,r5}**	**{r1,r3,r4}**	**s4:{r1,r3,r5}**	**{r1,r4,r5}**	**s3:{r2,r3,r4}**	**s1:{r2,r3,r5}**	**{r2,r4,r5}**	**{r3,r4,r5}**
score	19.7	15.6	14.5	18.5	19.4	15.3	19.45	19.85	15.25	19.15

5.3.1 Algorithm for θ-Equiv-top-k-Sets

Our proposed algorithm **OptTop-k-θ** runs in a loop by performing sorted accesses over the input lists through a cursor movement by calling **DivGetBatch()**, gradually produces **TopkSets**(i) sets whose scores monotonically decreases, and finally terminates when all θ equivalent top-k sets are found. **θ-Equiv-top-k-Sets** requires the scoring functions to be monotonic, we demonstrate **OptTop-k-θ** using one of the representative function WRMSD described in Subsection 5.2.2.

1. Generates and maintains a candidate set $(\mathcal{C}, i, j)$ of top-k sets as it reads j-th records from the cursors. $(\mathcal{C}, i, j)$ is needed for deciding **TopkSets**(i).
2. Local stopping: if the **TopkSets**(i) is present in $(\mathcal{C}, i, j)$.
3. Global stopping: if all θ Equivalent top-k Sets are found.

OptTop-k-θ borrows inspiration from the celebrated **NRA** (No Random Access) algorithm [89]. However, it is an *not an easy adaptation of **NRA**, because of the exponential nature of θ-**Equiv-top-k-Sets***. The algorithm leverages an efficient data structure based on the concept of item lattice that allows efficient computation of the possible size-k sets and incremental updates of their score bounds by reusing previously calculated scores, as described in Subsections 5.3.1 and 5.3.1, respectively.

Analyzing Sorted Access Cost Sorted access (SA) is enabled by implementing a getNext() interface, largely inspired by [89]. This cost is highly dependent on the underlying scoring function $\mathcal{F}$ and we believe there does not exist an unified way

Algorithm 11 Generic Framework for **θ-Equiv-top-k-MMSP**

Inputs: q, k, θ, database D, $\mathcal{F}$

Outputs: $PDF(S)$: probability distribution over a set S of top-k sets

$flag = 0$

$Opt = \infty$

$s =$ **TopkSets**$(1)(\mathcal{F}, D, k)$

$Opt = s.score$, $Score = Opt$

$S \leftarrow \{s\}$

i $\leftarrow$ 2

while $(Score \geq$ (1-$\theta) \times Opt) and (flag \neq 1)$ **do**

    $s =$ **TopkSets**$(i)(\mathcal{F}, D, k)$

    $S \leftarrow S \bigcup s$

    $Score = s.score$, $i \leftarrow i + 1$

end while

$PDF(S) \leftarrow$ **MaxMinFair**(S)

to express this cost that handles any arbitrary scoring function. For the purpose of illustration, let us assume that the numeric attributes are indexed using B-trees [107] and categorical attributes are indexed using inverted index [32]. For the simplicity of exposition, let us assume that $\mathcal{F}$ is monotonic and p attributes involved in scoring a record. Thus, getNext() is implemented as a series of sorted accesses over the indexes of each of the p-attributes until the next best record is determined based on $\mathcal{F}$ (using Fagin's algorithm [89] like implementation). Thus, if SA_w denotes the # SA's on attribute w in this process, the cost of getNext() is $\mathcal{O}(\Sigma_{w=1}^{p} SA_w)$.

Generate i-th best top-k set The first two operations are done inside Algorithm **TopkSets**(i), whose pseudo-code is presented in Algorithm 12. **TopkSets**(i) is responsible for generating the i-th best top-k set. For the ease of exposition, we assume there exists only one unique top-k set in each round, although ties could be handled seamlessly in the framework. Given the set $\mathcal{L}$ of sorted input lists, the algorithm sets a cursor on each list, and fetches the next record from those lists through $\mathcal{L}$ **DivGetBatch()** calls. As an example, if the input lists consist of both relevance and diversity, then **DivGetBatch()** fetches the next record from *sortedRelList* list as well as that from the *sortedDivList* list and their corresponding scores. The cursor points to the current position in the lists (let us assume that position to be j). It keeps track of the all seen records upto j-th position. Then **createNewSets** creates all possible size-k sets.

In order to accomplish (2), the other challenge involves score computations of size-k sets that are encountered so far. Since, **OptTop-k-**θ performs only sorted accesses, it may not be able to produce the exact score of a set of k records immediately - rather has to consider upper and lower bounds of score to argue if this set is a possible candidate for **TopkSets**(i). Upper bound score of a set s, $ub(s)$ (similarly lower bound score $lb(s)$) is the maximum possible (similarly the smallest) possible score s can get. Moreover, when more records are being read, these bounds are to be updated as well. Subsection 5.3.1 describes how that could be done efficiently.

Lower and upper bound score of a set. Clearly, the lower bound (upper bound) score of a set s, $lb(s)$ (similarly $ub(s)$) is the minimum (similarly maximum) possible score of s that **LowerBound** and **UpperBound** calculate. **LowerBound**(s) is calculated based on an objective function $\mathcal{F}$ and using the scores of any unseen component of $\mathcal{F}(s)$ by the smallest possible value. **UpperBound**(s) is done analogously, except the unseen component is replaced by the cursor reading at the j-th position. Lines 5-7 do that task.

Illustration using WRMSD. Imagine $\mathcal{F}$ is (weighted rel, max div). In that case $\mathcal{L}$ consists of two lists - a sorted relevance list *sortedRelList* and a sorted pairwise diversity lists *sortedDivList* in decreasing order of relevance and diversity values, respectively. Imagine the cursor is at the 2nd position of both these lists (i.e., $j = 2$)- therefore, so far it has seen $rel(r_1)$, $rel(r_2)$, $div(r_2, r_3)$, $div(r_3, r_5)$. Clearly, 4 records are seen so far, but all of their scores are not known - 4 different size-k ($k = 3$) sets could be produced. But, because of sorted access, the score of none of these sets could be calculated exactly. As an example, $ub(r_1, r_2, r_3) = 8.6 + 8.5 + 8.5 + 5 + 5 + 5$ if the weight λ is ignored. However, when the cursor reads another record, either from the relevance or from the diversity list, the ub of all sets need to be updated.

Deciding the i-th top-k set. Line 8 of **TopkSets**(i) produces and maintains a *threshold* and lines 9-12 decide if it needs to continue the computation any further or it is safe to terminate.

Definition 14. *Threshold is the maximum utility score of any unseen top-k set.* $threshold[j] = Max[ub(\mathcal{C}, i, j)]$

Given the cursor is at the j-th position of the input lists, if $threshold[j]$ falls below $Opt \times (1\text{-}\theta)$, there is no point of looking any further,**TopkSets**(i) can terminate by returning the best set present in $(\mathcal{C}, i, j)$.

Lemma 7. $s =$ ***TopkSets***(i), *if* $s = argmax(lb(\mathcal{C}, i, j))$ *and* $lb(s) \geq max(ub(\mathcal{C}, i, j) - s))$

Lines 13-17 make another key calculation based on Lemma 7. It checks if there exists a set s in $(\mathcal{C}, i, j)$ with the maximum lower bound, such that the $lb(s)$ is not smaller than the upper bound scores of all other remaining sets in $(\mathcal{C}, i, j)$. In that case, s is the i-th best set and **TopkSets**(i) terminates upon returning that set and its values. Indeed, when $\mathcal{F}$ is monotonic, no other unseen sets can have higher score than s.

Lemma 8. $s =$ ***TopkSets***(i), *if* $s = argmax(lb(\mathcal{C}, i, j))$ *and* $lb(s) \geq$ ***min***$(threshold[j]$, ***TopkSets*** $(i-1).score)$

Similarly, based on Lemma 8, the algorithm makes another important decision in Lines 22-27. If the maximum $lb(s)$ of s is not smaller than the minimum of $threshold[j]$ and the score of the top-k set seen in the $i-1$-th iteration, then $lb(s)$ is the top-k set in the i-th iteration. This lemma holds good, since the scores of the returned top-k sets decrease monotonically over iterations.

Pruning sets. Even when **TopkSets**(i) can not terminate, it checks if all sets in $(\mathcal{C}, i, j)$ are potential candidates to be the i-th best set - clearly, if the upper bound score of a set s in $(\mathcal{C}, i, j)$ is not larger than the lower bound scores of all other sets in $\mathcal{C}$, s could be pruned.

Subroutine createNewSets Given $N' < N$ number of items that are encountered by **TopkSets**(i) already, when a new item r is read through a **DivGetBatch()** call, **OptTop-k-**θ has to perform some hefty tasks.

- It needs to update $(\mathcal{C}, i, j)$ by adding additional size k sets that involve r.
- More importantly, it needs to update the lower and upper bound scores of the sets in $(\mathcal{C}, i, j)$ - or see if the score could be calculated exactly, if all required scores are read.

A naive idea is to regenerate all size $\binom{(N'+1)}{k}$ sets from scratch, which is computationally wasteful and exponential. To that end, we abstract the representation of the size k sets over a hierarchically ordered space as a lattice, and store ub and lb scores of the record sets there. This data structure offers a great benefit for doing both of these aforementioned tasks efficiently enabling incremental computation.

Data Structure. Given N' seen records, the lattice data structure maintains all $\binom{N'}{1}, \binom{N'}{2}, \ldots \binom{N'}{k}$ sets, as well as their utility score. A node in the lattice represents a possible set, singletons, pairs, triples, ..., size k sets, and so on. An edge represents the membership between two size l and $l+1$ sets. In order to solve **θ-Equiv-top-k-Sets** exactly, this space requirement is indeed the lower bound. We also note the

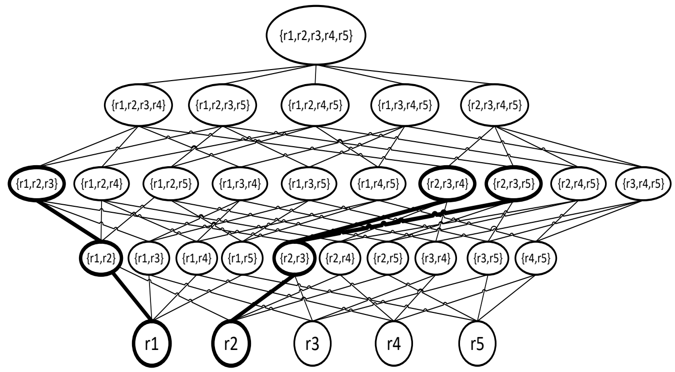


Figure 5.2 A complete lattice based on Example 5.2.1.

lattice structure could be made significantly lightweight (both computationally and storage-wise), if approximate solutions are acceptable, as we discuss in Section 5.4. A complete lattice for our running example is shown in Figure 5.2 given $N = 5$, although the data structure only stores information upto size k sets. The set $\{r_1,\ r_2,\ r_3\}$ at level three is created by union of three sets in level two, which are $\{r_1,\ r_2\}$, $\{r_1,\ r_3\}$, $\{r_2,\ r_3\}$. Hence the edges represent the connection between these sets in level l and $l + 1$.

Maintaining the structure. This data structure is updated incrementally as new records are read by **OptTop-k-θ**. Take the running example again and imagine $rel(r_1)$, and $div(r_2, r_3)$ is read. So far, the data structure have the following nodes r_1, r_2, r_3, $\{r_1, r_2\}$, $\{r_2, r_3\}$, $\{r_1, r_3\}$, and $\{r_1, r_2, r_3\}$. Next, imagine it reads $div(r_3, r_5)$, thus a new record r_5 is encountered. This creates a singleton, 3 new pairs, and 3 additional size-3 sets. Clearly, r_5 will include the following three additional size-k sets in $(\mathcal{C}, i, j)$, $\{r_1, r_2, r_5\}$, $\{r_2, r_3, r_5\}$, $\{r_1, r_3, r_5\}$.

Efficient bound computation and maintenance Imagine the cursor on the diversity list now moves to the third position and reads $div(r_1, r_3) = 4$. The upper bound scores of all of these following sets $\{r_1, r_2, r_3\}$, $\{r_1, r_2, r_5\}$, $\{r_2, r_3, r_5\}$, $\{r_1, r_3, r_5\}$ are to be updated now. One can naively calculate these bounds from the scratch - but there exists an opportunity of reusing previously done computation that is clearly more efficient.

After reading $div(r_1, r_3) = 4$, our representation updates the score of the node $\{r_1, r_3\}$ in the lattice. All nodes that have a direct or indirect edge to $\{r_1, r_3\}$, their scores are also updated.

Similar situation occurs, when a new record r is encountered - the lattice representation allows us to quickly identify the new nodes that now contains r, as well as how to efficiently reuse the previously computed score of a set s' of size smaller than k to compute score of set $\{s' \bigcup r\}$.

$$\mathcal{F}(s' \bigcup r, q) = \mathcal{F}(s', q) + \mathcal{F}(r, q) \tag{5.3}$$

Formally, our effort is to study score update as an incremental process and reuse sub-computations that are done before. We express the score (lb, ub, or exact) of a set as a summation of scores over the subsets and retrieve the previously computed scores and reuse it, as opposed to calculating the scores from scratch every time. Indeed, the lattice representation over the seen records allows us to decompose the score of a set as an aggregation over the sub-sets and reuse what has been done before.

Score reuse for WRMSD. Imagine an instance of **OptTop-k-θ** and the **DivGetBatch()** call has just returned the second row in the diversity list, namely $div(r_3, r_5) = 4$ and the goal is to produce top-k sets, where $k = 4$. A brand new record r_5 is just seen and this will add three additional size-3 sets $\{r_1, r_2, r_5\}$, $\{r_2, r_3, r_5\}$, $\{r_1, r_3, r_5\}$, three size-2 sets $\{r_1, r_5\}$, $\{r_2, r_5\}$, $\{r_3, r_5\}$, and one singleton r_5 on the lattice. The lattice structure facilitates score calculation of $WRMSD(\{r_1, r_2, r_3, r_5\})$ by reusing the scores that are calculated before. For the purpose of illustration, lets just consider the diversity component of the WRMSD calculation $WRMSD - Div(\{r_1, r_2, r_3, r_5\})$ and see how upper bound of scores could be calculated incrementally.

$$\begin{aligned} ub - div(\{r_1, r_2, r_3, r_5\}) = & Maxdiv[(r_1, \{r_2, r_3, r_5\})] \\ & + Maxdiv[(r_2, \{r_1, r_3, r_5\})] \\ & + Maxdiv[(r_3, \{r_1, r_2, r_5\})] \\ & + Maxdiv[(r_5, \{r_1, r_2, r_3\})]. \end{aligned}$$

Now consider $Maxdiv[(r_3, \{r_1, r_2, r_5\})]$ and note that this could simply be expressed as follows:

$$Maxdiv[(r_3, \{r_1, r_2, r_5\})] = Max(div(r_3, r_5), Maxdiv[(r_3, \{r_1, r_2\})] \tag{5.4}$$

$Maxdiv[(r_3, \{r_1, r_2\})]$ is pre-calculated, hence Equation (5.4) could be efficiently computed by taking a maximum over $Maxdiv[(r_3, \{r_1, r_2\})]$ score and $div(r_3, r_5)$. This allows sharing computation across sets.

Global stopping **OptTop-k-θ** halts when all θ-equivalent top-k sets are produced. This is checked by when one of the following two conditions is satisfied; (i). the last score received from **TopkSets**(i) is smaller than $(1 - \theta) \times Opt$, or (ii). the latest threshold fell below $(1 - \theta) \times Opt$ (which sets a flag to 1). It is guaranteed that there is no future unseen sets with score at most θ% smaller than the best top-k sets. At that point, **OptTop-k-θ** safely terminates and produces the exact solution.

Theorem 18. ***OptTop-k-θ*** *is an exact solution for θ**-Equiv-top-k-Sets** .*

Proof. (sketch). Given a monotonic scoring function, it is easy to see that **TopkSets**(i) produces the i-th best top-k set in the i-th iteration. **OptTop-k-θ** maintains all records across iteration, forms all potential top-k sets. Finally, when **OptTop-k-θ** terminates, the global stopping condition guarantees that no unseen set of k records will be θ-equivalent of the top-k set. Hence the proof. □

Running time of OptTop-k-θ. In Section 5.2, we prove that the counting problem involved in **θ-Equiv-top-k-Sets** is #P-hard. In reality, the running time is dominated by the number of records **OptTop-k-θ** reads before termination and is dominated by the factor $\binom{\#\text{ seen records}}{k}$, which is purely instance dependent. It could be proved that **OptTop-k-θ** is instance optimal.

5.3.2 Algorithm for MaxMinFair

The last line of Algorithm 11 calls Algorithm **MaxMinFair**, which maximizes the minimum selection probability of the records present in S. We propose a linear programming based optimum solution **Opt-SP** that takes the set of sets S as input, and produces $PDF(S)$, such that **MaxMinFair** optimizes. The problem is formally defined as,

$$\text{Maximize: } x$$

$$\text{subject to:}$$

$$\mathcal{P}(r_i) = \sum_{\forall r_i \in s, s \in S} P(s)$$

$$\mathcal{P}(r_i) \geq x, r_i \in s, s \in S$$

$$\sum_{\forall s \in S} P(s) = 1$$

Given the linear objective function and constraints this could be solved using an off-the-shelf linear programming solver using Simplex or Ellipsoid method.

Running Time. Opt-SP involves solving a linear program using Simplex or Ellipsoid method. Since the feasible region of the objective function is a polytope, these algorithms take polynomial time to the input size N and $|S|$.

Running Example. Using Example 5.2.1, $PDF(S)$ is produced as follows: $P(s1) = 0, P(s2) = 0, P(s3) = 0.5, P(s4) = 0.5$, leading to selection probability of $r_3 = 1$, and the remaining all 4 records each will have 0.5 selection probability.

5.4 Approximation Algorithms

We present two approximate solutions in this section. The first one is **RWalkTop-k-θ**. To solve **θ-Equiv-top-k-Sets**, instead of designing a deterministic exact solution that could be exponential, it leverages a random walk based approach on the item lattice that is highly efficient and is backed by probability theory. To solve **MaxMinFair**, it presents a highly efficient greedy solution **Gr-SP**. **ARWalkTop-k-θ** solves both **θ-Equiv-top-k-Sets** and **MaxMinFair** at the same time through an adaptive random walk. Both **RWalkTop-k-θ**and **ARWalkTop-k-θ** make use of the lattice structure described in Section 5.3, but it is computed only partially on the fly, making it significantly lightweight computationally and storage-wise.

5.4.1 Algorithm RWalkTop-k-θ

Algorithm 13 leverages probabilistic computation for producing **θ-Equiv-top-k-Sets** by making random walks on the item lattice. Following that, it solves **MaxMinFair** using a greedy technique.

Inputs to the algorithm are the query, k, objective function $\mathcal{F}$, θ, and the items in D. Additionally, it takes the optimum top-k set and its corresponding score from **TopkSets** 1. It starts by assigning each record a uniform probability of $1/N$. At each step it does uniform random sampling without replacement to select a record and repeats the process until a set has k records. This completes a single random walk on the item lattice, where the walk consists of the edges that are traversed. After it retrieves a size k set s, it computes $\mathcal{F}(s,q)$ and retains s, if $\mathcal{F}(s,q) \geq Opt - \theta$. It keeps repeating the process and stops when each retained s is visited atleast twice in the process.

Termination Condition of the Random Walk The termination condition used for random walk is inspired by the Good Turing Test that is often used in population studies to determine the number of unique species in a large unknown population [100]. Consider a large population of individuals drawn from an unknown number of species with diverse frequencies, including a few common species, some with intermediate frequencies, and many rare species. Let us draw a random sample of N_{samp} individuals from this population, which results in $n1$ individuals that are the lone representatives of their species, and the remaining individuals belong to species that contain multiple representatives in the sample population. Then, $P0$, which represents the frequency of all unseen species in the original population can be estimated as follows:

Lemma 9. *(Good Turing Test).* $P0 = n1/N_{samp}$.

The assumption here is that the overall probability of hitting one rare species is high while the probability of hitting the same rare species is low. Therefore, the more the sample hits the rare species multiple times, the less likely there are unseen species in the original population. We apply Lemma 9 to the θ-equivalent top-k sets construction, where a valid θ-equivalent top-k sets maps to the species and the probabilities of finding each such set in **RWalkTop-k-**θ are the frequencies. The set of θ-equivalent top-k sets discovered during **RWalkTop-k-**θ is the sample population. By ensuring this process visits each constructed set at least twice, we are essentially ensuring that n_1 is 0. Thus, using Lemma 9, P_0 can be estimated to be 0, which means it is highly likely that all θ-equivalent top-k sets are discovered.

Illustrative Example. Figure 5.2 shows the complete lattice involving Example 5.2.1. To solve θ**-Equiv-top-**k**-Sets**, the algorithm uniform randomly adds a record and continues the process until a size-3 is obtained. This way the set $s1$:$\{r_1, r_2, r_3\}$ is formed. If $s1$ is a valid answer, it is retained. The process continues until all valid sets are discovered at least twice.

Subroutine Gr-SP Subroutine **Gr-SP** is designed by leveraging the following lemma.

Lemma 10. *If every record r in S appears in only one set $s \in S$, the $PDF(S)$ is a uniform distribution that guarantees equal selection probability of the records.*

Proof. Lemma 10 demonstrates an ideal scenario, where a record $r \in s$, $s \in S$ appears in only one s. If the $PDF(S)$ is a uniform distribution, that is, $P(s) = 1/|S|, \forall s \in S$, by leveraging the definition of selection probability of a record (Definition 13), then, $\mathcal{P}(r) = 1/|S|$. Clearly, this guarantees that each records r to have the same selection probability. $\square$

Basically, the greedy algorithm is iterative and attempts to select a subset of sets from S that contains different records. Those subset of sets become part of O and gets a non-zero probability value. Specifically, It selects a set s from S in each iteration and adds to O, which includes the highest number of records that are not yet present in O but present in S. The process terminates when O contains all records in S. After that, each set that is present in O gets uniform probability of $\frac{1}{|O|}$. Any set $s \in \{S - O\}$, gets probability 0. We conjecture that this simple yet highly efficient algorithm accepts a 2-approximation factor, the formal proof is left to be explored in the future.

Illustrative Example. Imagine S contains the following 5 sets ($k = 2$), $s1$: $\{r_1, r_2\}$, $s2$: $\{r_3, r_4\}$, $s3$: $\{r_1, r_5\}$, $s4$: $\{r_3, r_5\}$, $s5$: $\{r_1, r_3\}$. If **Gr-SP** first adds $s1$ to O, then, in the next iteration it will add $s2$, and finally $s3/s4$. One possible solution will be $O = \{s1, s2, s3\}$. Each of these sets will get a probability of $1/3$ and the remaining two sets will have probability 0. The minimum selection probability of the records will be $1/3$.

Running time. With an appropriate data structure, such as bucket queue, **Gr-SP** takes $\mathcal{O}(N \times |S|)$ to run.

5.4.2 Algorithm ARWalkTop-k-θ

The last algorithm **ARWalkTop-k-θ** we discuss does not separately compute **θ-Equiv-top-k-Sets**, and then, **MaxMinFair** - instead, solves these two problems together. It makes use of Lemma 10 to design an adaptive random walk.

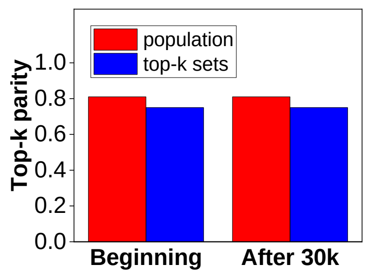
The adaptive random walk based algorithm **ARWalkTop-k-θ** is similar to the random walk part of **RWalkTop-k-θ**, except it performs the random walk adaptively, by lowering the probability of the records that are already part of some valid s, and boosting the probability of the remaining records that have not been part of any valid s yet. The goal is to discover θ-equivalent top-k sets where the same record r repeats as few times as possible across the sets - ideally appears in one and only one s. The stopping condition is still guided by the Good Turing Test as described above. Once the process terminates, each set s in S gets uniform probability, and accordingly the selection probability of the records are calculated.

For each record $r \in N$, the algorithm keeps track of the sets in S that contain r ($r.seenCnt$). Instead of picking a record uniformly at random, it then, selects r with a probability that is inversely proportional to $r.seenCnt$. The intuition is that if a record r has already appeared in many $s \in S$, picking it again will hurt the minimum selection probability of other records r' that did not appear as frequently. Therefore, in the i-th iteration of the random walk, it is likely to discover a set of k records that contains new records that are not present in S yet.

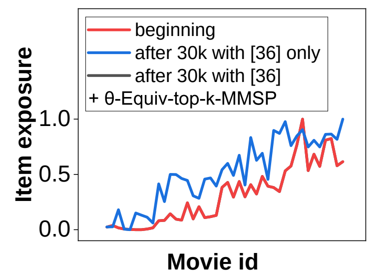
Illustrative Example. Imagine Example 5.2.1 again and assume that $s1$: $\{r_1, r_2, r_3\}$ is discovered. After that, the $r_1.seenCnt$, $r_2.seenCnt$, $r_3.seenCnt$ are increased to 1, and the probabilities of these records are readjusted proportional to their $1/r.seenCnt$. Consequently r_1, r_2, r_3 now have smaller probabilities, whereas, r_4, r_5 have higher probability. Then the random walk is repeated again and the process terminates based on the Good Turing Test. Once S is obtained, each $s \in S$ is assigned uniform probability to produce $PDF(S)$.

Table 5.5 Dataset Statistics

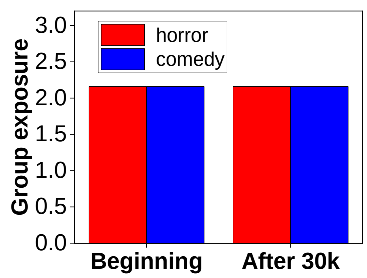
Dataset	Size	Used Attributes
Yelp	112,686	latitude, longitude, review count
IMDB-top 1000	1,000	numVotes, genre,rating
IMDB	10,000	numVotes
Airbnb	39,882	price
Synthetic	10,000	power law distribution
Makeblobs	1,000,000	random samples from Gaussian distribution



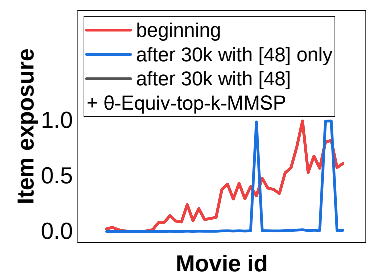
(a) **Top-k statistical parity [134]**



(b) **Individual exposure**



(c) **Group exposure [178]**



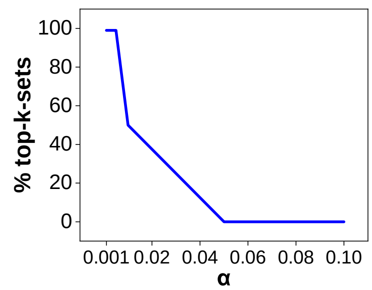
(d) **Individual exposure**

Figure 5.3 Comparison of **θ-Equiv-top-k-MMSP** with Group Fairness Models [134], [178].

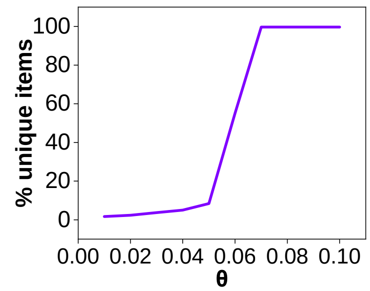
5.5 Experimental Evaluations

Our experimental evaluations have five primary goals.

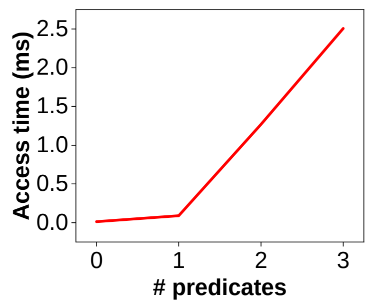
Goal (1) (Subsection 5.5.1). We compare **θ-Equiv-top-k-MMSP** with two recent related works on group fairness. Related work [134] studies top-k statistical parity, whereas, [178] studies exposure based group fairness. These works are applicable when the produced output is a rank. We study when **θ-Equiv-top-k-MMSP** is integrated inside these related works [134]m [178], how they promote both equal exposure and the respective group fairness criteria.



(a) Success rate of forming **θ-Equiv-top-k-Sets** varying α



(b) % of unique records varying θ (IMDB-top 1000)



(c) Sorted access cost varying # query predicates (IMDB-top 1000)

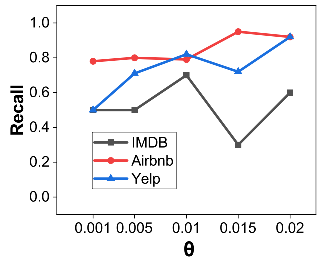
Figure 5.4 Impact of data, problem parameter and cost of sorted access.

Goal (2) (Subsection 5.5.2). Examine the impact of data and problem parameter θ. First, we analyze the impact of data distribution on the success rate of forming top-k sets. Then, we analyze the impact of θ on the probability of selecting the long tail.

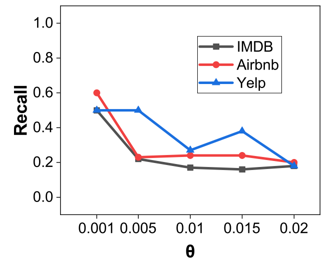
Goal (3)(Subsection 5.5.3). Examine cost of sorted access that **OptTop-k-θ** requires by varying # query predicates.

Goal (4) (Subsection 5.5.4). Examine quality of the approximate algorithms. For **θ-Equiv-top-k-Sets** , we present recall [112] of the efficient alternatives **RWalkTop-k-θ** and **ARWalkTop-k-θ** compared to **OptTop-k-θ**. For **MaxMinFair**, we present approximation factors (objective function of approximate solution/ objective function of exact solution) of **Gr-SP** and **H-SP** wrt **Opt-SP**.

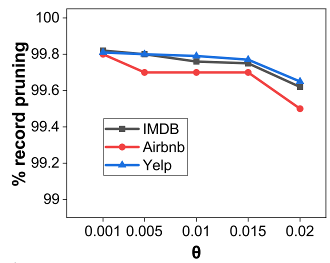
Goal (5) (Subsection 5.5.5). Investigate scalability. For **θ-Equiv-top-k-Sets**, we



(a) **RWalkTop-k-θ** recall varying θ



(b) **ARWalkTop-k-θ** recall varying θ



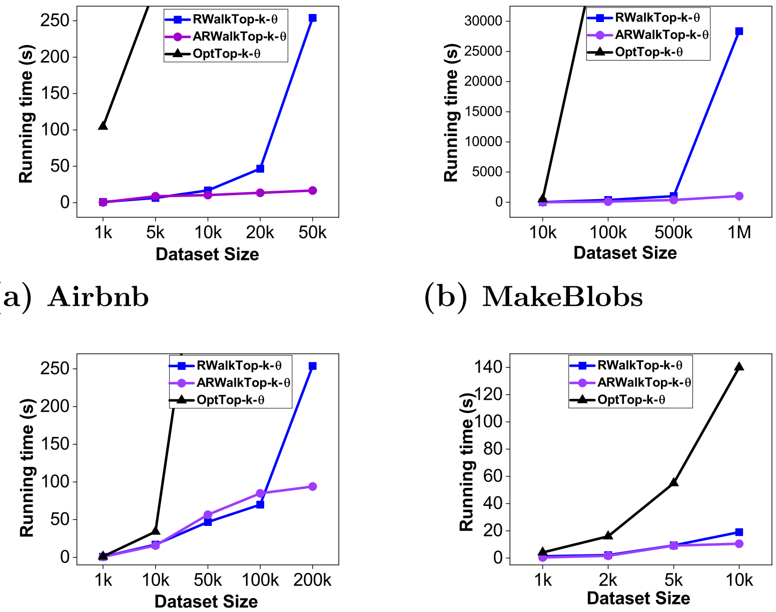
(c) Record pruning of **OptTop-k-θ**

Figure 5.5 Recall and record pruning percentage.

present pruning capabilities of **OptTop-k-θ** , as well as study the scalability of the different algorithms designed for **θ-Equiv-top-k-Sets** and **MaxMinFair** .

1. Experimental setup. All algorithms are implemented in Python 3.8. All experiments are conducted on a server machine with 128GB RAM memory, OS: windows server 2019 datacenter, version: 1809, CPU: Processor 11th Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz, 3504 Mhz, 8 Core(s), 16 Logical Processor(s). Obtained results are the average of three separate runs. Github has further details [120].

2. Datasets. Experiments are conducted on fix datasets, four real and two synthetic datasets. For real datasets, we use **Yelp** [209], **IMDB-top 1000** [124], **IMDB** [119], and **Airbnb** [10]. Our first synthetic dataset is **MakeBlobs** [144] from the sklearn package that produces data points from a normal distribution, the other dataset is synthetically generated using power law distribution. Table 5.5 has an overview.



(a) **Airbnb** (b) **MakeBlobs**

(c) **Yelp** (d) **IMDB**

Figure 5.6 **RWalkTop-k-θ** vs **ARWalkTop-k-θ** vs **OptTop-k-θ** scalability by varying dataset size N.

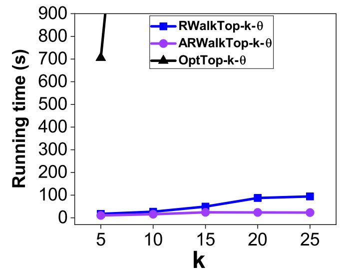
3. Implemented Algorithms.

We note that existing works [19, 94, 101] do not have an easy extension to solve **θ-Equiv-top-k-MMSP** because the solution frameworks do not adapt to solve **θ-Equiv-top-k-Sets**.

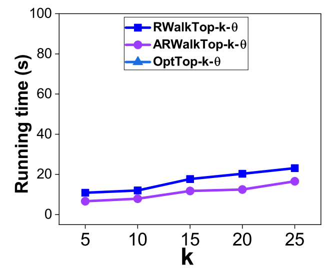
- **θ-Equiv-top-k-Sets.** We compare the exact algorithm **OptTop-k-θ** with the two approximate solutions **RWalkTop-k-θ** and **ARWalkTop-k-θ**.
- **MaxMinFair.** We implement a simple baseline **H-SP** first. It goes over the sets in S one by one and checks if all records in a set s are present in other sets in $\{S - s\}$. If yes, s is deleted from S. After that, the remaining sets are returned, each associated with uniform probability. We compare the LP-based exact solutions **Opt-SP**, with approximate solutions **Gr-SP** and **H-SP**.
- **Group fairness.** Two representative related works [134, 178] on group fairness are implemented.

4. Representative utility functions.

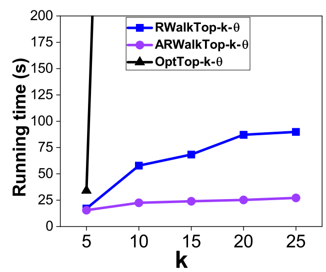
1. Maximize relevance. $\Sigma_{\forall r \in s} Rel(r, q)$



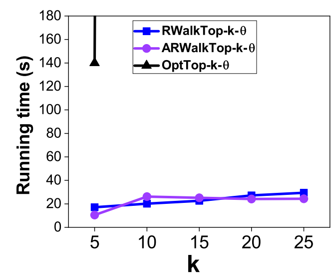
(a) **Airbnb**



(b) **MakeBlobs**



(c) **Yelp**



(d) **IMDB**

Figure 5.7 RWalkTop-k-θ vs **ARWalkTop-k-θ** vs **OptTop-k-θ** scalability by varying k.

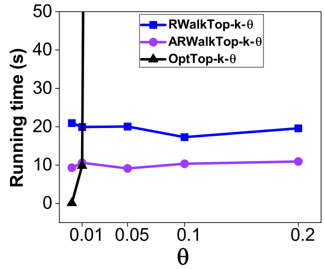
2. Weighted relevance and max sum diversity (WRMSD)
   Maximize $\lambda \times \Sigma_{r \in s} Rel(r, q) + (1 - \lambda) \times \Sigma_{r \in s} Max_{r, r_j \in \{s-r\}} Div(r, r_j)$, where λ is a weight between $[0, 1]$.

3. Maximize diversity. Maximize $\Sigma_{r \in s} Max_{r, r_j \in \{s-r\}} Div(r, r_j)$

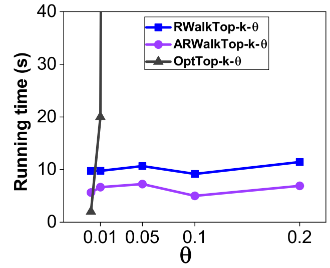
5. Query & Parameters. Queries are selected randomly. Unless specified, the default parameters are $N = 10k$, $k = 5$, $\mathcal{F}$ = WRMSD with $\lambda = 0.99$, $\theta = 0.01$.

5.5.1 Goal 1: Comparison with group fairness

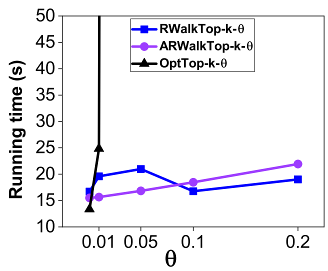
Results in Figure 5.3 present comparison between **θ-Equiv-top-k-MMSP** with two existing works on group fairness - top-k parity [134] and exposure-based group fairness [178] using IMDB-1000 movie dataset. These results empirically demonstrate five things. i. group fairness at the beginning using respective group fairness measure. ii. group fairness using respective group fairness measures + **θ-Equiv-top-k-MMSP** after the query is executed $30k$ times. iii. individual exposure at the beginning. iv. individual exposure after $30k$ if only the respective group fairness criteria is



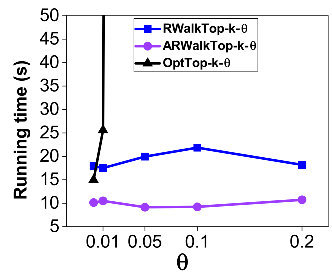
(a) **Airbnb**



(b) **MakeBlobs**



(c) **Yelp**



(d) **IMDB**

Figure 5.8 RWalkTop-k-θ vs **ARWalkTop-k-θ** vs **OptTop-k-θ** scalability by varying θ.

considered . v. individual exposure after $30k$ if the respective group fairness criteria + **θ-Equiv-top-k-MMSP** is considered. Individual exposure of an item is measured as how many times it is present in top-k and it is max-min normalized. Group fairness is imposed using genre (comedy and horror) attribute. The goal of these experiments is to demonstrate that when the query is returned multiple times (in this case to $30k$ different users), the related works preserve their respective group fairness constraints, but heavily compromise individual exposures. However, when the respective related works + **θ-Equiv-top-k-MMSP** is implemented, both group fairness and individual exposures are satisfied consistently throughout. Figure 5.3a shows top-k parity [134] at the beginning and after $30k$. The red bar represents parity value for the entire population, and the blue represents parity for top-k sets. Figure 5.3c presents similar results for group exposure [178] which imposes equal exposure for each group. Notice, both of these group fairness criteria remain unchanged at the beginning and after $30k$. Figures 5.3b and 5.3d, on the other hand, compare the individual exposure of the items at the beginning (red line), individual exposure after $30k$ iterations satisfying respective group fairness criteria only (blue line), and individual exposure after $30k$ iterations satisfying respective group fairness criteria + **θ-Equiv-top-k-MMSP** (black line). From both of these figures, it is evident that the exposure of the individual records remain unchanged when **θ-Equiv-top-k-MMSP** is combined with the the specific group fairness criteria (because black lines completely overlap with red lines), whereas, records get inequitable exposure when only the respective group-fairness is considered (red and blue lines do not overlap). Indeed, **θ-Equiv-top-k-MMSP** , when integrated inside existing group fairness models, ensures both fair group fairness and equal item exposure.

5.5.2 Goal 2: Impact of data distribution and θ

Figure 5.4a shows the success rate of forming **θ-Equiv-top-k-Sets** varying data distribution. We synthetically generate long tail data using function $x^{-\alpha}$ satisfying power law, and vary $\alpha < 1$, which is the length of the tail. When $\alpha < 1$, with increasing α, the length of the tail decreases; the success rate of forming top-k sets also decreases.

Figure 5.4b shows the impact of θ on percentage of unique records present in **θ-Equiv-top-k-Sets** using IMDB-1000 dataset. When $\theta = 0.07$, i.e., it is allowed to tolerate only 7% smaller score than that of the top-1 set, **θ-Equiv-top-k-Sets** contains the entire long tail. This is a key observation, as these results demonstrate that **θ-Equiv-top-k-MMSP** retrieves and enables equal exposure of all the movies on the long tail, which will not happen otherwise.

5.5.3 Goal 3: Cost of sorted access (SA)

Figure 5.4c shows SA time varying #query predicates. B-tree [107] and and inverted indexes are created on numerical (e.g.,ratings) and categorical attributes (e.g., genre) [32], respectively. The SA cost increases with increasing number of predicates, as expected, and how it increases depends on the score distribution over those predicates, but it always takes a few milliseconds.

5.5.4 Goal 4: Quality analysis

We first present the quality study related to the algorithms designed for **θ-Equiv-top-k-Sets** , following which, we present those results for the algorithms designed for **MaxMinFair** .

Quality Analysis of θ-Equiv-top-k-Sets We study the algorithms designed for **θ-Equiv-top-k-Sets** from the quality standpoint. We present the *Recall* percentage [112], which is the percentage of equivalent top-k sets returned by the underlying algorithm w.r.t. the exact solution **OptTop-k-θ**(ground truth).

A. Recall of RWalkTop-k-θ . We measure the quality of **RWalkTop-k-θ** by presenting the *Recall* value as described above, which produces the ratio of the top-k sets returned by **RWalkTop-k-θ** compared to that of the exact solution **OptTop-k-θ** by varying θ. Figure 5.5a shows that the recall of **RWalkTop-k-θ** stays steady mostly (close to 80% for almost all real datasets) or increases with increasing θ. At some point it becomes as high as 91%. When data distribution is uniform (synthetic data), clearly **RWalkTop-k-θ** becomes more effective with increasing θ, which is unsurprising. These results also validate the applicability of the Good Turing Test for our studied problem, informing that for almost all of the datasets, the random walk is returning around 80% of the θ equivalent top-k sets, while being significantly computationally efficient.

B. Recall of ARWalkTop-k-θ. Figure 5.5b shows the *Recall* value for the **ARWalkTop-k-θ** algorithm. As expected, **ARWalkTop-k-θ** is inferior to solve **θ-Equiv-top-k-Sets** compared to **RWalkTop-k-θ** , as it only produces sets that are highly different from each other, giving rise to fewer number of sets.

ARWalkTop-k-θ reaches up to 60% recall for Airbnb dataset. *Recall* decreases with increasing θ here, since more top-k sets with common items become eligible with increasing θ, which **ARWalkTop-k-θ** does not return.

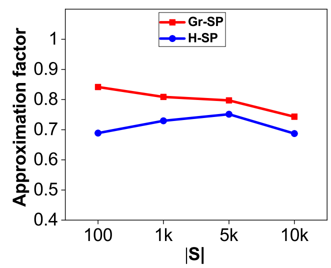
Quality analysis of MaxMinFair **A. Approximation Factor.** We calculate the approximation factor by dividing the minimum selection probability of the records returned by **Gr-SP** with that of **Opt-SP**. Since **MaxMinFair** is a maximization problem, hence the approximation factor is always ≤ 1. Similarly, the approximation factor of **H-SP** is also computed. As we shall demonstrate in Subsection 5.5.5, despite being an exact solution, **Opt-SP** is not highly scalable, since it involves a linear program. Figure 5.9 (a) shows the approximation factor using the sets returned by **RWalkTop-k-θ** algorithm for **Gr-SP** and **H-SP**. Since minimum selection probability for **Gr-SP** is higher than **H-SP**, its approximation factor is larger. The approximation factors demonstrate an encouraging facts. the minimum approximation factor value for **Gr-SP** is 0.74 and that of **H-SP** is 0.68, where as the maximum is 0.84 and 0.75, respectively. Figure 5.9 (b) present the approximation factor by varying k on 1000 sets returned by **RWalkTop-k-θ** algorithm for **Gr-SP** and **H-SP**. The minimum value of approximation factor of **Gr-SP** is 0.77, and for **H-SP** is 0.60, and the maximum values are 0.81 and 0.74, respectively.

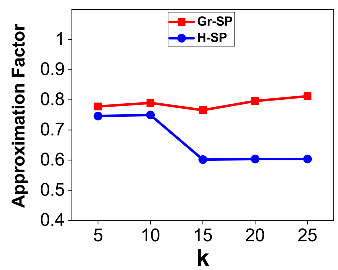
5.5.5 Goal 5: Scalability analysis

We first present the scalability study related to the algorithms designed for **θ-Equiv-top-k-Sets** , following which, we present those results for the algorithms designed for **MaxMinFair** .

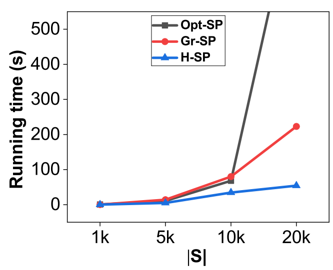
Scalability Analysis for θ-Equiv-top-k-Sets We compare the scalability aspects of the three designed algorithms by varying pertinent parameters.

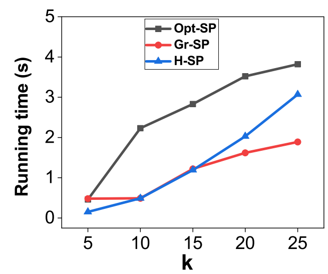
A. Pruning Effectiveness. We show that **OptTop-k-θ** solves **θ-Equiv-top-k-Sets** by accessing a very few records in the sorted lists. Figure 5.5c shows effective





(**a**) Approx factor varying $|S|$ (**b**) Approx factor varying k





(**c**) Scalability by varying $|S|$ (**d**) Scalability by varying k

Figure 5.9 MaxMinFair approx factor and scalability.

record pruning of **OptTop-k-θ** varying θ. Record pruning percentage is $= \dfrac{(N - \text{number of seen records})}{N}$. **OptTop-k-$\theta$** is able to prune 99% of the dataset to exactly solve **θ-Equiv-top-k-Sets**. Also with θ, more equivalent sets are to be found, **OptTop-k-θ** needs to read more records, thereby pruning percentage slightly decreased by increasing θ.

B. Running time varying N. Figure 5.6 shows the scalability of the three proposed algorithms for **θ-Equiv-top-k-Sets** by increasing N. As expected, due to the exponential nature of **θ-Equiv-top-k-Sets** , **OptTop-k-θ** is not scalable over large value of N. In contrast, the other two proposed algorithms are scalable. **ARWalkTop-k-θ** is more scalable than **RWalkTop-k-θ** since it finds less number of sets because of its adaptiveness, it stops earlier. With 1M records in MakeBlobs, **ARWalkTop-k-θ** takes only a few minutes to finish.

C. Running time varying k. Figure 5.7 demonstrates the scalability of the three proposed algorithms by varying k. As expected, **OptTop-k-θ** does not scale well.

Consider Figure 5.7(c) using Yelp dataset. When $k = 5$, **OptTop-k-θ** takes 34.02 seconds to run, and the number of seen records is 28. $\binom{28}{5} = 98280$ sets are generated and examined only to produce 12 final top-k sets. Now consider that it is increased to $k = 10$. This may end up producing $\binom{28}{10} = 13123110$ sets even with only 28 seen records, which is 133× larger than before. This exponential increase is expected due to the computational nature of **θ-Equiv-top-k-Sets** . On the other hand, **RWalkTop-k-θ** and **ARWalkTop-k-θ** are highly scalable, and not very sensitive to increasing k.

D. Running time varying θ. Figure 5.8 demonstrates the scalability of the three proposed algorithms by varying θ. Increasing θ increases the size of $|S|$. As expected, **OptTop-k-θ** is highly sensitive to this parameter and does not scale well. In comparison, the random walk based algorithms **RWalkTop-k-θ** and **ARWalkTop-k-θ** are less sensitive and scale reasonably well with increasing θ.

E. Running time varying $\mathcal{F}$. We measure the running time of **RWalkTop-k-θ** and **ARWalkTop-k-θ** using three representative utility functions, described in Section 5.5, by varying parameters N, θ and k. The figures are excluded for space restriction. We observe that the nature of the underlying objective function does not as such impact the running time.

Scalability Analysis of MaxMinFair In this section, we present the scalability analysis of the three algorithms designed for **MaxMinFair**. We evaluate the scalability varying $|S|$, N, k.

A. Running time varying $|S|$. Figure 5.9 (c) shows running time of the **Opt-SP**, **Gr-SP**, **H-SP** with $k = 5$. The heuristic **H-SP** exhibits the highest scalability among all and the linear programming based exact algorithm **Opt-SP** has the least scalability, as expected. Similar observation holds when N is varied.

Nevertheless, both **Gr-SP** and **H-SP** are highly scalable and the results corroborate their theoretical running time.

B. Running time varying k. Figure 5.9 (d) shows the scalability with varying k and $|S| = 1000$. Similar observation holds as before that agorithms **Gr-SP** and **H-SP** are highly scalable to increasing k. This observation is also consistent to their theoretical analysis.

5.5.6 Summary of results

(a) Our first observation is **θ-Equiv-top-k-MMSP** alleviates a fairness limitation inherent to existing group fairness models, such as, top-k statistical parity [134] and exposure based group fairness [178]. (b) Consistent with our motivation, we empirically demonstrate that with a very small θ **θ-Equiv-top-k-MMSP** allows exposure of most of the records in long tail data. (c) Our third observation demonstrates the computational effectiveness of **OptTop-k-θ** - despite the fact **θ-Equiv-top-k-MMSP** is computationally intractable, our designed solution **OptTop-k-θ** is highly effective in pruning the vast majority of the records from the input database to produce the exact solution for **θ-Equiv-top-k-Sets**. The pruning effectiveness is at times as high as 99%. (d) We experimentally observe that **RWalkTop-k-θ** is a highly scalable algorithm that is several order of magnitude faster than the exact solutions **OptTop-k-θ** and **Opt-SP**, yet the produced results are highly comparable qualitatively. This solution achieves high recall, sometime, as high as 91% recall value, while taking a few seconds to run. These results demonstrate the efficiency as well as effectiveness of **RWalkTop-k-θ** to be used and deployed inside real world applications. (e) Our final observation is that **ARWalkTop-k-θ** is a highly efficient solution that can easily scale to a very large N with millions of records, and is suitable for applications that can accommodate modest inaccuracy.

5.6 Related Work

Group Fairness. Most approaches to algorithmic fairness interpret fairness as lack of discrimination [99] seeking *that an algorithm should not discriminate against its input entities based on attributes that are not relevant to the task at hand.* Such attributes are called protected, or sensitive, and often include among others gender, religion, age, sexual orientation and race. Demographic parity is a classical group fairness notion originally studied in the machine learning literature [81, 212, 216] to ensure that the designed classification models give rise to similar false positive and false negative rates across different protected attribute groups.

W.r.t ranking and top-k results, the algorithmic fairness literature deals with group fairness along the lines of demographic parity this is typically expressed by means of some fairness constraint requiring that the $top - k$ results (for any k) to contain enough records from some groups that are protected [18, 94, 104, 123, 139, 163, 175, 178, 197, 207, 214, 217]. Two representative surveys on ranking and recommendation are [162, 215].

In [178], authors propose a framework that allows the formulation of fairness constraints on rankings in terms of exposure allocation. The optimization problem is formalized to maximize utility of the returned items given a probabilistic ranking function subject to equal exposure of different protected attribute groups. The proposed approach does not lend itself to satisfy individual exposure of items, unless $k = 2$. [178] is implemented in Section 5.5.

Individual Fairness. Individual fairness, on the other hand, as proposed by Dwork et al [81], intends to ensure "similar individuals are treated similarly". Dwork et al. explain that a classifier is individually fair if the distance between probability distributions mapped by the classifier is not greater than the actual distance between the records [81]. Biega et al. propose measures that identify unfairness at the level of individual subjects considering position bias in ranking [38]. Mahabadi et al. study the individual fairness in k-clustering. Their goal is to develop a clustering algorithm of the records so that all records are treated (approximately) equally [143]. Patro et al. [159] investigate the fair allocation problem and study individual fairness in two-sided platforms consisting of producers and customers on opposite sides. Fish et al. study individual fairness in social network [92] to maximize the minimum probability of receiving the information for poorly connected users. In the context of item-fairness in recommender systems [133], the authors define item-fairness requirement as that the coverage of all items must exceed a given threshold. Similar definition is proposed later on in [61] to ensure that items should receive the amount of exposure proportional to their relevance. In [193], the authors propose item-fairness notions that require minimum coverage for all items, so that, similar items must have similar coverage.

Authors in [38] also study exposure of individual items and intend to make that fair. This work intends to minimize the absolute difference between attention and relevance "subject to" an NDCG threshold. For this related work, the number of rankings m is an input parameter, whereas, the number of Equivalent Top-k sets for us is "subject to" an Equivalence threshold θ wrt utility. There is no easy way to translate between their "subject to" constraints and ours. [38] is suitable to ensure positional exposure of items considering multiple queries, whereas, we study it for a single query. [38] does not study which of the m outputs should be returned to an end user (as opposed to our process of returning an answer associated with its probability). Consequently, [38] does not adapt to solve **θ-Equiv-top-k-MMSP** .
It has been recognized that group fairness alone has its deficiencies [95]. In two independent efforts, Flanigan et. al. [94] and Garcia-Soriano et. al. [101] study how to enable equitable selection probability of the records under group fairness constraints and propose maxmin-fair distributions of ranking. Zemel et al. develop a learning algorithm for fair classification that ensures both group fairness and individual

fairness [216]. [19] studies individual fairness in similarity search to ensure points within distance r from the given query have the same probability to be returned. In [65], the authors propose a new ranking function that deals with web pages with hyperlinks and alleviates their unequal exposure. Due to this specific nature, the solution does not extend to **θ-Equiv-top-k-MMSP** .

Top-k Algorithms. Given a user query, a top-k result contains k records that have the highest scores [167]. Designing effective scoring functions as well as efficient algorithms [1,2] lend to numerous applications in recommendation and search [4,50, 53,86,138,168,192,195] and is an active area of research.

These related works are defined wrt ranked order, whereas, ***θ-Equiv-top-k-MMSP*** *focuses on a set based notion (if an item is present in top-k, it has exposure, else not). Neither these problems nor their designed solutions extend to our problem.*

5.7 Conclusion

We formalize **θ-Equiv-top-k-MMSP** to redesign existing top-k algorithms for long tail data to ensure fairness. Given a query, it computes a set of top-k sets that are *equivalent* and assigns a probability distribution over these sets, such that, after many users draw a set from these sets according to its assigned probability, the selection probabilities of the records present in these sets are as uniform as possible. We present multiple algorithmic results with theoretical guarantees as well as present extensive experimental evaluation. One of the directions that we are currently exploring lies in understanding pre-processing techniques that can speed up the computation of **θ-Equiv-top-k-Sets**.

Algorithm 12 TopkSets (i)

Inputs: a set $\mathcal{L}$ of input lists, i, $\mathcal{F}$, k, **TopkSets**$(i-1).score$, θ, Opt

Outputs: $nextBest$: i-th best set

$cursor \leftarrow 0$, $seenR \leftarrow \emptyset$
for $j = cursor$ **to** $Max_{l\in\mathcal{L}}Len(l)$ **do**
  $seenR = \{seenR \bigcup \textbf{DivGetBatch()}(l_1(j)), \textbf{DivGetBatch()}(l_{|\mathcal{L}|}(j))\}$
  $(\mathcal{C}, i, j) \leftarrow$ **createNewSets**(seenR[j])
  for s **in** $(\mathcal{C}, i, j)$ **do**
    lb(s), ub(s) $\leftarrow$ **LowerBound**(s), **UpperBound**(s)
  end for
  threshold[j] $\leftarrow$ **max**(ub)
  if $threshold[j] < Opt \times (1-\theta)$ **then**
    $nextBest = argmax(\mathcal{C}, i, j)$, $flag = 1$
    return $nextBest$
  end if
  for s **in** $(\mathcal{C}, i, j)$ **do**
    if lb[s] $\geq$ **max**(ub($(\mathcal{C}, i, j) - s$)) **then**
      $nextBest \leftarrow s$
      return $nextBest$
    end if
    if ub[$(\mathcal{C}, i, j)$] ¡ **max**(lb($(\mathcal{C}, i, j) - s$)) **then**
      Prune $\{(\mathcal{C}, i, j) - s\}$
    end if
  end for
  if max(lb[$(\mathcal{C}, i, j)$ $\geq$ **min**(threshold[j], **TopkSets**$(i-1).score$ **then**
    $nextBest \leftarrow$ **argmax**(lb$(\mathcal{C}, i, j)$)
    Break
  end if
  $cursor \leftarrow j+1$
end for

Algorithm 13 RWalkTop-k-θ

Inputs: query q, D, k, $\mathcal{F}$, θ

Outputs: $PDF(S)$

while true **do**
    $s = \{\}, S = \{\}$
    while $|s| \leq k$ **do**
        pick a uniform random $r \in \{D - s\}$,
        $s \leftarrow \{s \bigcup r\}$
    end while
    if $\mathcal{F}(s, q) \geq (1 - \theta) \times Opt$ **then**
        $S \leftarrow S \bigcup \{s\}$
    end if
    $visit.s \leftarrow visit.s + 1$
    if $visit.s \geq 2, \forall s \in S$ **then**
        break
    end if
end while
$PDF(S) \leftarrow$ **Gr-SP**(S)

CHAPTER 6

ACCESS PRIMITIVE FOR TOP-K DIVERSITY COMPUTATION

6.1 Introduction

Diversity has a wide variety of applications in search, recommendation [1, 2, 86, 166, 188, 193] and data exploration. The goal of diversification algorithms is to return results that are relevant as well as cover user intent. In the data management community, returning top-k diverse results of a query has been extensively studied, and there exists many seminal works [55, 106, 211] that propose objective functions and efficient algorithms to achieve a trade-off between relevance and diversity.

The original implementation of many representative algorithms, such as, GMM [106], MMR [106], SWAP [211] that do not make any assumptions on the nature of the diversity functions are iterative in nature and make the decision of updating the top-k set by making a greedy choice based on the current top-k set and the remaining records that are not yet in top-k. These representative algorithms go through the cumbersome step of pairwise diversity computation of records between and across these two sets even to make a single update in the top-k set. Indeed, for a large database containing N records, this repetitive computation is expensive $\mathcal{O}(N)$, since typically $k << N$. We are also aware of a handful of existing works [98,156] that are specifically designed on geometric space and avoid this repetitive computation. However, to the best of our knowledge, most of the existing works assume this expensive computation to be necessary, when diversity is designed for arbitrary non-metric functions or even studied in general metric space. Contrarily, our effort here is to reduce that computation without making any explicit assumptions about the diversity function, that is, considering diversity functions to be fully arbitrary or even non-metric.

Our first contribution lies in identifying one major computational bottleneck in existing popular diversification algorithms and how to accelerate that process. We identify the basic ingredients of developing **DivGetBatch()** as an access primitive such that it remains agnostic to any specific underlying diversity or distance computation function. This primitive is also guaranteed to produce identical top-k results as of the original diversity algorithms. The fundamental idea is to make the comparison go over a group of records, as opposed to record pairs, thereby accelerating the computation. In other words, the large number of N records are to be grouped in a small number of C nodes and some higher level diversity aggregates are to be maintained between the nodes. Towards that, we develop a generic computation framework that builds an index **I-tree** offline and maintains two other auxiliary data-structures (*MinsimMatrixNode* and *MaxsimMatrixNode*) that are highly generic in nature and suitable to handle updates. Indeed, the design of **I-tree** is rather simple and may appear to share resemblance with existing indexing techniques (Section 6.7 contains detailed discussion and empirical evaluation towards that). Our primary contribution lies in proposing a simple enough indexing technique that could be easily designed using off-the-shelf popular record partitioning algorithms, such as, K-Means [112], but study how to make it generic enough to work on a variety of diversification algorithms over arbitrary diversification functions. In fact, existing popular indexing techniques, such as *K-B-D-tree* [170], *kd-tree* [34], M-Tree [68], Ball-Tree [135], *R-tree* [110] assume that coordinate information of the records are available and used to create data structures to answer a large spectrum of distance queries, where distance may be based on Euclidean, cosine similarity, or general L_p norms. ***However,* I-tree *assumes the records to be atomic and the diversity function to be arbitrary.***

Our second contribution is to develop query processing algorithms for *MMR*, *GMM*, and *SWAP* [55, 106, 211] using **DivGetBatch() (Sections 6.3, 6.4, 6.5)**.

Fundamentally, we have redesigned the original algorithms to run over pairs of groups of records as opposed to pairs of records to save up processing time. **We make theoretical claims and proofs on the exactness and the running time of the augmented algorithms in expectation (assuming uniform data and query distributions) and in the worst case**. As an example, we prove that augmented $SWAP$ (**Aug-SWAP**) takes $\mathcal{O}(N/C * k * \log k + N)$ time in expectation compared to $\mathcal{O}(N * k * \log k)$ time of the original algorithm. It is easy to notice that augmented $SWAP$ is guaranteed to run faster than the original algorithm, as $Max(N/C * k * \log k, N)$ (C is the number of groups) is smaller than $N * k * \log k$. The summary of the complexity results are presented in Tables 6.1 and 6.2.

Our third contribution is developing principled solutions for creating and maintaining **I-tree (Section 6.6)**. **I-tree** is a complete m-ary tree [72] with height l. There exists many ways to build **I-tree** (e.g., hierarchical graph partitioning or clustering could be used). We identify that the main computational bottleneck of **I-tree** under batch updates lies in updating *MinsimMatrixNode* and *MaxsimMatrixNode*. Therefore, we formalize the index maintenance problem as an optimization problem, with the goal of minimizing the number of updates in these data structures. We present an integer programming-based exact solution **OPTMn** for that, and a greedy heuristic **GrMn** that is highly scalable in nature.

Our final contribution is experimental **(Section 6.7)**. We use large real-world datasets, one large publicly available synthetic dataset to show that the augmented algorithms return results identical to their originals, while ensuring between a 3× to 24× speedup on large datasets. We study the effects of different parameters empirically and provide guidance for appropriate design choice. We empirically present exhaustive results to pre-process and maintain **I-tree**. Our empirical results corroborate our theoretical analyses.

Table 6.1 Technical Results For Running Time Analysis w.r.t. $|CandR|$

Algorithm	**Variant**	**Expected time w.r.t** $\|CandR\|$
MMR	Original	$\mathcal{O}(N * k^2)$
	Augmented	$\mathcal{O}(C * k^2 + N + \sum_{i=1}^{k} \|CandR_i\| * k)$
GMM	Original	$\mathcal{O}(N * k)$
	Augmented	$\mathcal{O}(C * k + \sum_{i=1}^{k} \|CandR_i\|)$
$SWAP$	Original	$\mathcal{O}(N * k * \log k)$
	Augmented	$\mathcal{O}(N + \sum_{i=1}^{N} \frac{\|CandR_i\|}{N} * (C + k * \log k))$

Moreover, we compare the proposed index **I-tree** with a set of existing indexing structure, such as, M-Tree [68], KD-Tree [34], and Ball-Tree [135]. These latter trees are primarily designed for the Euclidean space. Our experimental results unanimously selects **I-tree** as the winner. The augmented algorithms implemented using **I-tree** is at least 18× faster in query processing and as much as 170× faster for certain configuration. **I-tree** achieves more than 1.5× speedup during the index construction and at times it is more than 20× faster w.r.t. the baselines.

To summarize, we make the following contributions:

- We develop **DivGetBatch()**, an access primitive and show how to integrate it inside popular diversity algorithms to save up running time (Sections 6.3, 6.4, 6.5). We present in depth theoretical analyses of the augmented algorithms.
- We propose a computational framework to support **DivGetBatch()**(Section 6.6. The framework consists of a pre-computed index **I-tree** and a query processing step. We also present non-trivial solutions to maintain **I-tree** under dynamic updates.
- We run an extensive experimentation that demonstrates the effectiveness of building and maintaining **I-tree** and **DivGetBatch()**, and corroborates our theoretical claims (Section 6.7).

Table 6.2 Technical Results for Running Time Analysis w.r.t. C, m, l

Algorithm	**Variant**	**Expected time w.r.t** C	**Expected time w.r.t** m **and** l
MMR	Original	$\mathcal{O}(N * k^2)$	$\mathcal{O}(N * k^2)$
	Augmented	$\mathcal{O}((N/C + C) * k^2 + N)$	$\mathcal{O}((N/m^l + m^l) * k^2 + N)$
GMM	Original	$\mathcal{O}(N * k)$	$\mathcal{O}(N * k)$
	Augmented	$\mathcal{O}(N/C + C) * k)$	$\mathcal{O}(N/m^l + m^l) * k)$
$SWAP$	Original	$\mathcal{O}(N * k * \log k)$	$\mathcal{O}(N * k * \log k)$
	Augmented	$\mathcal{O}(N/C * k * \log k + N)$	$\mathcal{O}(N/m^l * k * \log k + N)$

Index	Activity	Time	Space	Time	Space
I-tree	Construction	$\mathcal{O}(N * C^2 * t + N^2)$	$\mathcal{O}(C^2)$	$\mathcal{O}(N * m^{2l} * t + N^2)$	$\mathcal{O}(m^{2l})$
	Maintenance	$\mathcal{O}(N * \lvert Y \rvert)$	$\mathcal{O}(C^2)$	$\mathcal{O}(N * \lvert Y \rvert)$	$\mathcal{O}(m^{2l})$

6.2 Background and Approach

This section is organized in two parts. In Subsection 6.2.1, we present the background of the studied problem and define it. In Subsection 6.2.2, we present the fundamental ideas of our approach.

6.2.1 Motivating example and problem definition

The basic principle of existing diversification algorithms, such as *MMR*, *GMM*, and *SWAP* is either to incrementally build a top-k set of diverse results or to greedily replace records in a top-k list to find the most diverse ones. In both cases, the leading cost directly depends on the number of pairwise record comparisons. Imagine a toy database D containing $N = 10$ records. Since the records are considered atomic, Table 6.4 shows a record-record similarity matrix, $simMatrixRecord$, normalized between [0-1] for our example. Diversity between r_i, r_j is simply $1 - sim(r_i, r_j)$. Given a query Q, in order to produce $k = 2$ results, an algorithm such as *MMR* [55] first assigns all 10 records in D to a potential candidate set R. Then it iterates over all 10 records once to find the best record in terms of MR score (based on diversity and relevance), and adds that to the result set S and discards that from R. It repeats the same process once more to produce the resulting set $S = \{r_{10}, r_8\}$. In particular, there is a repeated pairwise computation of the following kind:

```
While  k ≤ 2 :
        rec ← R[1]
          For    i = 2; i <= |R|; i++
              if    MR(Q, R[i], S) ≥ MR(Q, rec, S)
               rec ← R[i]
          EndFor
        S ← S ⋃ rec, R ← R − rec
        k ← k + 1
EndWhile
```

Problem Definition 5. *Develop an access primitive* **DivGetBatch()** *and integrate it inside existing popular diversity algorithms.***DivGetBatch()** *satisfies the following three criteria:*

- *It guarantees identical top-k results as that of the original algorithms.*
- *It is generic, i.e., it works for any diversity functions - diversity being metric or not. A function is metric if it satisfies three properties: identity, symmetry, and triangle inequality.*
- *When integrated inside existing algorithms, it accelerates the computation and returns the results faster.*

The proposed primitive simplifies the aforementioned implementation as follows - instead of iterating over the entire R set (which is $\mathcal{O}(N)$), it returns potentially a much smaller set of records $CandR$, from which the result set S would be updated.

$\mathbf{CandR} \leftarrow \mathbf{DivGetBatch}(\mathbf{R}, \mathbf{Q}, \mathbf{S})$

While $k \leq 2:$

$rec \leftarrow Max(MR(CandR, Q, S))$

$S \leftarrow S \bigcup rec, CandR \leftarrow CandR - rec$

$k \leftarrow k + 1$

EndWhile

6.2.2 Approach

DivGetBatch() is designed by developing a computational framework, described in Figure 6.1. The basic idea is to store "higher level aggregates"', such as *minimum and maximum diversity scores of a group of records instead of keeping individual pairwise diversity scores between the records.* We formally define the minimum and maximum diversity scores as *bounds* in later sections. As an example, if the same set of records are grouped in three nodes, as shown inside the indexing box of Figure 6.1 and the maximum and minimum diversity scores are preserved between them, $node_2$ and $node_3$ can be discarded in the first iteration of processing of *MMR* pruning 6 out of the 10 records and returning only $\{r_1, r_2, r_4, r_{10}\}$ in R. This indeed leads to a significant speedup.

Offline vs. Online.

In this work, we assume that both data and query follow uniform distributions. A keen reader may notice that to accelerate diversity computation using **I-tree**, one has to "group" records and maintain some higher level aggregates between them. Grouping a large database of N records is time-consuming, as that would require partitioning them based on pairwise diversity. Indeed, this process of grouping must happen once and offline.

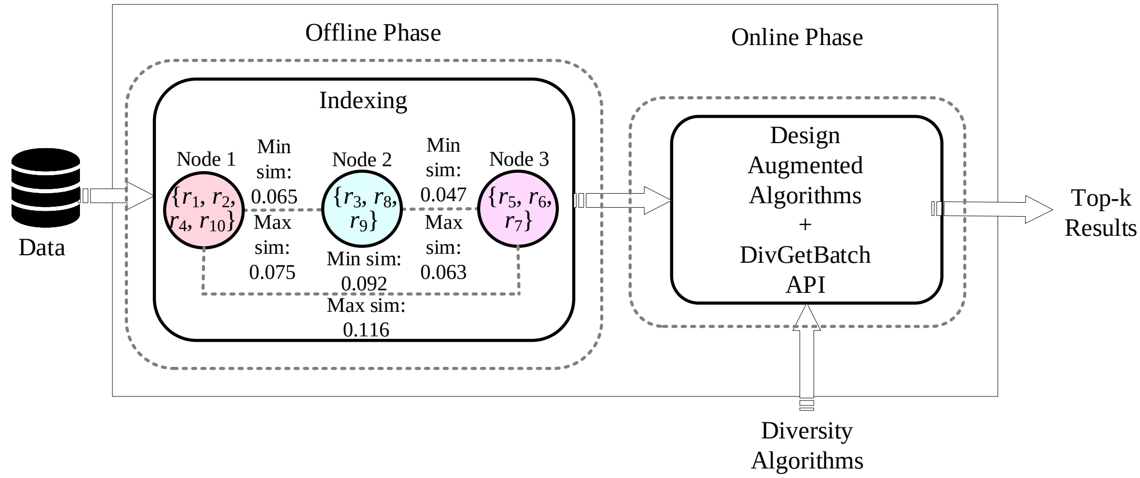


Figure 6.1 Proposed computational framework.

Precisely because of this, we resort to pre-process the records to group them and develop index **I-tree**, and use that later for processing diversity queries. This is the offline computation of the proposed framework.

Just like **DivGetBatch()**, **I-tree** is a general purpose complete tree like structure and could be designed in more than one way. It needs to satisfy three properties.

- **I-tree** has m arity and l height or levels (user inputs).
- Two highly important auxiliary data structures maintain similarity bounds between the nodes in **I-tree**: *MinsimMatrixNode* and *MaxsimMatrixNode* for maintaining minimum and maximum similarity bounds [1].
- For three nodes n, n', and n_j in **I-tree**, if n is a parent of n', and n_j is part of a different subtree and at the same level as n, the following relationship holds: **Min** $sim(n, n') \geq$ **Min** $sim(n, n_j)$, and **Max** $sim(n, n') \geq$ **Max** $sim(n, n_j)$, (basically nodes that are part of the same subtree have higher min and max similarity bounds compared to the nodes that are not).

The indexing algorithm *BuildTree* (Algorithm 18) partitions (refer to the Subroutine *Partition*) the records. It also maintains additional data structures that contain similarity scores between nodes for efficient query processing. An example

[1]Diversity between a pair of records is simply $1 - \textit{similarity}$ between them.

Table 6.3 Notations & Interpretations

Notations	
D	Database containing N records
S	Result set
Z	Set of nodes that contain S
R	Remaining records in the dataset
Q	Query
k	Number of records in resulting set
m, l	Arity & Total number of levels in the **I-tree**
C	Number of nodes in the **I-tree**
$CandR$	Candidate record set returned by API
Y	A batch of new records to be updated in **I-tree**

of a two-level index tree is shown in Figure 6.2. At the first level, *BuildTree* creates a root node containing all N records and m children of the root node. From the point of abstraction, it is not important at this stage to describe how the data is partitioned. Basically, the goal is to keep similar records together while separating non-similar ones. There are multiple off-the-shelf techniques such as clustering and graph partitioning to carry out this task.

In our implementation, we use the popular k-means algorithm [112] for partitioning. The algorithm repeats the partitioning procedure until it reaches l levels. We refer to Section 6.6 for further details.

Next, we present the generic recipe of using **DivGetBatch()** online or during the query processing time.

Generic Online Algorithm using DivGetBatch() The inputs to **DivGetBatch()** is **I-tree**, query Q, current candidate set of answers S, remaining records R, as well as the algorithm specific objective function f. The output is

Algorithm 14 Generic **DivGetBatch()** API

Inputs: **I-tree**, S, R, Q, f
Outputs: $CandR$: remaining eligible set of records for next iteration
for $y = 1$ **to** l **do**
  for n **in I-tree** $[y].nodes$ **do**
    uB, $lB \leftarrow$ **Calculate-Bounds**(**I-tree**, n, y, f, S, Q, R)
    $uBs \leftarrow \bigcup uB$, $lBs \leftarrow \bigcup lB$
  end for
  $M \leftarrow$ **Skip-Nodes**(**I-tree**, y, uBs, lBs)
  $V \leftarrow \{$ **I-tree** $[y].nodes - M\}$
end for
$CandR = \{r \mid r \in n, n \in V\}$
return $CandR$

$CandR$, a set of candidate records that cannot be eliminated and require further processing by the original algorithm. **DivGetBatch()** explores **I-tree** level by level during query time and exploits two of its higher-level constructs: a. **Calculate-Bounds:** it computes similarity bounds [2] between Q and the nodes in **I-tree** based on a specific algorithm and objective function f. In particular, it computes an upper and a lower bound of diversity scores of the node. The goal is to decide if it is beneficial to go inside the node and explore the subtree under it. b. **Skip-Nodes:** based on the previous decision, the algorithm either skips the node and its entire subtree or explores the node.

Algorithm 14 shows the pseudo-code of the **DivGetBatch()** API.

[2]Please note diversity could be easily calculated from similarity bounds.

6.3 MMR Query Processing with DivGetBatch()

The first algorithm we study is MMR [55] algorithm. We describe the original version of the algorithm and our augmented version and provide theoretical analysis on how our augmented version outperforms the original one.

6.3.1 MMR algorithm

Maximal Marginal Relevance (MMR) algorithm is a seminal work on result diversification [55]. MMR is based on Marginal Relvance (MR) score (Equation (6.1)) that it maximizes in each iteration. Given a query, MR introduces a λ coefficient to strike a balance between the relevance score, computed between the records and the query, and the diversity score, computed between the records themselves.

MMR is greedy in nature that grows the size of the top-k set by adding records one by one in the top-k set by considering the relevance of the record and diversity with the previously selected records, using the formula below:

$$MMR(r) \leftarrow argmax_{r \in R \setminus S} MR(r),$$

$$MR(r) \leftarrow \lambda sim(r, Q) - (1 - \lambda) max_{r_j \in S} sim(r, r_j), \tag{6.1}$$

where Q is the query, S is the previously selected items, R is the remaining records in the dataset, r is a candidate record from R, and r_j is another record from S. λ is a tunable parameter. The time-consuming part of the algorithm lies in computing the MR score for each $r \in \{R \setminus S\}$ and returning the one with the highest MR score.

The MMR algorithm takes $\mathcal{O}(|R| \times |S|)$, when we add one new record to set S, demonstrating that it has an order of $N \times k$. The algorithm repeats k times and produces top-k results.

Table 6.4 Similarity Matrix for Records

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	Q
r_1	1.000	0.979	0.065	0.989	0.105	0.110	0.092	0.066	0.068	0.969	0.187
r_2	0.979	1.000	0.070	0.992	0.107	0.112	0.092	0.071	0.074	0.999	0.190
r_3	0.065	0.070	1.000	0.068	0.057	0.061	0.048	0.982	0.986	0.071	0.052
r_4	0.989	0.992	0.068	1.000	0.111	0.116	0.096	0.069	0.072	0.986	0.180
r_5	0.105	0.107	0.057	0.111	1.000	0.976	0.880	0.055	0.058	0.106	0.039
r_6	0.110	0.112	0.061	0.116	0.976	1.000	0.783	0.059	0.063	0.112	0.041
r_7	0.092	0.092	0.048	0.096	0.880	0.783	1.000	0.047	0.049	0.092	0.036
r_8	0.066	0.071	0.982	0.069	0.055	0.059	0.047	1.000	0.986	0.072	0.054
r_9	0.068	0.074	0.986	0.072	0.058	0.063	0.049	0.986	1.000	0.075	0.054
r_{10}	0.969	0.999	0.071	0.986	0.106	0.112	0.092	0.072	0.075	1.000	0.191

6.3.2 Aug-MMR algorithm

Aug-MMR algorithm is designed to circumvent this aforementioned time consuming computation by leveraging **DivGetBatch()**. The general idea is to return a small subset of records, as opposed to all $|R|$ records (which is $\mathcal{O}(N)$) in each iteration, thereby saving computation. The rest of the algorithm is identical to its original version and is presented in Algorithm 15.

We now describe subroutine 15, how **DivGetBatch()** exactly works in **Aug-MMR**. Inputs to **DivGetBatch()** are **I-tree**, S, R, Q, and f (i.e., objective function of MMR). The output is $CandR$, the candidate set of records for which MR scores are to be computed to retain the best record. Based on Algorithm 14, we now describe the specifics of two higher-level constructs for **Aug-MMR**.

Calculate-Bounds: This function leverages $MinsimMatrix\text{-}Node$ and $MaxsimMatrixNode$ to calculate lower ($lBMR$) and upper bounds ($uBMR$), respectively. The bounds essentially represent the score of a node based on f (Equation (6.1)) and mathematically can be expressed as follows:

$$lBMR_{node} \leftarrow \lambda \mathbf{Min}\ sim(node, Q) - \\ max_{node' \in Z}(1-\lambda)\mathbf{Max}\ sim(node, node'), \quad (6.2)$$

Algorithm 15 Aug-MMR

Inputs: **I-tree**, D, MMR, Q, k

Outputs: S: final top-k result set.

1: $R \leftarrow D$, $S = \phi$

2: **for** $t = 1$ **to** k **do**

3: $CandR \leftarrow$ **DivGetBatch**(**I-tree**, R, S, Q, MMR)

4: $S = \{S \bigcup MMR(r)_{r \in CandR}\}$

5: **end for**

6: **return** S

$$uBMR_{node} \leftarrow \lambda \textbf{Max } sim(node, Q) - min_{node' \in Z}(1-\lambda)\textbf{Min } sim(node, node'), \quad (6.3)$$

where Z is the set of nodes that contain S,
Min $sim(node, Q)$ and **Max** $sim(node, Q)$ are the minimum and the maximum similarity between any records in *node* and Q, respectively, and **Min** $sim(node, node')$ and **Max** $sim(node, node')$ are the minimum and the maximum similarity between any two records in *node* and *node'*, respectively. Since $lBMR$ is the smallest score of *node*, it is calculated by taking the minimum of *sim* score in the first part of the equation and subtracting that from the maximum of *sim* score in the second part. Contrarily, $uBMR$ refers to the maximum MR score of *node* (Equation (6.3)) and can be calculated by reversing the min and max of the (Equation (6.2)).

Skip-Nodes: The argument of node skipping is simple - if the $uBMR$ score of a node is not larger than the $lBMR$ of another node, then the former node and its entire subtree could be pruned. The records from the remaining nodes form the

$CandR$ set.

$$CandR \leftarrow \{N - \{r \in \mathbf{I} - \mathbf{tree}.n \mid uBMR_n < \max_{\forall n'}(lBMR_{n'})\}\} \tag{6.4}$$

this is done by finding the maximum value of $lBMR_{n'}$ of all nodes and then discard ones with $uBMR$ less than it. **Running Example:** A step by step calculation of **DivGetBatch()** is shown in Table 6.5. The maximum and minimum similarity between $node_1$ and query Q is 0.180 and 0.191. In first iteration of CALCULATE-BOUNDS, lower bound of MR of $node_1$ which is $lBMR_{node_1} = 0.8 * 0.180 - (1 - 0.8) * 0 = 0.144$, and upper bound of MR of $node_1$, $uBMR_{node_1} = 0.8 * 0.191 - (1 - 0.8) * 0 = 0.153$. Similarly, $lBMR_{node_2}$, $uBMR_{node_2}$, $lBMR_{node_3}$, and $uBMR_{node_3}$ are -0.047, 0.044, 0.029, and 0.033, respectively. In SKIP-NODES, the maximum of all $lBMR$s is found 0.144 which is $lBMR_{node_1}$.

$uBMR_{node_2}$ and $uBMR_{node_3}$ are smaller than $lBMR_{node_1}$. Therefore, $node_2$ and $node_3$ are discarded from further calculation in iteration 1. Records of $node_1$ $\{r_1, r_2, r_4, r_{10}\}$ are returned by **DivGetBatch()** to **Aug-MMR** algorithm. **Aug-MMR** performs calculation similar to original MMR on $\{r_1, r_2, r_4, r_{10}\}$ which results in $S = \{r_{10}\}$. Likewise, the maximum and minimum similarity between $node_1$ and $node_1$ are 0.969 and 1.0. In the second iteration of CALCULATE-BOUNDS, $lBMR_{node_1} = 0.8 * 0.180 - (1 - 0.8) * 0.969 = -0.050$ and $uBMR_{node_1} = 0.8 * 0.191 - (1 - 0.8) * 1.0 = -0.047$. Similarity, $lBMR_{node_2}$, $uBMR_{node_2}$, $lBMR_{node_3}$, and $uBMR_{node_3}$ are 0.028, 0.029, 0.010, and 0.009, respectively. In SKIP-NODES, the maximum of all $lBMR$s is $lBMR_{node_2} = 0.028$. $uBMR_{node_1}$ and $uBMR_{node_3}$ are smaller than $lBMR_{node_2}$. Thus, $node_1$ and $node_3$ are discarded from further calculation in iteration 2. Records of $node_2$ $\{r_3, r_8, r_9\}$ are returned by **DivGetBatch()** to **Aug-MMR** algorithm. **Aug-MMR** performs calculation similar to original MMR on $\{r_3, r_8, r_9\}$ which results in $S = \{r_{10}, r_8\}$

Table 6.5 First Two Iterations of DivGetBatch() in Aug-MMR

Functions	Nodes	Bounds	Iteration 1	Iteration 2
Calculate-Bounds	$node_1$	$lBMR$	$0.8 * 0.180 - (1 - 0.8) * 0 = 0.144$	-0.050
		$uBMR$	$0.8 * 0.191 - (1 - 0.8) * 0 = 0.153$	-0.047
	$node_2$	$lBMR$	$0.8 * 0.0191 - (1 - 0.8) * 0 = 0.0152$	0.028
		$uBMR$	$0.8 * 0.054 - (1 - 0.8) * 0 = 0.044$	0.029
	$node_3$	$lBMR$	$0.8 * 0.036 - (1 - 0.8) * 0 = 0.029$	0.010
		$uBMR$	$0.8 * 0.041 - (1 - 0.8) * 0 = 0.033$	0.009
Skip-Nodes			$lBMR$ array: 0.144, 0.041, 0.029 $uBMR$ array: 0.153, 0.044, 0.033 $node_2$, $node_3$ are skipped. $CandR$= $\{r_1, r_2, r_4, r_{10}\}$. $MMR(r_1, r_2, r_4, r_{10}) \leftarrow r_{10}$ Number of records discarded is 6	$lBMR$ array: -0.050, 0.028, 0.010 $uBMR$ Array: -0.047, 0.029 , 0.009 $node_1$, $node_3$ are skipped. $CandR = \{r_3, r_8, r_9\}$ $MMR(r_3, r_8, r_9) \leftarrow r_8$ top-2 set = $\{r_{10}, r_8\}$

Aug-MMR algorithm proofs

Claim 1. Aug-MMR *returns identical top-k results as that of original MMR.*

Proof. The proof is constructed using one helper lemma and one observation: Lemma 11 proves that **DivGetBatch()** never prunes a record that is part of the original top-k; Observation 1 shows that once the control comes back from **DivGetBatch()**, **Aug-MMR** works exactly as the original MMR in each iteration. Combining these lemma and observation, **Aug-MMR** returns identical top-k results as that of the original MMR. □

Lemma 11. DivGetBatch() *never prunes a record that is part of the original top-k.*

Proof. As part of this proof, we first prove that Skip-Nodes *never discards* the record which has the highest MR score in that iteration.

Recall Property 1 and note that for every two nodes n and n' in the same subtree, if n is a parent of n', then n contains all records in n', thereby having larger

$uBMR$ and $lBMR$ values. Therefore, if a node n is skipped, any child of n is also safe to be skipped.

We use helper Lemma 12 to prove that the actual MR score of any record in a node $node$ is bounded between $uBMR_{node}$ and $lBMR_{node}$. Let us assume, the next desired record $r_d \in node_d$ produces maximum MR value among all $R \setminus S$ records. MR_{r_d} is greater than $minMR_{node}$ for $\forall node$. Using Equation (6.6):

$$\begin{aligned} MR_{r_d} &\geq max_{node \in \mathbf{I-tree}[l].nodes} minMR_{node} \\ &\geq max_{node \in \mathbf{I-tree}[l].nodes}(lBMR_{node}), \end{aligned}$$

Using Equation (6.6), $MR_{r_d} = MaxMR_{node_d} \leq$ $uBMR_{node_d}$. As a result,

$$\begin{aligned} uBMR_{node_d} &\geq MR_{r_d} \\ &\geq max_{node \in \mathbf{I-tree}[l].nodes}(lBMR_{node}). \end{aligned} \tag{6.5}$$

According to Equations (6.5) and (6.4), $node_d$ will not be discarded, and all records inside $node_d$ including r_d will be returned by **DivGetBatch()** or send to the next level for further processing. This logic extends for all the iterations. Therefore, **DivGetBatch()** never prunes a record that is part of the original top-k. $\square$

Lemma 12. *MR score of any record $r \in node$ (say MR_r) is bounded by upper and lower bound $uBMR_{node}$ and $lBMR_{node}$, respectively. That is,*

$$lBMR_{node} \leq MR_{r \in node} \leq uBMR_{node}. \tag{6.6}$$

Proof. We will first prove that maximum relevance value (say MR_{rmax}) of any record (say $r_{max} \in node$) is less than equal to $uBMR_{node}$. Where, MR_{rmax} can be expressed as:

$$MR_{rmax} = \lambda sim(r_{max}, Q) - (1 - \lambda) max_{r_j \in S} sim(r_{max}, r_j)]. \tag{6.7}$$

First part of the Equation (6.7) is always less than equals to first part of the Equation (6.3). That is:

$$\lambda sim(r_{max}, Q) \leq \lambda max_{r_i \in node} sim(r_i, Q) \\ = \lambda \textbf{Max } sim(node, Q), \tag{6.8}$$

Next, we show that second part of the Equation (6.7) is always greater than second part of the Equation (6.3).

Let us assume; $r_w \in S$ produces max value for the second part of Equation (6.7). That second part can be rewritten as $(1-\lambda)sim(r_{max_{node}}, r_w)$. Let us assume, $r_w \in node_w$ where $node_w \in Z$. For any $node' \in Z$, we can write:

$$(1-\lambda)sim(r_{max}, r_w) \geq (1-\lambda)min_{r_i \in node, r_j \in node'} \\ sim(r_i, r_j) \\ \geq min_{node' \in Z}(1-\lambda)\textbf{Min } sim(node, node'), \tag{6.9}$$

From these two inequalities (6.8) and (6.9), we can conclude $MR_{rmax} \leq uBMR_{node}$ or, $MR_{r \in node} \leq uBMR_{node}$.

Similarly, the lower bound $lBMR_{node}$ can be shown as follows: $lBMR_{node} \leq minMR_{node}$.

Thus, any record in *node* is certain to have MR value in between $uBMR_{node}$ and $lBMR_{node}$.

□

Observation 1. *Once the control comes back from* **DivGetBatch()***,* **Aug-MMR** *works exactly as the original MMR in each iteration.*

Aug-MMR has identical MR score calculation and MMR selection as that of the original MMR.

Claim 2. **Aug-MMR** *requires $\mathcal{O}((N/C + C) * k^2 + N)$ time in expectation.*

Proof. In the original MMR algorithm, each iteration for finding one record takes $\mathcal{O}(N * k)$ times. For k iterations, the overall running time is therefore $\mathcal{O}(N * k^2)$. The running time of **Aug-MMR** does not need to go over all N records in each iteration. Instead, it relies on **DivGetBatch()** to obtain a smaller set $CandR$ records.

Part 1. Running time of the API: A single iteration of **DivGetBatch()** needs to go over all the nodes in **I-tree** and takes $\mathcal{O}(C * k)$ time. **DivGetBatch()** has to compute two subroutines:

Calculate-Bound and *Skip-Nodes*. To compute these two functions, it takes $\mathcal{O}(N)$ time. Therefore, the overall running time is $\mathcal{O}(C * k^2 + N)$, where C is the total number of nodes.

Part 2. Running time of the rest of computation: The rest of the computation depends on the size of $CandR$. Let us assume, **DivGetBatch()** returns $|CandR_i|$ records in the i-th iteration. Accordingly, we have:

$$T_{\mathbf{Aug-MMR}} = \mathcal{O}(C * k^2 + N + \sum_{i=1}^{k} |CandR_i| * k).$$

The expected case analysis basically delves deeper into the analysis of **Part 2** and studies the expected running time considering different size of $CandR_i$ and its corresponding probability.

Let us assume, in iteration i, the $|CandR_i|$ records touch x number of nodes in **I-tree**. Indeed, x_i is the number of nodes with $|CandR_i|$ records in **I-tree**, that the augmented algorithms have to access during the query processing. Let us also assume node n_i contains v_i records. We start the proof assuming there is only one level in **I-tree** (i.e., $l = 1$), and then generalize it later on. If $l = 1$, the expected running time of Part 2 calculation of **Aug-MMR** in the i-th iteration is:

$$E = \mathcal{O}(\sum_{i=1}^{C} prob(x_i) \times \text{computation cost}_{\mathbf{Aug-MMR}}(x_i)).$$

Now, probability of returning x nodes $= \binom{C}{x}$ * probability of x nodes getting selected * probability of $(C-x)$ nodes not getting selected.

We assume that both data and query follow uniform distributions, thereby, each node has an equal probability of getting selected or skipped. The probability of choosing a node is $1/C$. Therefore, the probability of not getting selected is $(1-1/C)$.

The size of the returned record set, i.e., $|CandR|$, if $x = i$ nodes are accessed:

$$
\begin{aligned}
|CandR|_i = {} & (1/C)^i * (1-1/C)^{C-i} * [(v_1 + v_2 + ... + v_i) \\
& + (v_1 + v_3 + ... + v_{i+1}) + (v_2 + v_3 + ... + v_{i+1}) \\
& + (v_3 + v_4 + ... + v_{i+2}) + \ldots] \\
& = (1/C)^i * (1-1/C)^{C-i} * \binom{C-1}{i-1} \\
& * (v_1 + v_2 + \ldots + v_C) \\
& = (1/C)^i * (1-1/C)^{C-i} * \binom{C-1}{i-1} * N.
\end{aligned}
$$

Therefore, the overall expected cost of Part 2 is:

$$
\begin{aligned}
|CandR| & = N * \sum_{i=1}^{C} (1/C)^i * (1-1/C)^{C-i} * \binom{C-1}{i-1} \\
& = N * (1/C)/(1-1/C) * \sum_{i=1}^{C} (1/C)^{i-1} * \\
& (1-1/C)^{C-(i-1)} * \binom{C-1}{i-1}. \\
& \mathit{Let}\ j = i-1: \\
& = N * (1/C)/(1-1/C) * \sum_{j=0}^{C-1} (1/C)^j * \\
& (1-1/C)^{C-j} * \binom{C-1}{j} \\
& = N * (1/C)/(1-1/C) * (1-1/C) *
\end{aligned}
$$

$$\sum_{j=0}^{C-1}(1/C)^j * (1-1/C)^{(C-1)-j} * \binom{C-1}{j}$$

$$= N * (1/C)/(1-1/C) *$$

$$(1-1/C) * (1/C + 1 - 1/C)^{C-1} = N/C.$$

Expected running time of **Aug-MMR** algorithm considering both Part 1 and Part 2 computation is:

$E_{\mathbf{Aug-MMR}} = \mathcal{O}((N/C + C) * k^2 + N)$.

Now consider the case when $l > 1$. Probability of selecting a node in first level is $1/m$, given m is the arity of **I-tree**. Probability of selecting a node in second level = probability of selecting that node out of m node in that branch * probability of selecting it's parent = $1/m^2$. Similarly, Probability of selecting a node at leaf node is $1/m^l = 1/C$. Thus, in the general case, when $l > 1$, expected running time of **Aug-MMR** is $\mathcal{O}((N/C + C) * k^2 + N)$, which is same as before.

□

Worst-case Aug-MMR . In the worst-case, all N records are returned by **DivGetBatch()** in each iteration, which makes $\sum_{i=1}^{k} |CandR_i| = N * k$. Thus, the worst-case running time is $\mathcal{O}((N + C) * k^2)$.

6.4 *GMM* Query Processing with DivGetBatch()

The second algorithm we study is GMM algorithm. We describe the original version of the algorithm and our augmented version and similar to the previous section. We also provide proofs on how our augmented version outperforms the original one.

6.4.1 GMM algorithm

The next algorithm we study is GMM [106] that tries to find a subset of k most diverse records among N records by maximizing the minimum pairwise distance. GMM does not require any external query. Based on the original design, the first two records in the result set S are provided in constant time by an *oracle*. Then, the algorithm iteratively goes through all records in R and finds a record whose minimum *diversity* (maximum similarity) with the previously selected records is the largest (smallest). It continues until $|S|=k$. The objective function is:

$$GMM(r) \leftarrow argmax_{r \in R \setminus S} min_{r_j \in S} Div(r, r_j), \tag{6.10}$$

where $Div(r, r_j)$ is the diversity score between record r and r_j. A keen reader may notice that GMM uses diversity (Div) in the objective function, whereas, in our study, we store similarity between records. Unless specified otherwise, $Div = 1 - sim$. The two similarity matrices, one that captures the similarity between every pair of records, and the other that captures that of between nodes, could be used to calculate Div.

6.4.2 Aug-GMM algorithm

Aug-GMM leverages the **DivGetBatch()** API to reduce the number of records to iterate on. Algorithm 16 describes the pseudo-code, where the **DivGetBatch()** returns a small subset of records $CandR$ which later on is fed to the original GMM algorithm to get the $nextBest$ record.

Calculate-Bounds: This function keeps track of the upper and lower bounds of scores between nodes ($uBGMM$ and $lBGMM$, respectively) using the same principles as that of the original GMM objective function (Equation (6.10)).

$$lBGMM_{node} \leftarrow min_{node' \in Z} \min Div(node, node'), \tag{6.11}$$

$$uBGMM_{node} \leftarrow min_{node' \in Z} \max Div(node, node'), \tag{6.12}$$

where Z is the set of nodes containing S, $\min Div(node,$
$node')$ and $\max Div(node, node')$ are the minimum and the maximum diversity scores between any two records in $node$ and $node'$, respectively. In Equation (6.11), minimum of the minimum diversity over all nodes in Z ensures the lower bound of GMM, such that all records in $node$ will have equal or greater value than $lBGMM_{node}$. Conversely, in Equation (6.12), minimum of the maximum diversity over all nodes in Z ensures the upper bounds, such that all records in $node$ will have equal or lower GMM value than $uBGMM_{node}$.

Skip-Nodes : This function is identical to Skip-Nodes of MMR in principle. The skip-rationale of **Aug-GMM** is:

$$CandR \leftarrow \{N - \{r \in \mathbf{I} - \mathbf{tree}.n \mid uBGMM_n < \max_{\forall n'}(lGMM_{n'})\}\} \tag{6.13}$$

Running Example: Let us assume $k = 3$ and the first two records of S are arbitrarily chosen as r_1 and r_3. Initially, $S = \{r_1, r_3\}$. From Figure 6.1, r_1 and r_3 are inside $node_1$ and $node_2$, respectively. Hence, $Z = \{node_1, node_2\}$. Node-Node diversity $Div(node, node')$ can be calculated using Div = 1 - Sim. $Div(node_3, node_1)$ = (0.884, 0.908) and $Div(node_3, node_2)$ = (0.937, 0.9530). By using Equations (6.11) and (6.12), $lBGMM_{node_3}$ = 0.884 (as min of min div) and $uBGMM_{node_3}$ = 0.908 (as min of max div). Similarly, $lBGMM_{node_1}$, $uBGMM$
$_{node_1}$, $lBGMM_{node_2}$, and $uBGMM_{node_2}$ are 0, 0.031, 0, and 0.018. $lBGMM_{node_3}$ (0.884) is greater than $uBGMM$

$_{node_1}$ (0.031) and $uBGMM_{node_2}$ (0.018). Using Equation (6.13), $node_1$ and $node_2$ can be discarded. Obtaining records from $node_3$, $candR = \{r_5, r_6, r_7\}$ is returned from **DivGetBatch()**. Finally, GMM$(r_5, r_6, r_7) = r_5$ is called and the result set $S = \{r_1, r_3, r_5\}$ is achieved.

Aug-GMM algorithm proofs

Claim 3. **Aug-GMM** *returns identical top-k results as that of original GMM.*

Proof. Akin to MMR proof, this proof is also constructed using one helper lemma and one observation: Lemma 13 proves that **DivGetBatch()** never prunes a record that is part of the original top-k; Observation 2 shows that in each iteration, once the control comes back from **DivGetBatch()**, **Aug-GMM** works exactly as the original GMM. Combining these lemma and observation, **Aug-GMM** returns identical top-k results as that of the original GMM. □

Lemma 13. **DivGetBatch()** *never prunes a record that is part of the original top-k.*

Proof. As part of this proof, we first prove that Skip-Nodes *never discards* the record which has the highest GMM score in that iteration.

We use helper Lemma 14 to prove that the actual GMM score of any record in a node *node* is bounded between $uBGMM_{node}$ and $lBGMM_{node}$. The rest of the proof is identical to Lemma 11 of **Aug-MMR**. □

Lemma 14. *GMM score of any record $r \in node$ (say GMM_r) is bounded by upper and lower bound*
$uBGMM_{node}$ and $lBGMM_{node}$, respectively. That is,

$$lBGMM_{node} \leq GMM_{r \in node} \leq uBGMM_{node}.$$

Proof. Let us first consider $uBGMM_{node}$, by assuming
$F(node, r_j) = max_{r_i \in node} Div(r_i, r_j)$, it can be re-written as:

$$uBGMM_{node} \leftarrow min_{node' \in Z}[max_{r_j \in node'} F(node, r_j)], \tag{6.14}$$

Let us assume, maximum GMM value produced by any record in $node$ is $maxGMM_{node}$. According to Equation (6.10), $maxGMM_{node}$ is expressed as follows:

$$\begin{aligned} maxGMM_{node} &= max_{r_i \in node}[min_{r_j \in S} Div(r_i, r_j)], \\ &= min_{r_j \in S}[max_{r_i \in node} Div(r_i, r_j)], \\ &= min_{r_j \in S} F(node, r_j), \\ &\leq min_{node' \in Z}[max_{r_j \in node'} F(node, r_j)], \\ &= uBGMM_{node}, \text{[using Equation (6.14)].} \end{aligned}$$

similarly, it can be proved that, $minGMM_{node} \geq lBGMM_{node}$.

□

Observation 2. *Once the control comes back from* **DivGetBatch()***,* **Aug-GMM** *works exactly as the original GMM in each iteration.*

Aug-GMM does exactly same calculation as the original GMM does on a set of records as a result it will produce the same record as GMM does in a single iteration.

Claim 4. Aug-GMM *requires $\mathcal{O}(N/C + C) * k)$ time in expectation.*

Proof. In the GMM algorithm, each iteration for finding one record takes $\mathcal{O}(N)$ times. For k iteration, the overall running time is $\mathcal{O}(N * k)$. Similar to **Aug-MMR**, **Aug-GMM** does not need to go over all N records in each iteration, instead relies on **DivGetBatch()** to obtain a smaller set $CandR$ records.

Part 1. Running time of the API: A single iteration of **DivGetBatch()** needs to go over all the nodes in **I-tree** and takes $\mathcal{O}(C)$ time. **DivGetBatch()** has to compute two subroutines:

Calculate-Bound() and *Skip-Nodes*(). To compute these two functions, it takes $\mathcal{O}(C)$ time. Therefore, the overall running time is $\mathcal{O}(C * k)$, where C is the total number of nodes.

Part 2. Running time of the rest of computation: Similar to **Aug-MMR**, The rest of the computation depends on the size of $CandR$. Let us assume, **DivGetBatch()** returns $|CandR_i|$ records in the i-th iteration. Hence, we have:

$$T_{\mathbf{Aug-GMM}} = \mathcal{O}(C * k + \sum_{i=1}^{k} |CandR_i|).$$

The expected case analysis basically delves deeper into the analysis of **Part 2** and studies the expected running time considering different size of $CandR_i$ and its corresponding probability. By performing similar calculation as that of **Aug-MMR** as shown before, the expected cost of **Aug-GMM** is:

$$E_{\mathbf{Aug-GMM}} = \mathcal{O}((N/C + C) * k).$$

□

Worst-case Aug-GMM . In the worst-case, all N records are returned by **DivGetBatch()** in each iteration, which makes $\sum_{i=1}^{k} |CandR_i| = N * k$. Then the worst-case running time is: $\mathcal{O}((N + C) * k)$.

6.5 *SWAP* Query Processing with DivGetBatch()

The last algorithm we study is *SWAP* [211]. We describe the original version and our proposed augmented version. Similar to the previous sections, we provide theoretical analysis.

Algorithm 16 Aug-GMM

Inputs: **I-tree**, D, GMM, k

Output: S: final top-k result set

$S \leftarrow$ two records selected by an oracle
$R \leftarrow \{D - S\}$
for $t = 1$ **to** $k - 2$ **do**
    $CandR \leftarrow$ **DivGetBatch**(**I-tree**, R, S, GMM)
    $S = \{S \bigcup GMM(r)_{r \in CandR}\}$
end for
return S

6.5.1 $SWAP$ algorithm

SWAP [211] is a greedy algorithm that produces top-k results based on a given query Q and a tunable parameter that controls how much relevance could at most drop between any two records in the top-k results. The algorithm starts by sorting the records w.r.t. relevance and initializing the top-k result set S with the k-records with the highest relevance score with Q. It finds a *candidate record* from the current top-k set that has the smallest diversity contribution based on Equation (6.15). Indeed, in each iteration, it attempts to swap one record from R \ S with the candidate record. It starts scanning the remaining sorted relevance list from the top. In every iteration, it attempts to swap one record from the current top-k set with another from sorted R if the latter record has a higher contribution to diversity while ensuring the threshold of relevance drop. The algorithm terminates when the relevance drop is below the threshold, or R is fully scanned.

$$Divcont(r_i, S) = \sum_{r_j \in S} Div(r_i, r_j). \tag{6.15}$$

6.5.2 Aug-SWAP algorithm

Aug-SWAP is identical to the $SWAP$, i.e., it scans the sorted relevance list R, after initializing the top-k set S. It calls the **DivGetBatch()** API to retrieve a smaller set of candidate records $CandR$. These $CandR$ records are eligible to be considered during the next swap. If a record in R is not in $CandR$, then it is skipped. The rest of the process is identical to the original $SWAP$ algorithm. Algorithm 17 contains the pseudo-code.

Calculate-Bounds: Once the records are sorted w.r.t. relevance score, the diversity computation becomes query independent, and only between the records. This function calculates the upper and lower bounds of diversity contribution of nodes by leveraging $MinsimMatrixNode$ and $MaxsimMatrixNode$ considering the set of nodes Z that contains S, as below:

$$uBSWAP_{node} \leftarrow \sum_{node' \in Z} \max Div(node, node'), \tag{6.16}$$

$$lBSWAP_{node} \leftarrow \sum_{node' \in Z} \min Div(node, node'), \tag{6.17}$$

where $\max Div(node, node')$ and $\min Div(node, node')$ are the max and the min diversity between $node$ and $node'$. Naturally, the maximum (minimum) diversity is the maximum (minimum) of node diversities between $node$ and the nodes in Z.

Skip-Nodes: This function will then check if $uBSWAP_{node}$ is less than the diversity contribution of the candidate record (6.18); If the condition is true, it will prune the node and the entire subtree under it. In such a case, none of the records inside this node are eligible for swap because they will not increase the overall diversity of S. **DivGetBatch()** returns the records for all non-pruned nodes:

$$CandR \leftarrow \{N - \{r \in \mathbf{I} - \mathbf{tree}.n \mid uBSWAP_n < \\ min_{r_i \in S} \sum_{r_j \in S} Div(r_i, r_j)\}\} \tag{6.18}$$

Running Example: Lets say, $k = 2$, $UB = 0.9$, sorted $R = \{r_8, r_7, r_2, r_1, r_4, r_9, r_3, r_6, r_{10}\}$, and initial top-2 records selected as S=$\{r_8, r_7\}$. Using Equation 6.15, $Divcont(r_7, S) = 0.953$ and the *candidate* is r_7. From Figure 6.1, $Z = \{node_2, node_3\}$. Using Equations (6.16), (6.17), and Figure 6.1, if Div = 1 - sim, we have:
$uBSWAP_{node_1} = maxDiv(node_1, node_2) = 0.935$,
$lBSWAP_{node_1} = minDiv(node_1, node_2) = 0.925$.
Then, Equation 6.18 is applied and $node_1$ is discarded, $node_2$, $node_3$ are returned by **DivGetBatch()**, and $CandR = \{r_3, r_9, r_5, r_6\}$. Next record in the sorted list is r_2, which is not in $CandR$. As a result, r_2 will be skipped.

Aug-SWAP algorithm proofs

Claim 5. Aug-SWAP *returns identical top-k results as that of original* SWAP.

Proof. This proof is constructed using one helper lemma and one observation. Lemma 15 proves that **DivGetBatch()** does not skip a record that has a higher diversity contribution than that of the candidate record. Observation 3 shows that once all records returned in $CandR$, **Aug-SWAP** is identical to $SWAP$. Combining these lemma and observation, **Aug-SWAP** returns identical top-k results as that of the original $SWAP$.

□

Lemma 15. DivGetBatch() *never prunes a record that is part of the original top-k.*

Algorithm 17 Aug-SWAP

Inputs: **I-tree**, D, UB, k, $SWAP$

Output: S: final top-k result set.

$R \leftarrow$ Sort D on score;
$S \leftarrow$ TOPKITEMS(R, k)
$candidate \leftarrow argmin_{r_i \in S} Equation$ 6.15
$CandR \leftarrow R$
$pos \leftarrow k + 1$
while $candidate.score$ - $R[pos].score < UB$ **do**
    if $R[pos]$ **in** $CandR$ **then**
        if $Divcont(R[pos], S) > Divcont(candidate, S)$ **then**
            $S \leftarrow \{S - candidate \bigcup R[pos]\}$
            $CandR \leftarrow$ **DivGetBatch**(**I-tree**, R, S, Q, $SWAP$)
            $candidate \leftarrow argmin_{r_i \in S} Equation$ 6.15
        end if
    end if
    pos++
end while
return S

Proof. As part of this proof, we first prove that in each iteration SKIP-NODES *never discards* a record which has the higher diversity contribution than that of the candidate record. Let us assume, $r_{cand} \in S$ has lowest diversity contribution in S.

$$Divcont(r_{cand}, S) = min_{r_i \in S} \sum_{r_j \in S} Div(r_i, r_j)\}$$
$$= min_{r_i \in S} Divcont(r_i, S).$$

We use helper Lemma 16 to prove that the actual $DivCont$ score of any record in a node $node$ is bounded between $uBSWAP_{node}$ and $lBSWAP_{node}$. Let us assume, $r_d \in node_d$ is a record inside $node$, therefore,

$$\begin{aligned} uBSWAP_{node_d} &\geq Divcont(r_d, S) \\ &\geq Divcont(r_{cand}, S) \\ &= min_{r_i \in S} \sum_{r_j \in S} Div(r_i, r_j), \end{aligned}$$

as a result,

$$uBSWAP_{node_d} \geq min_{r_i \in S} \sum_{r_j \in S} Div(r_i, r_j). \tag{6.19}$$

From Equations (6.18) and (6.19), it is evident that $node_d$ containing r_d will not be skipped by SKIP-NODES. This logic extends to all the iterations SKIP-NODES calls. Hence the proof. □

Lemma 16. *Divcont score of any record $r \in node$ is bounded by upper and lower bound $uBSWAP_{node}$ and $lBSWAP_{node}$ respectively. That is,*

$$lBSWAP_{node} \leq Divcont(r, S)_{r \in node} \leq uBSWAP_{node}. \tag{6.20}$$

Proof. By replacing the value of $\max Div(node, node')$, the upper bound can be written as:

$$uBSWAP_{node} \leftarrow \sum_{node' \in Z} max_{r_i \in node, r_j \in node'} Div(r_i, r_j). \tag{6.21}$$

For any record $r \in node$ and $r_j \in S$, $r_j \in node_d$ and $node_j \in Z$,

$$Div(r, r_j) \leq max_{r_i \in node} Div(r_i, r_j),$$

Or,

$$\sum_{r_j \in S} Div(r, r_j) \leq \sum_{node' \in Z} max_{r_i \in node, r_j \in node'} Div(r_i, r_j),$$

As a result, $Divcont(r, S) \leq uBSWAP_{node}$. similarly, we can prove: $Divcont(r, S) \geq lBSWAP_{node}$.

□

Observation 3. *Once the control comes back from* **DivGetBatch()***,* **Aug-SWAP** *works exactly as the original $SWAP$ does in each iteration.*

Aug-SWAP performs identical calculation of $SWAP$ on the records that are not pruned by **DivGetBatch()**.

Claim 6. **Aug-SWAP** *requires $\mathcal{O}(N/C * k * \log k + N)$ time in expectation.*

Proof. In the original $SWAP$ algorithm, each iteration to select a new record to be swapped with the candidate record takes $\mathcal{O}(k * \log k)$ time. Therefore, for going over all records in R, it takes $\mathcal{O}(N * k * \log k)$. **Aug-SWAP** does not need to perform $\mathcal{O}(N * k * \log k)$, instead relies on **DivGetBatch()** to obtain a smaller set $CandR$ records.

Part 1. Running time of the API: A single iteration of **DivGetBatch()** needs to go over all the nodes in **I-tree**. **DivGetBatch()** has to compute two subroutines: *Calculate-Bound* and *Skip-Nodes*. By updating only the most recent swapped records and using dynamic programming, the two subroutines' overall running time is $\mathcal{O}(C)$, where C is the total number of nodes. However, how many times the API gets called depends on the number of times the swap condition gets satisfied (recall lines 8-10 in **Aug-SWAP** algorithm).

Part 2. Running time of the rest of computation: The other major computation of this algorithm is the running time of a record be swapped, which is $\mathcal{O}(k * \log k)$ and $Divcont$ running time in the Algorithm 5 line 8, which is $\mathcal{O}(k)$. How many times $Divcont$ gets executed depends on Line 7 in the **Aug-SWAP** algorithm is satisfied. The number of times $SWAP$ gets executed depends on swap condition, which is Line 8 in the **Aug-SWAP** algorithm. Finally, the entire R needs to be exhausted (as long as the bound drop threshold is satisfied), which takes $\mathcal{O}(N)$ time. As a result, we have:

$$\begin{aligned} T_{\textbf{Aug-SWAP}} = \mathcal{O}(&Number\, of\, times\, swap\, is\, satisfied \\ &* \textbf{DivGetBatch}()\, runtime + \\ &Number\, of\, times\, swap\, is \\ &satisfied * SWAP\, runtime + \\ &number\, of\, times\, line\, 7\, is\, satisfied * \\ &Divcont\, runtime + N). \end{aligned}$$

By considering running time of single *Divcont*,
$SWAP$, and **DivGetBatch**() call, overall running time of **Aug-SWAP** becomes:

$$\begin{aligned} T_{\textbf{Aug-SWAP}} = \mathcal{O}(&Number\, of\, times\, swap\, is\, satisfied \\ &* C + Number\, of\, times\, swap\, is\, satisfied \\ &* k * \log k + number\, of\, times\, line\, 7 \\ &is\, satisfied * k + N). \\ = \mathcal{O}(&\sum_{i=1}^{N} [probability\, of\;\; swap\, satisfied \\ &* C + probability\, of\;\; swap\, satisfied \\ &* k * \log k + probability\, of number\, of \\ &times\, line\, 7\, is\, satisfied * k] + N) \end{aligned}$$

Expected size of $CandR$ is $\sum_{i=1}^{N} \frac{|CandR_i|}{N}$. Probability of line 7 satisfied = probability that $R[pos]$ is in $CandR$ = $\frac{\sum_{i=1}^{N} \frac{|CandR_i|}{N}}{N}$. Without further information, the probability of a record getting swapped is 1/2 (same as not getting swapped). Probability of $SWAP = 1/2 *$line 7 is satisfied $= 1/2 * \frac{\sum_{i=1}^{N} \frac{|CandR_i|}{N}}{N}$. Expected running time (cost) of **Aug-SWAP** is:

$$
\begin{aligned}
E_{\mathbf{Aug-SWAP}} = & \sum_{i=1}^{N}[1/2 * \frac{\sum_{i=1}^{N} \frac{|CandR_i|}{N}}{N} * (C + k * \log k) \\
& + \frac{\sum_{i=1}^{N} \frac{|CandR_i|}{N}}{N} * k] + N \\
& = 1/2 * \sum_{i=1}^{N} \frac{|CandR_i|}{N} * (C + k * \log k) \\
& + \sum_{i=1}^{N} \frac{|CandR_i|}{N} * k + N \\
& = \mathcal{O}(\sum_{i=1}^{N} \frac{|CandR_i|}{N} * (C + k * \log k) + N)
\end{aligned}
$$

First, we study the Part 2 computation having two costs associated with it, cost of *Divcont* and cost that of $SWAP$. Based on Line 7 of Algorithm 5, if $CandR$ is large, it is likely to have $R[pos]$ inside it. In fact, if $CandR$ contains all R records, $R[pos]$ will always be there. For the purpose of illustration, let us assume, in the i-th iteration, $|CandR_i|$ records touch x number of nodes in **I-tree** and node n_i contains v_i records. Therefore, the probability that $R[pos]$ is in $CandR_i = \frac{\sum_{q=1}^{x} v_q}{N}$.

The expected running time of $SWAP$ in terms of C is: $\binom{C}{x}$ * probability of x nodes getting selected * probability of $(C-x)$ nodes not getting selected * probability of $R[pos]$ is in $CandR_i$ * probability of swap * cost of swap.

The probability of $x = i$ and $R[pos]$ is in $CandR_i$ is:

$$
\begin{aligned}
& = (1/C)^i * (1 - 1/C)^{C-i} * [(v_1/N + v_2/N + \cdots + v_i/N) \\
& + (v_1/N + v_3/N + \cdots + v_i/N) + \ldots \\
& + (v_{C-i}/N + \cdots + v_C/N)] \\
& = (1/C)^i * (1 - 1/C)^{C-i} * \binom{C-1}{i-1} * (\frac{v_1 + v_2 + \cdots + v_c}{N}).
\end{aligned}
$$

$$= (1/C)^i * (1 - 1/C)^{C-i} * \binom{C-1}{i-1}.$$

Therefore, the expected running time (cost) of $SWAP$ is,

$$E_{SWAP} = 1/2 * N * k * \log k * \sum_{i=1}^{C} (1/C)^i * (1 - 1/C)^{C-i} *$$
$$\binom{C-1}{i-1} = 1/2 * N/C * k * \log k.$$

Expected running cost of $Divcont$ is $\binom{C}{x}$ * probability of x nodes getting selected * probability of $(C - x)$ nodes not getting selected * probability of $R[pos]$ is in $CandR_i$ * cost of $Divcont$. Therefore, the expected running time (cost) of $Divcont$ is:

$$E_{Divcont} = N * k * \sum_{i=1}^{C} (1/C)^i * (1 - 1/C)^{C-i} * \binom{C-1}{i-1}$$
$$= N/C * k.$$

The expected cost of Part 2 becomes:

$$E_{Part_2} = 1/2 * N/C * k * \log k + N/C * k.$$

The expected running time (cost) of Part 1 is = $\binom{C}{x}$ * probability of x nodes getting selected * probability of $(C - x)$ nodes not getting selected * probability of $R[pos]$ is in $CandR_i$ * probability of swap * cost of **DivGetBatch()**. Using similar calculation as above, expected cost of part 1 is:

$$E_{part_1} = 1/2 * N * \sum_{i=1}^{C} (1/C)^i * (1 - 1/C)^{C-i} *$$
$$\binom{C-1}{i-1} * C = N/2.$$

Expected running time of **Aug-SWAP** algorithm considering both Part 1 and Part 2 computation is:

$$E_{\mathbf{Aug-SWAP}} = 1/2 * N/C * k * \log k + N/C * k + N/2$$
$$+N = \mathcal{O}(N/C * k * \log k + N)$$

Now consider the case when $l > 1$ for **Aug-SWAP**. Probability of selecting a node in first level is $1/m$, given m is the arity of **I-tree**. Probability of selecting a node in second level = probability of selecting that node out of m node in that branch * probability of selecting it's parent = $1/m^2$. Similarly, Probability of selecting a node at leaf node is $1/m^l = 1/C$. As the records are only returned from leaf nodes, the expected probability that $R[pos]$ is in $CandR_i$ does not change for $l > 1$. The running time of **DivGetBatch()** $= \mathcal{O}(m^l) = \mathcal{O}(C)$ also stays same . The rest of the computation does not directly depend on l. As a result, expected running time of **Aug-SWAP** for $l > 1$ is same as before.

□

Worst-case Aug-SWAP. In the worst-case, none of the records are skipped, so the number of swap is $\mathcal{O}(N)$. Therefore, the worst-case running time is: $\mathcal{O}(N{*}C{*}k{*}\log k)$.

Our technical results are summarized in Tables 6.1 and 6.2.

6.6 I-tree

The index is a hierarchical complete tree-like structure [131] that partitions D into multiple groups of records. Each node in **I-tree** consists of a group of similar records. The index structure maintains a higher level aggregate similarity between nodes [3]. **I-tree** is applicable not only to the studied three algorithms, but also to any content-based algorithm that is either based on replacing items in the top-k or building the top-k in an incremental fashion.

[3] Diversity between a pair of records is simply $1 - \textit{similarity}$ between them.

Algorithm 18 Indexing Algorithm *BuildTree(node)*

Inputs: database D of N records, m: arity of the tree, l: number of levels,

Outputs: **I-tree**, *simMatrixNode*: node-node similarity matrix, *recordMap*: a mapping of all records and their belonging node id for each level.

$rootnode \leftarrow N$ records, $y = 0$

$nodelist[y] \leftarrow rootnode$

while $y \leq l$ **do**

  for $node$ **in** $nodelist[y]$ **do**

    $cnodes \leftarrow Partition(node, m)$

    I-tree $[y][node]$.ADDCHILD(cnodes)

    $w \leftarrow \bigcup cnodes$

    $recordMap[y][r] \leftarrow$ *node id* containing record r in y

  end for

  $MinsimMatrixNode[y][i][j] \leftarrow$ Use Equation (6.23)

  $MaxsimMatrixNode[y][i][j] \leftarrow$ Use Equation (6.24)

  $nodelist[y] \leftarrow w$

  $y \leftarrow y + 1;$

end while

6.6.1 Index construction

The input to the indexing step is a $N \times N$ matrix, named *simMatrixRecord*. It represents the similarity scores between every pair of records, r_i and r_j, in the database and two additional parameters, l and m, which are the number of levels and arity of the tree, respectively. The output is a complete m-ary tree with l levels, referred to as **I-tree**.

The indexing algorithm *BuildTree* (Algorithm 18) partitions (refer to the Subroutine *Partition*) the records. It also maintains additional data structures that contain similarity scores between nodes for efficient query processing. An example of a two-level index tree is shown in Figure. 6.2. At the first level, *BuildTree* creates a root node containing all N records and m children of the root node. From the point of abstraction, it is not important at this stage to describe how the data is partitioned. Basically, the goal is to keep similar records together while separating non-similar ones. There are multiple off-the-shelf techniques such as clustering and graph partitioning to carry out this task.

In our implementation, we use the popular k-means algorithm [112] for partitioning. The algorithm repeats the partitioning procedure until it reaches l levels. Therefore, **I-tree** contains a total of C nodes such that:

$$C = \sum_{i=0}^{l} m^i = \frac{m^{l+1} - 1}{m - 1} = \mathcal{O}(m^l) \tag{6.22}$$

Inside **I-tree**, additional data structures are maintained:

a. A *recordMap* of size $N \times l$ that maps the id of a record with the id of its node in each level from $1 \ldots l$. b. *MinsimMatrixNode* and *MaxsimMatrixNode* that contain inter-node minimum and maximum similarities between any two nodes in the same level, respectively. Particularly, for two nodes n and n' in level y, *MinsimMatrixNode* and *MaxsimMatrixNode* contain:

$$MinsimMatrixNode[i, j] = Min_{r \in i, r' \in j} sim(r, r'), \tag{6.23}$$

$$MaxsimMatrixNode[i, j] = Max_{r \in i, r' \in j} sim(r, r'), \tag{6.24}$$

where, $r \in n, r' \in n'$. Figure 6.1 contains these scores for 3 nodes of our running example.

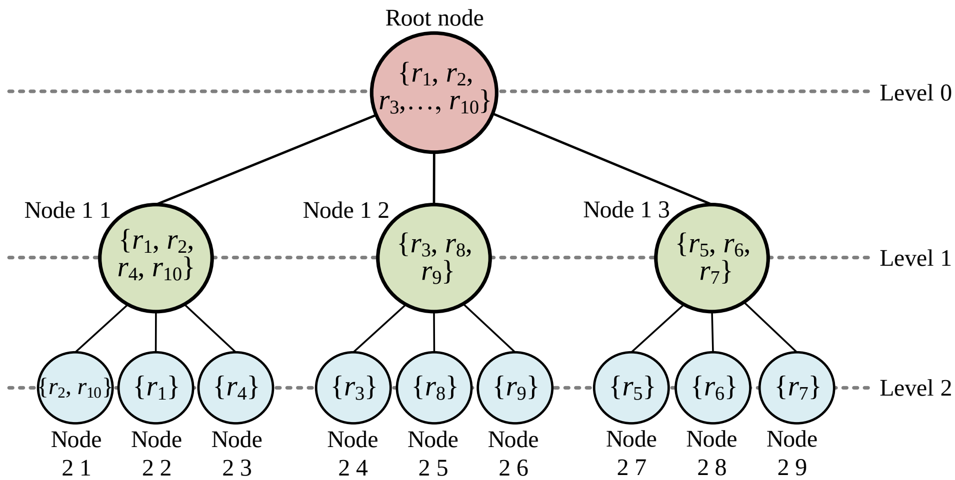


Figure 6.2 I-tree.

6.6.2 Index maintenance

Even for a single insertion or deletion, **I-tree** requires the following two activities: a. insertion/deletion of that record from/into **I-tree**; b. updating *MinsimMatrixNode* and *MaxsimMatrixNode*, if these insertion/deletion require updating the minimum and maximum similarity scores between nodes. One can easily see that (a) could be achieved in a constant time when l =1 and $\mathcal{O}(l)$ when l greater than 1. However, a single insertion/deletion may require as many as $2 \times (C - 1)$ updates in these two matrices.

Batch Update We study how to maintain **I-tree** considering both insertions and deletions.

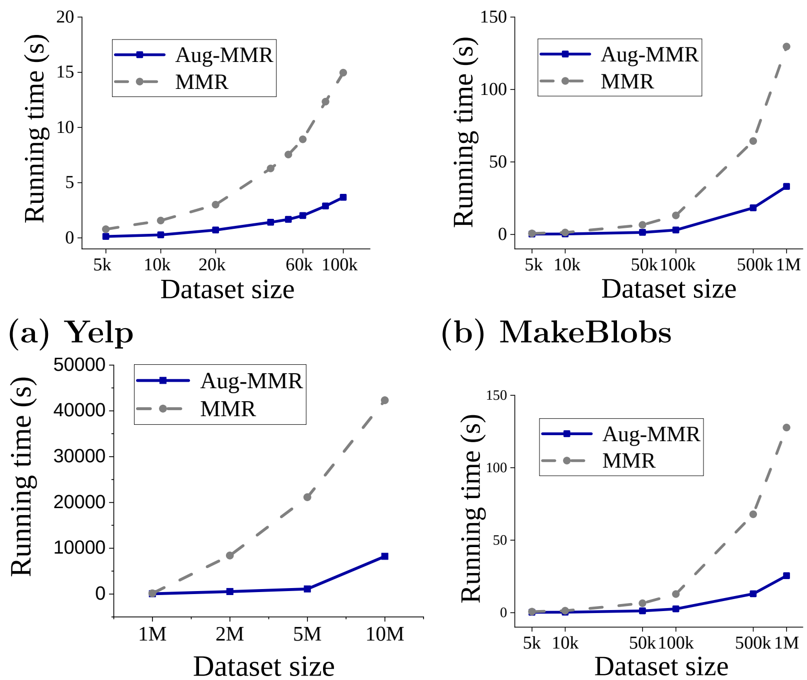
Batch Deletion. Let us assume a batch of R records are to be deleted from **I-tree**. The process deletes these R records one by one and then checks how many entries in *MinsimMatrixNode* and *MaxsimMatrixNode* need update (if the deleted records contribute to these aggregate values, then that require updates in those two matrices, else not). The overall process takes $\mathcal{O}(|Y| \times C \times N)$ time.

Batch Insertion. This problem is more complicated. If the records are inserted arbitrarily inside **I-tree**, then, each insertion may potentially cause a total of $2 \times (C -$

1) updates in the *MinsimMatrixNode* and *MaxsimMatrixNode* data structures. This is *the leading computational cost of batch insertion*. Moreover, when a batch of records are inserted, it is possible to have multiple records to get inserted inside the same node, and that should not be double-counted in the process. Finally, one needs to insert the records to those nodes, such that the aggregates stored in *MinsimMatrixNode* and *MaxsimMatrixNode* remain "tight" to enable effective pruning. These nuances are explored in formalizing the batch insertion problem.

Problem Definition 6. (Batch Insert.) *Let* $Minsim$ $MatrixNode[i,j]$ *(similarly* $MaxsimMatrixNode[i,j]$*) denote the value after* $|Y|$ *insertions at the* $[i,j]$*-th entry at the* MinsimMatrixNode *(similarly* $MaxsimMatrix$ $Node$ *matrix). Let* $Minsim_{ij}$ *and* $Maxsim_{ij}$ *be two binary variables, such that which* $Minsim_{ij} = 1$ *(similarly* $Maxsim_{ij}$*) , if it requires an update after insertions,* 0 *otherwise. Our goal is to insert a batch of records* Y *such that, it minimizes* $\sum_{i,j} Minsim_{ij} + \sum_{i,j}$ $Maxsim_{ij}$*, i.e., the total number of updates in these two matrices.*

Algorithms. We present an integer programming-based solution **OPTMn** for solving the batch insert problem. While **OPTMn** indeed produces the optimal solution, due to its exponential nature, it does not scale to a very large dataset considering a large number of insertions. As an alternative, we present **GrMn** a greedy heuristic algorithm which makes greedy choices and indirectly attempts to minimize the number of updates in *MinsimMatrixNode* and *MaxsimMatrixNode* matrices. The idea is to make a greedy decision by inserting each of the incoming records to that node which it is closest to (based on the underlying similarity measure) and then check if that insertion requires any updates in *MinsimMatrixNode* and *MaxsimMatrixNode* matrices. The running time of this algorithm is $\mathcal{O}(|Y| \times N)$.



(a) **Yelp** (b) **MakeBlobs**

(c) **MakeBlobs, large scale** (d) **MovieLens**

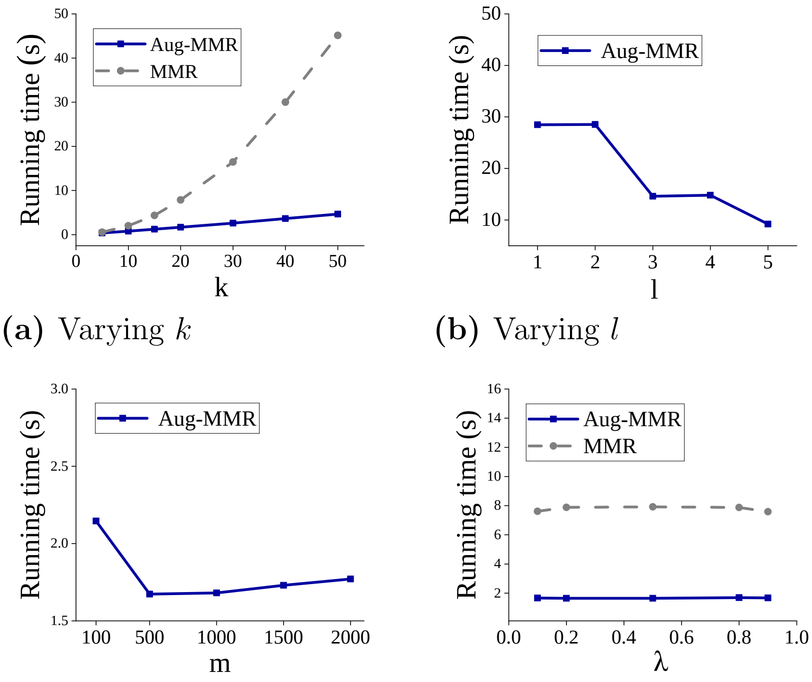
Figure 6.3 Aug-MMR vs MMR scalability.

6.7 Experimental Evaluation

Our experimental evaluations have three primary goals. First, we analyze if the augmented algorithms return identical results to their original counterparts using multiple large-scale datasets. Second, we examine the efficiency and scalability of the augmented algorithms and compare them with multiple baselines. Finally, we empirically study the cost of building and maintaining **I-tree**. For brevity, we present a subset of results that are representative.

Experimental setup. All algorithms are implemented in Python 3.8. All experiments are conducted on a cluster server OSL machine with 32GB RAM memory, OS: Scientific Linux release 7.8 (Nitrogen), CPU: Intel(R) Xeon(R) CPU E3-1245 v6 @ 3.70GHz. Obtained results are the average of three separate runs. [4]

[4]The code and data could be found at https://github.com/MouinulIslamNJIT/divGetBatch, Retrieved on 4/7/2023



(a) Varying k (b) Varying l

(c) Varying m (d) Varying λ

Figure 6.4 Aug-MMR vs MMR varying parameters.

Table 6.6 Dataset Statistics

Dataset	Size	#Total features	#Features used	Dataset type
Yelp	112,686	12	3	Real
MovieLens	1,000,209	3	2	Real
MovieLens non-metric	8,453	3	2	Real
UCI Gas dataset	13,911	128	128	Real
MakeBlobs	10,000,000	varied	20	Synthetic

Table 6.7 Aug-MMR vs MMR Running Time (s) on **MakeBlobs** with $l = 2$, $m = 6$

	Dataset Size			
Algorithm	**5k**	**10k**	**50k**	**10k**
Aug-MMR	4.33	8.69	43.57	306.11
MMR	19.77	40.16	197.28	1206.90

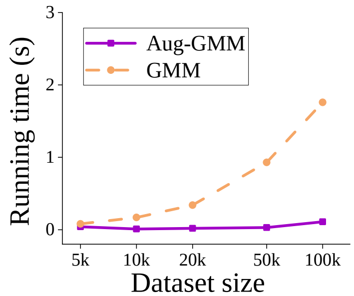
Diversity and Similarity. We use normalized Euclidean distance (dist) as diversity to validate our designed solutions in the geometric space, Cosine similarity [112] in general metric space. For non-metric distance, we use Movielens datasets and quantify the diversity between a pair of movies as the number of users who have rated either of these two movies but not both. We additionally use an arbitrary diversity function generated synthetically on Makeblobs dataset, such that it does not satisfy triangle inequality. Thus, diversity values are atomic for the last two cases, and are not derived from the feature vectors. For all these cases, $sim = 1 - dist$.

Query selection. In our experiments, queries are chosen randomly.

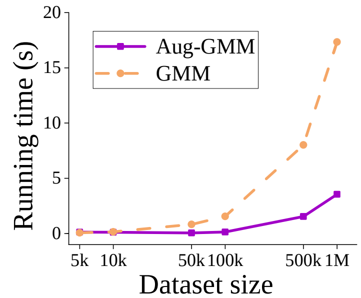
Performance Measures. We measure precision@k [112] for qualitative analysis. Efficiency of the proposed method is demonstrated with $|CandR|/N \times 100$, pruning $= 1 - |CandR|/N \times 100$, as well as by presenting the running times of the algorithms in seconds and computing *speedup* as follows:

$$speedup = \frac{T_{original-algorithm}}{T_{augmented-algorithm}} \tag{6.25}$$

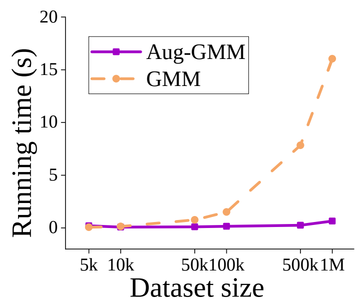
where T denotes running time in seconds. Finally, we present time to build **I-tree** and the space required for that.



(a) Yelp



(b) MakeBlobs



(c) MovieLens

Figure 6.5 Aug-GMM vs GMM scalability.

Datasets. Experiments are conducted on five datasets, four real and one publicly available synthetic data. For real datasets, we use **Yelp**[5], **UCI Gas** dataset [6] that is high dimensional, **MovieLens** 1M records, and **MovieLens non-metric** dataset[7]. For synthetic data, we use **MakeBlobs** from the sklearn package.[8] An overview of the datasets is given in Table 6.6.

6.7.1 Baselines

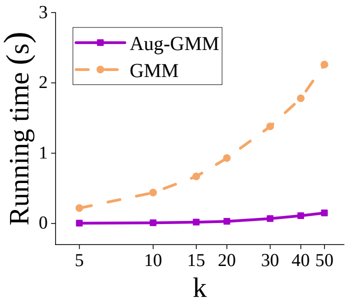
In this section, we introduce diversity-based algorithms and index structure baselines that we compare to our proposed solutions.

[5] `https://www.yelp.com/dataset/documentation/main`, Retrieved on 4/7/2023

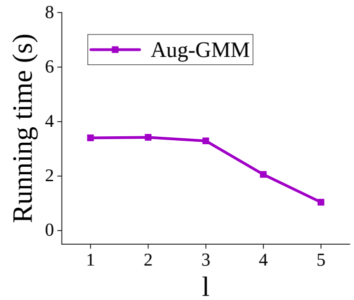
[6] `https://archive.ics.uci.edu/ml/datasets/gas+sensor+array+drift+dataset`, Retrieved on 4/7/2023

[7] `https://grouplens.org/datasets/movielens/`,Retrieved on 4/7/2023

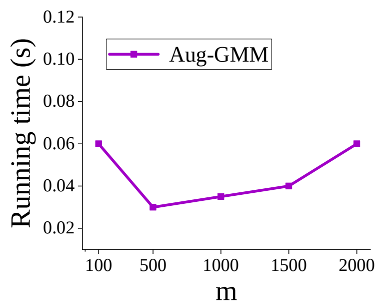
[8] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`,Retrieved on 4/7/2023



(a) Varying k



(b) Varying l



(c) Varying m

Figure 6.6 Aug-GMM vs *GMM* performance varying parameters.

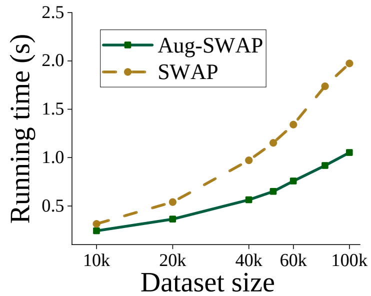
Diversity Baselines For diversity-based methods, three representative algorithms are implemented.

MMR [55]: computes an objective score based on two parameters: relevance to the query and diversity with other records. As shown in Equation (6.1), they are combined in a linear expression with a λ coefficient. The algorithm repeats this computation k times to produce top-k.

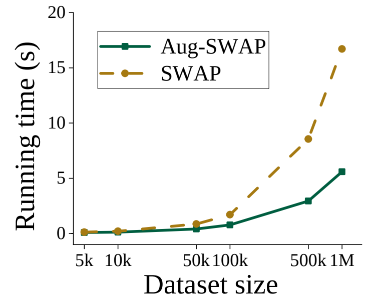
GMM [106]: finds the k most diverse records by selecting the maximum of minimum distances between undiscovered records and previously selected ones at each iteration (Equation (6.10)). Like MMR, it also iteratively builds the top-k set.

SWAP [211]: This greedy algorithm first finds the initial top-k records, then greedily interchanges records that are part of the current top-k with the ones that are remaining, if the *swap* improves diversity contribution (Equation (6.15)).

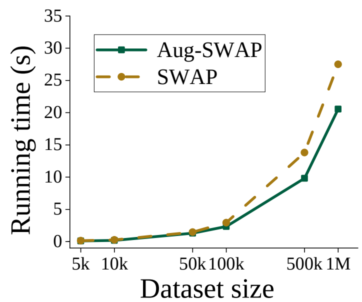
SPP [98]: Space Partitioning and Probing (SPP in short) is an algorithm that minimizes the number of accessed objects while finding exactly the same result as



(a) **Yelp**



(b) **MakeBlobs**



(c) **MovieLens**

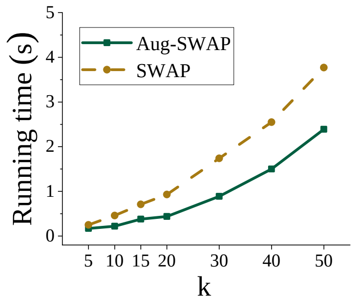
Figure 6.7 Aug-SWAP vs SWAP scalability.

MMR. *SPP* belongs to a family of algorithms that rely only on score-based and distance-based access methods, and does not require retrieving all the relevant objects. SPP is designed only for the geometric space.

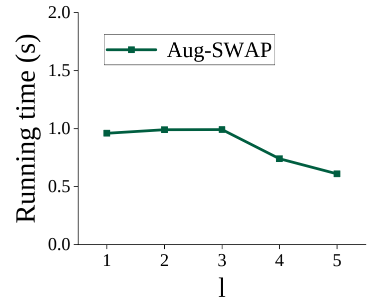
Index Structure Baselines We implement three additional baselines to compare against **I-tree**. These indexing techniques are limited to metric space, and can not be applied on arbitrary diversity function not satisfying triangular inequality.

KD-tree [34]:KD-tree is a multidimensional Binary Search Tree. The tree is created by bisecting each dimension and finding the median. *KD-tree* can perform searches in multidimensional space for efficient nearest neighbor search.

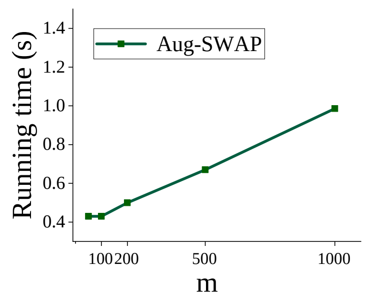
Ball-tree [135]: Ball-tree is a binary tree in which every node defines a D-dimensional hypersphere or ball, containing a subset of the points to be searched. Each node in the tree defines the smallest ball that contains all data points in its



(a) Varying k



(b) Varying l



(c) Varying m

Figure 6.8 Aug-SWAP vs SWAP varying parameters.

subtree. This gives rise to the useful property that for a given test point t outside the ball, the distance to any point in a ball B in the tree is greater than or equal to the distance from t to the surface of the ball. *Ball-tree* only supports binary splits.

The arity of the tree in both *KD-tree* and *Ball-tree* is fixed to 2.

M-Tree [68]: *M-tree* is similar to *Ball-tree*, but supports multiple splits. Every node n and leaf lf residing in a particular node N is at most distance r from N, and every node n and leaf lf with node parent N keeps the distance from it. It also has the similar property of *Ball-tree*, which is for a given test point t outside the node, the distance to any point in a node in the tree is greater than or equal to the distance from t to the surface of the node.

We are incorporating Node-Node distance matrix to these baseline tree index structures so that they can be used for **I-tree** API.

Cover-Tree [36]: Another popular indexing structure is cover tree which is used to enable efficient nearest neighbor search in metric space. To be able to work with

DivGetBatch(), the indexing technique must work in a fashion that the parent nodes of the index structure (in this case a tree) covers the records that are present in their sub-tree. This allows us to effectively maintain the inter-diversity bounds across the nodes and when a node gets pruned, all its children also does. Contrarily, in a cover tree, only the leaf nodes together contain and cover all the records and no other intermediate/ higher level nodes does. Therefore, it is not obvious how to adapt this indexing technique and integrate it inside our proposed access primitive.

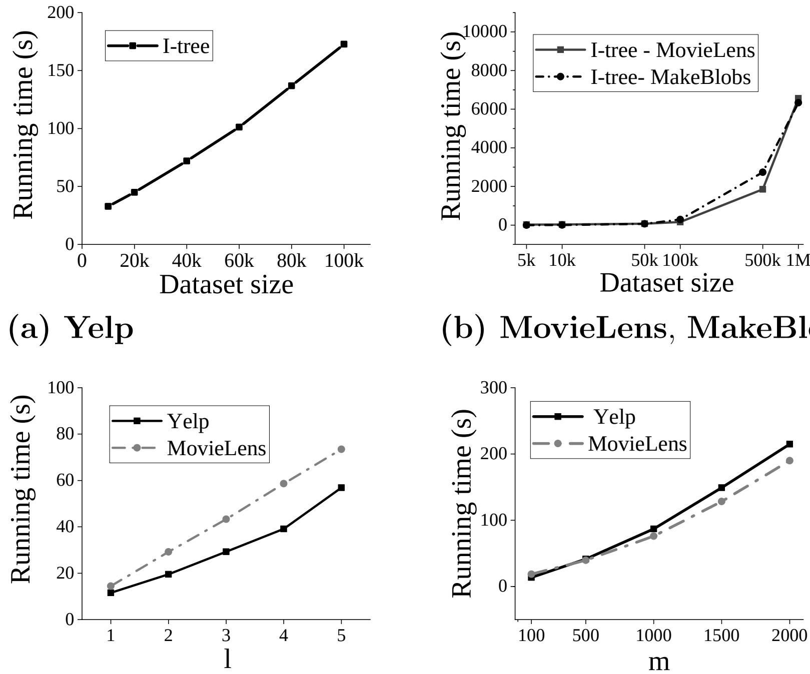
Index Maintenance Baselines **OPTMn** and **GrMn** are compared with two baselines.

NonIncrMn *Algorithm:* In **NonIncrMn**, **I-tree** is built from scratch after every $|Y|$ insertions. **NonOlMn** *Algorithm:* This algorithm makes a local decision to insert each record based on Problem 6, without accounting for overlapping updates inside the same node in **I-tree**.

6.7.2 Summary of results

Our first set of experiments verify that our results from all three augmented algorithms are *identical* to their original counterparts. We measure precision@k [112] for different k, and our empirical results obtain 100% precision score.

Our next set of experimental results demonstrate that the running time of the augmented algorithms are consistent with our theoretical analyses. We achieve a 19× and 24× speedup for **Aug-MMR** and **Aug-GMM**, on **Makeblobs** 10M and **MovieLens** 1M data, respectively. We achieve a 3× speedup for **Aug-SWAP** on **MakeBlobs** 1M dataset. These results corroborate that our proposed framework is suitable to scale on large datasets. We also show that **I-tree** works on any arbitrary distance functions while other baselines are designed for only metric distance functions. We have conducted experimental analysis on two different non-metric distance functions (one obtained from the real data), these experimental



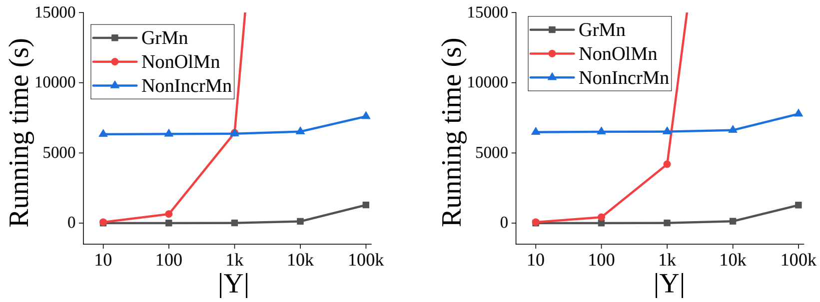
(a) Yelp **(b) MovieLens, MakeBlobs**

(c) varying l **(d)** varying m

Figure 6.9 I-tree construction time.

results demonstrate that **Aug-MMR** attains 82% pruning compared to the baseline solutions, resulting in about 2.7 times speed up on an average. On the other hand, the results obtained from high dimensional UCI Gas dataset demonstrate that the proposed framework is still effective even in higher dimension, as **Aug-MMR** attains about 1.7 speed up on an average.

Figures 6.11 demonstrate the index construction and the query processing time trade-off of **I-tree** and we compare that with our implemented baseline indexes, KD-tree, Ball-Tree, M-Tree. These results convincingly demonstrate that **I-tree** enables the fastest query processing time, while requiring comparable index construction time. The results demonstrate that **I-tree** is always more than $18\times$ faster in query processing and as much as $170\times$ faster for certain configurations. For preprocessing, it is always more than $1.5\times$ faster and at times it is more than $20\times$ faster. We also present $|CandR|$ percentage and pruning percentage of **I-tree**



(a) **MakeBlobs** (b) **MovieLens**

Figure 6.10 I-tree maintenance time varying $|Y|$.

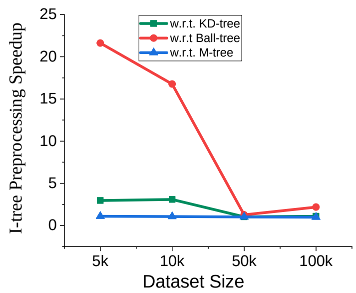
compared to other index baselines in Tables 6.9 and 6.10 which shows that **I-tree** outperforms all baselines with having 90% pruning.

The results convincingly demonstrate that **I-tree** is lightweight to compute and space efficient (for the largest dataset, it takes 109 minutes to build the index, which is acceptable because it is done offline and only once). Finally, we demonstrate that our proposed solution **OPTMn** is an ideal choice for incremental index maintenance, while the greedy heuristic **GrMn** is highly scalable while being not too inferior from the optimal solution **OPTMn** qualitatively. **GrMn** takes 22 minutes to insert $100k$ data into 1M dataset, while building **I-tree** from scratch is unrealistic as **NonIncrMn** takes 2 hours.

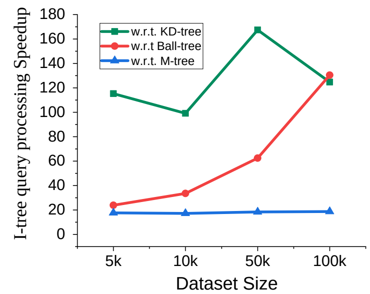
6.7.3 Quality analysis

The goal of these experiments is to empirically validate if the augmented algorithms produce the same results as their original counterparts. Additionally, we present how effective **DivGetBatch()** is in pruning records by presenting the size of $CandR$.

We have calculated precision@k while varying k from 10 to 50, considering the original and augmented algorithms. We obtain the precision@k equal to 100% always.



(a) I-tree Index Preprocessing speedup w.r.t baselines



(b) I-tree Query Processing speedup w.r.t baselines

Figure 6.11 Index Construction and Query Processing time for tree baselines and **I-tree**.

6.7.4 Scalability analysis

We run two types of scalability experiments. (i) demonstrate the efficacy of the augmented diversification algorithms and compare them appropriately with the baselines; (ii) demonstrate the efficacy of the indexing technique - present index construction and maintenance time, and compare them appropriately with the baselines. Additionally, we also present the memory requirements of **I-tree**. We analyze these effects by increasing dataset size and other pertinent parameters.

Augmented Diversification Algorithms We first vary dataset size, then additional parameters that impact the query processing time. To demonstrate efficacy, we present two things. (1) The percentage of remaining records returned by **DivGetBatch()**, which is which is $|CandR|/N \times 100$ and pruning $(1 - |CandR|/N \times 100$. (II) Query processing time in seconds.

Effectiveness in Pruning. In Table 6.8, we present the number of remaining records returned by **DivGetBatch()**, which is $|CandR|$ using **MovieLens** dataset. We can observe that there is a remarkable reduction compared to the original dataset. For example, **Aug-MMR** returns only 814 records. The biggest number is for **Aug-SWAP** with 66513 records, but still returning only 6% of the records.

Table 6.8 $|CandR|$ Percentage Returned by DivGetBatch() on MovieLens

	Dataset Size					
Algorithm	**5k**	**10k**	**50k**	**100k**	**500k**	**1M**
Aug-MMR	13%	5.21%	0.56%	0.09%	0.08%	0.08%
Aug-GMM	59.96%	15.48%	4.16%	2.67%	0.31%	0.4%
Aug-SWAP	14.96%	28.11%	10.07%	48.74%	9.27%	0.66%

Table 6.9 $|CandR|$ Percentage Returned by DivGetBatch() Using Different Index Structures for Aug-MMR on MakeBlobs

	Dataset Size			
Algorithm	**5k**	**10k**	**50k**	**100k**
I-tree	10%	10%	10%	10%
KD-tree	96.72%	96.72%	96.87%	97.34%
Ball-tree	96.7%	95.62%	96.56%	96.56%
M-tree	97.92%	97.19%	98.32%	98.07%

Table 6.9 and Table 6.10 show $|CandR|$ and pruning percentage returned by **DivGetBatch()** for **Aug-MMR** algorithm using different index structures and MakeBlobs dataset. We can see that by fixing $C = 32$, *KD-tree*, *Ball-tree*, and *M-tree* pruning are below 5%, while **I-tree** pruning considerably outperforms all baseline which is 90%.

Effectiveness in Number of Accesses. In order to perform a fair comparison between our augmented algorithms and SPP, we compare the number of I/O accesses SPP does and present that number for **Aug-MMR** (SPP is designed to optimize that access). We calculate the number of accesses in **DivGetBatch()** by counting the distinct records present in $CandR$ in k rounds. The results are presented in Table 6.11. We can see that **Aug-MMR** has less number of access. For example on 100k data, **I-tree** has 2799 number of access while SPP has 26521 number of access.

Table 6.10 Pruning Percentage by DivGetBatch() Using Different Index Structures for Aug-MMR on MakeBlobs

	Dataset Size			
Algorithm	**5k**	**10k**	**50k**	**100k**
I-tree	90%	90%	90%	90%
KD-tree	3.3%	3.3%	3.1%	2.6%
Ball-tree	3.3%	4.3%	3.4%	3.4%
M-tree	2%	2.8%	1.6%	1.9%

Table 6.11 Number of Access Percentage for Aug-MMR and SPP on MakeBlobs

	Dataset Size			
Algorithm	**5k**	**10k**	**50k**	**100k**
I-tree	10%	10%	5.2%	2.79%
SPP	20.44%	9.57%	27.31%	26.52%

Varying Dataset. Figures 6.3, 6.5, and 6.7 compare the running times of our three augmented algorithms and their baselines using our three datasets. As N increases, the running times of each algorithm and its baseline increase, but we observe that our algorithms are consistently faster and they scale significantly better. Figure 6.3 shows **Aug-MMR**'s scalability on all three datasets. We fix m to 1000, $k = 20$ and $l = 1$ for all dataset sizes while N is increased from 5000 up to 1M. We can see that on **MovieLens**, varying N from 5000 to 1M, **Aug-MMR** is 5× faster than *MMR*. Figure 6.5 shows **Aug-GMM** 's scalability. On **MovieLens**, varying N from 5000 to 10M, **Aug-GMM** is 24× faster than *GMM*. Consistent with the theoretical analysis, **Aug-GMM** is faster than **Aug-MMR** for the same settings because **Aug-MMR** has an additional k term in the expected cost equation. Figure 6.7 shows **Aug-SWAP** 's scalability on all three datasets. For the 1M data of **MakeBlobs** we obtain a 3× speedup over *SWAP*. We obtain a 1.33× speedup for **Movielens** because the total number of swaps in **MovieLens** are higher.

Table 6.12 Index Comparisons

Index	Metric Functions	Non metric Functions	90% Pruning
I-tree	✓	✓	✓
KD-tree [34]	✓	×	×
Ball-tree [135]	✓	×	×
M-tree [68]	✓	×	×

Table 6.13 Aug-MMR vs MMR Running Time on MakeBlobs 100k Records

	Distance function		
Algorithm	**Euclidean**	**Cosine**	**Non-metric**
Aug-MMR	3.08	4.64	13.06
MMR	13.12	15.36	15.27

We also measure the scalability of **Aug-MMR** compared to MMR using large scale data sizes of 2M, 5M, and 10M using makeBlobs dataset. The results are shown in Figure 6.3(c) in which with $m = 1000$ and $l = 1$, we have up to 19× speedup.

Moreover, we run **Aug-MMR** on high-dimensional euclidean distance considering more number of features using 1M and 2M makeBlobs dataset. for 1M data, 1M and 20 features, MMR takes 12492.64 (s), and **Aug-MMR** takes 2817.14 (s). For 2M data and 20 features, MMR takes 25812.43 9 (s), **Aug-MMR** takes 6317.20 (s) which in both case show 4× speedup.

Additionally, Figure 6.12 presents the scalability of the proposed **Aug-MMR** algorithm compared to MMR using UCI Gas dataset with 10k records and 128

Table 6.14 Aug-MMR vs MMR on Movielens Non-metric Data

Algorithm	Running time (s)	Average Pruning
Aug-MMR	0.19	82.66%
MMR	0.52	0

Table 6.15 I-tree Maintenance on MakeBlobs 10k Records

$\|Y\|$	Algorithm	# updates	running time (s)
10	**OPTMn**	14	3.59
	GrMn	76	0.007
	NonOlMn	14	0.29
	NonIncrMn	2446	1.30
100	**OPTMn**	59	512.42
	GrMn	76	0.05
	NonOlMn	142	2.97
	NonIncrMn	2447	1.44
1000	**OPTMn**	59	18768.68
	GrMn	76	0.43
	NonOlMn	1068	34.58
	NonIncrMn	2449	1.45

features. We set $\lambda = 0.8$ and vary k from 10 to 25. By increasing k, **Aug-MMR** shows more scalability than MMR. **Aug-MMR** is about 1.7 times faster than the baseline implementation.

Finally, we run **Aug-MMR** on l more than 1 to show the efficiency of our proposed algorithm using multi-level **I-tree**. Table 6.7 shows that for l=2, **Aug-MMR** speedup is almost 4× for all dataset sizes.

Varying Parameters. We study the effect of different parameters on running time. Some parameters belong to the offline indexing algorithm, such as the number of levels (l) and arity of **I-tree** (m) and the total number of nodes (C). Other parameters are part of the online augmented algorithms. For example, k for the number of returned records and λ coefficient for **Aug-MMR** . In Figures 6.4, 6.6, 6.8, we vary parameters using **Yelp** dataset with a fixed size of 50000 records. In our experiment, optimum

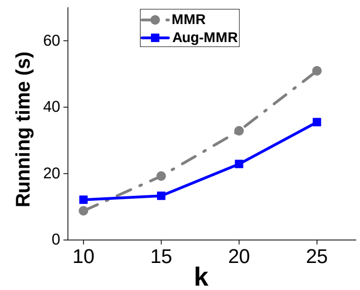


Figure 6.12 Aug-MMR vs MMR running time on UCI Gas data.

parameter settings for offline indexing are obtained by performing multiple runs and selecting the best. The index created using those parameter settings can be used in multiple runs of the online phase.

Varying k**.** Figures 6.4(a), 6.6(a), and 6.8(a) present how running time changes as we vary k from 5 to 50 for different baselines while fixing l, m, and λ to $1,500$, and 0.8, respectively. The running time increases quadratically for MMR and **Aug-MMR**, linearly for GMM and **Aug-GMM**, and in $\mathcal{O}(k * \log k)$ fashion for $SWAP$ and **Aug-SWAP**. These results are as consistent with our theoretical analysis, because of the presence of k^2 term in the MMR and **Aug-MMR**'s expected cost, k in GMM and **Aug-GMM**'s expected cost, and $k * \log k$ of that of $SWAP$ and **Aug-SWAP**. **Varying** m**.** Figures 6.4(c), 6.6(c), and 6.8(c) show the impact of varying m on the running time of the three algorithms. While varying m, we fix other parameters: $k = 20$, $l = 1$. The choice of m depends on the distribution of the dataset. As we increase m, the bounds for augmented algorithms become tighter while time for **DivGetBatch()** increases. We can see that there is a drop in running time and which indicates the optimum value for m for these three algorithms. For example, in **Aug-MMR** and **Aug-GMM**, the ideal value is $m = 500$ and for **Aug-SWAP**, it is $m = 100$.

Varying l**.** Figures 6.4(b), 6.6(b), and 6.8(b) show the impact of varying l on the running time of the three algorithms. We fix other parameters: $k = 20$,

Table 6.16 I-tree Maintenance Algorithm GrMn vs Construction from Scratch Algorithm NonIncrMn Running Time on MakeBlobs 10k Records

$\|Y\|$	Insertion Algorithm	Preprocessing time-offline (s)	query processing time-online (s)
10	**GrMn**	0.007	1.25
	NonIncrMn	1.30	0.55
100	**GrMn**	0.05	1.33
	NonIncrMn	1.44	0.60
1000	**GrMn**	0.43	1.96
	NonIncrMn	1.45	0.80
10000	**GrMn**	1.02	8.18
	NonIncrMn	4.65	1.61

and setting m to 2. C, the total number of nodes in **I-tree** becomes $2, 7, 15, 31, 63$, respectively for $l = 1, 2, 3, 4, 5$. In general, by fixing m and increasing l, C increases, and overall running time decreases. This is consistent with our theoretical analysis, as the expected running time contains a $1/C$ term.

Varying λ. Figures 6.4(d), 6.6(d), and 6.8(d) show that varying λ in *MMR* and **Aug-MMR** does not significantly change the running time. We have fixed $k = 20$, $l = 1$, and $m = 500$. The result is evident by observing the expected cost equations of *MMR* and **Aug-MMR** algorithms which do not contain a λ term. Though MR scores changes with λ, it has very little effect on the overall running time of *MMR* and **Aug-MMR** algorithms.

Varying diversity Functions Table 6.13 shows the results for **Aug-MMR** compared to MMR using different distance measures: euclidean distance measure, cosine similarity as general metric, and a non-metric distance function. Using 100k data from MakeBlobs dataset and m= 1000, l = 1 and number of features = 2, we can see that **Aug-MMR** performs $4\times$ better than MMR using both euclidean and cosine similarity metrics. For non-metric arbitrary distance function, the distance between records do not satisfy triangular inequality. Using this method, we see 15%

improvement, since the relevance and diversity scores are created arbitrarily and the result depends on the data distribution.

Table 6.12 shows overall comparison for **I-tree** and other baselines. SPP uses *KD-tree* as its index so we did not add it to the table. We can see that, unlike other baselines, **I-tree** can be used in non-metric functions and outperforms with 90% pruning of the original dataset.

Table 6.14 shows the results for **Aug-MMR** compared to MMR using non-metric distance function computed from MovieLens non-metric dataset. The total number of movies is 8,453, $\lambda = 0.8$, and $k = 20$. The diversity between a pair of items (movies) is calculated as the number of users that have rated either of those movies, but not both. Table 6.14 demonstrates that **Aug-MMR** outperforms *MMR* with 82.66% pruning of the original dataset, resulting in about 2.7 times speed up on an average.

Index construction and maintenance

Comparison with Baselines - Index Construction vs. Query Processing. In these set of experiments, we compare the index construction and query processing time trade-off of **I-tree** and compare that with of *KD-tree*, *Ball-tree*, and *M-tree* considering **Aug-MMR**. We adapt k-means and k-medoids [112] for building **I-tree** with number of iterations set to 300. The dataset that is used in this experiments is MakeBlobs. Figure 6.11 presents the **I-tree** speedup compared to other baselines for index preprocessing and query processing time. The results demonstrate that **I-tree** is always more than 18× faster in query processing and as much as 170× faster for certain configurations. For preprocessing, it is always more than 1.5× faster and at times it is more than 20× faster.

Index Construction. Now that it is obvious that **I-tree** outperforms the other indexing baselines, we further profile its efficacy.

In Figures 6.9(a) and (b), we vary dataset size and fix other parameters, $m = 1000$, $l = 1$. As we can observe in Figure 6.9(a), on the 100K **Yelp** dataset, indexing time is 172.69 seconds. In Figure 6.9(b), indexing time is 105 minutes on the 1M **MakeBlobs** dataset, and 109 minutes on the 1M **MovieLens**. Figures 6.9(c) and (d) show that the running time increases linearly when parameters m and l are systematically increased. In Figure 6.9(c), by varying l, we fix dataset size to 50000, and m to 2 (since $C = m^l$, by increasing l, the total number of nodes will increase). Finally, in Figure 6.9(d), we vary m, while fixing dataset size to 50000 and $l = 1$. These figures demonstrate that the preprocessing time increases linearly with varying parameters. **I-tree** takes 253 MB of space for 1M data with $m = 1000$ and $l = 1$.

Index Maintenance. For analyzing the index maintenance, we use two datasets, **MakeBlobs** and **MovieLens**. We compare **OPTMn** and its efficient counterpart **GrMn** with the baselines **NonOlMn**, and **NonIncrMn**. As expected, **OPTMn** has the least number of updates, but due to its inherent exponential nature, it does not scale beyond $10k$ dataset size with more than $|Y| = 1000$ records. Table 6.15 presents these results. We also see **GrMn**, even though not the optimal one, but stays consistently close to **OPTMn**. This table also shows that **GrMn** is better than the baselines in both running time and number of updates.Figures 6.10(a) and (b) present running time comparisons on very large datasets. **GrMn** is highly scalable, and the other two baselines take more time than **GrMn**. These results corroborate that **GrMn** is a suitable alternative to solve the index maintenance problem.

Incremental Index Maintenance vs Maintenance from Scratch. Table 6.16 shows comparison between **GrMn** and **NonIncrMn** index update algorithms. We present index preprocessing time in the offline phase, and query processing time in the online phase for the **Aug-MMR** algorithm. Clearly, **GrMn** requires smaller preprocessing time and higher query processing time compared to **NonIncrMn**. As it could be seen from Table 6.16, with 10,000 updates, the query processing time of **GrMn** becomes almost 5× slower than that of **NonIncrMn**. Contrarily, the preprocessing time of **GrMn** is about 4.5× faster than that of **NonIncrMn** at that setting. Since query processing time is more important and must be optimized, it seems, for 10,000 updates, it is better to build the index from scratch instead of maintaining it incrementally.

6.8 Related Work

6.8.1 Results diversification

Result diversification remains to be an active research topic with extensive applications in recommendation and search [1, 2, 4, 50, 86, 141, 157, 166, 167, 185, 188, 189, 193, 195], including very recent works that study diversity for fairness and popularity [147, 178, 214].

6.8.2 Content based algorithms

Content-based algorithms, which are our primary focus here, are of two kinds: *Interchange algorithms* first select k relevant records and then exchange selected records with remaining records to increase the overall diversity (*SWAP* [211] is an example). *Incremental greedy algorithms* iteratively build the top-k set by selecting the best record at each round. Notable examples of this latter kind are Maximal Marginal Relevance (*MMR*) method [55], Greedy Max-Min (*GMM*) [106], Max-Sum [105], IA-SELECT [6], and dLSH [1].

SPP [98] is a bounded diversification algorithm that produces same result as MMR while minimizing the number of accessed records. In [78], Drosou et al. introduce both greedy and interchange algorithms for the diversity over continuous data. In [80], the authors propose greedy algorithms for considering diversity over dynamic data by presenting *Insert* and *Delete* operations over the cover tree indexing structure. They also exploit the *GMM* algorithm for returning diversified top-k results. In [79], the authors propose greedy algorithms for diversity over a representative subset of objects, $DisC$, which is a mapping of the original data. They also present a degree of diversification, radius r, instead of k size results. Their proposed algorithms exploit the *M-tree* [68] indexing structure.

From a different perspective, one can categorize diversification algorithms into three groups: record-level, feature-level, and category-level. In record-level algorithms (*MMR*, *GMM*, and *SWAP*), the input is the distance value between records regardless of which record feature is more important. The score value is calculated based on an objective function that calculates distances/diversity. The inputs of feature-level algorithms are record features. Examples are DivGen and GenFilt [14]. The feature with the highest score is obtained from all records based on a ranking, and the goal is to skip some features and prune records having low scoring features. In the category-level algorithms, records are grouped into multiple categories. Such algorithms apply some constraints that will return no more than one or two records from the same category [3, 213].

6.8.3 Comparison with existing indexes

Compared to our proposed **I-tree**, existing indexing techniques are vector space based methods where coordinate information of the records are used to create data structures to answer a large spectrum of distance queries, where distance may be based on Euclidean, cosine similarity, general L_p norms, and so on. Popular solutions in low to moderate dimensional space include *K-B-D-tree* [170], *kd-tree* [34], *R-tree* [110], R^**-tree* [33], *SS-tree* [199] or more recent *X-tree* [35], *UB-tree* [31], *SR-tree* [125]. All these methods use the domain object feature vectors to measure the distance between objects and create a similarity index. As opposed to that, we consider the records to be atomic (and not a collection of vectors), and the diversity function could be metric or not. Therefore, these methods do not extend to solve our problem.

There exists other popular tree data structures like *Cover-tree* [36], *Ball-tree* [135] and *M-tree* [68] that are used for nearest neighbor search. Unlike our **I-tree**, these trees can only be used for metric distance functions.

In summary, we present an access primitive **DivGetBatch()** *that leverages a precomputed data structure* **I-tree** *to integrate MMR, GMM, and SWAP to expedite their processing time. The design of our primitive is independent of features and categories and is applicable with any distance measure, making it generic and useful. We study MMR, GMM, and SWAP, since we believe these are notable choices in the existing diversity literature space, and many more recent works adapt these algorithms [1, 22, 78–80, 117, 158, 169, 200, 201, 208].*

6.9 Conclusion

We propose an access primitive **DivGetBatch()** to expedite diversification algorithms while returning their exact top-k results. We present a computational framework to develop **DivGetBatch()** that contains a pre-computed index structure **I-tree** and describe how to rewire popular diversification algorithms using **DivGetBatch()**. Unlike existing indexes that primarily work on vector spaces (assuming the records have co-ordinates), we consider the records to be atomic as opposed to a collection of vectors. We make rigorous theoretical analysis of the exactness and running times of the augmented algorithms. We present principled solutions to maintain **I-tree** under batch updates. Our experiments on large real-world datasets corroborate our theoretical analysis, and show that our solution yields a 24× speedup on large datasets.

In the future, we are interested to study how to enable approximate top-k result diversification with guarantees leading to even faster running times. We also intend to explore how to adapt our proposed framework if diversity is assumed to satisfy metric property, in particular, the triangle inequality.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Summary

In this dissertation, we have addressed various aspects of promoting diverse and fair query results inside top-k and ranking. Our research covered a range of problems and presented practical solutions with provable guarantees. Firstly, we introduced the RAPF problem, which incorporates group fairness criteria (p-fairness) into the classical rank aggregation problem. We provided solutions for both binary and multi-valued protected attributes, demonstrating the effectiveness of our proposed methods through extensive experiments on real-world and synthetic datasets. Our work in this area opens up promising research directions, such as exploring alternative models and extending RAPF for top-k or considering incomplete information scenarios. Secondly, we tackled the margin finding problem in top-k preference aggregation models under single ballot substitutions, considering multiple protected group attributes to promote fairness. Our suite of algorithms with provable guarantees and rigorous experimental analysis demonstrated the effectiveness of our proposed solutions. Thirdly, we studied the suitability of Instant Run-off Voting (IRV) as a preference aggregation method for selecting k different winners that satisfy query constraints. Through formalization and optimization, we explored the margin finding problem and presented principled models and algorithms. Our experimental analyses supported the suitability of IRV as a preference aggregation method and sparked ongoing research on adapting AlgExact for single transferable voting schemes. Next, we introduced **θ-Equiv-top-k-MMSP** to redesign existing top-k algorithms for long-tail data to ensure fairness. The proposed method computed a set of top-k sets that are equivalent and assigned a probability distribution over these sets, promoting uniform selection probabilities for records in these sets. Our algorithmic results and experimental evaluations showcased the positive impact of our fairness notion on downstream applications and complementing group fairness considerations. Future research directions include exploring pre-processing techniques to expedite the computation of **θ-Equiv-top-k-Sets**. Lastly, we proposed an access primitive, DivGetBatch(), to expedite diversification algorithms while returning exact top-k results. We presented a computational framework with a pre-computed index structure (I-tree) and rewired popular diversification algorithms using DivGetBatch(). Our theoretical analysis and extensive experiments on real-world datasets confirmed the effectiveness and speedup achieved by our solution. Future research aims to enable approximate top-k result diversification with guarantees for even faster running times and adapting the framework to metric property assumptions. Overall, our dissertation contributes a comprehensive set of models and algorithms for promoting diverse and fair query results. The proposed solutions open up multiple interesting research directions for the ongoing investigation, addressing challenges and advancing the field of fair and diverse ranking and preference aggregation.

7.2 Future Work

As an ongoing problem, this research investigates how to select top-k features (predictors) for different subgroups considering datasets that are heavily incomplete.

Top-k Subgroup Feature Selection in Heavily Incomplete Datasets. Feature selection is the process of selecting attributes in the raw data that are highly informative to determine the class label, and is an important step in supervised learning [137,182,205]. While there exists multiple methods [140,179,191] for selecting attributes that are effective to determine the class label, the usefulness of these traditional methods heavily depends on the completeness of the underlying data. On the other hand, data incompleteness is a pervasive problem, especially when dealing with hard-to-reach subgroups, such as, racial minorities, ethnicities, and individuals from low socioeconomic backgrounds [44, 198]. Here, a subgroup is defined as a set of individuals who meet specific social demographic attribute criteria [97, 126]. Furthermore, data integration processes often lead to loss due to human error or the complexities of migration [155]. This ongoing work aims to address these challenges and proposes novel methodologies guided by machine learning techniques to generate top-k features for specific subgroups, even in the presence of substantial incompleteness. Given the incomplete nature of data, traditional techniques for feature selection like wrapper-based and filter-based [11, 43, 164] methods are not effective.

Running Example: credit risk analysis. Consider a database of $n = 1000$ candidates who have applied for a loan (Table 7.1). Each candidate has a set of 3 (x) demographic attributes: gender, ethnicity, and age. The combination of values of these attributes determines a subgroup. Additionally, the raw data contains $m = 100$ predictors of these candidates, such as income, credit history, education level, house rent, and more. Finally, there is a decision variable called "loan status" associated with each candidate. However, the dataset contains limited information of specific subgroups - as an example, Asian females over 40 years old are heavily underrepresented in the data. Only 30% of income data and 20% of credit history data is available for this subgroup. Furthermore, for the decision variable "Loan status," no data is present. The goal is to identify the top-k (=5) most important predictors for the Asian females over 40 years old. The missing class labels and the predictor values make the traditional feature selection process inapplicable to this setting.

Problem Definition 7. *Given a database D comprising m predictors $F : \{f_1, f_2, \ldots, f_m\}$, n records, and an additional class label or decision variable Z (continuous or discrete), a set of x attribute value combinations derives ℓ subgroups as $G = \{g_1, g_2, \ldots, g_\ell\}$.*

Table 7.1 Credit Risk Analysis Dataset Where Little Information About Asian Females Over Forty Years Old Present

# records	Group defining attributes			Predictors				Decision veriables
	Gender	Ethnicity	Age	Income	Credit history	...	House rent	Loan status
300	Male	White	Over 40	100% data present	100% data present	...	100% data present	100% data present
350	Male	White	Under 40	100% data present	100% data present	...	100% data present	100% data present
50	Female	Asian	Under 40	20% data present	30% data present	...	30% data present	0% data present
150	Female	white	Over 40	100% data present	100% data present	..	100% data present	100% data present
150	Male	Hispanic	Over 40	100% data present	100% data present	...	100% data present	100% data present

However, the data in D is heavily incomplete, particularly concerning Z or the predictors (F's) for subgroups that are underrepresented in the dataset. Given one such subgroup g_i, produce k features from F that are most useful to predict Z_i.

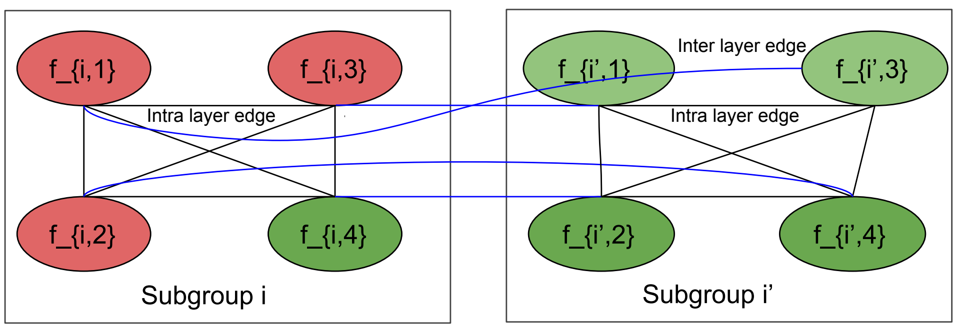


Figure 7.1 Multiplex graph for estimating feature importance of unknown (red) nodes.

Proposed Direction. Motivated by a recent work [103], which does entity resolution using machine learning approach, we explore how to develop a solution based on Graph Neural Network (GNN). The basic intuition of the approach is the following - capture informativeness (how much one can estimate about one variable if the other variable is known) between predictors inside and across subgroups, as well as that of the predictors and the class label via a multiplex graph [111] , which is a special type of a multi-relational graph, and estimates missing relationship values by using representation of the relationships that are currently present.

Quantifying informativeness. There are more than one way to quantify informativeness of two variables - either both are predictors inside or across subgroups, or one is a predictor and the other is a class label inside or across subgroups. As an initial approach, we consider Mutual Information (MI) [87, 174, 191] which captures information-theoretic "correlation" between two random variables and quantifies the amount of information obtained about one through the other. When f_1 and f_2 are discrete, $MI(f_1, f_2)$ is defined as follows: $P(f_1, f_2)\frac{P(f_1,f_2)}{P(f_1)\times P(f_2)}$. where $P(f_1, f_2)$ is the joint probability distribution function of f_1 and f_2, and $P(f_1)$ and $P(f_2)$ are the marginal probability distribution functions of f_1 and f_2 respectively.

Constructing the GNN. Formally, a multi-relational graph is a triplet $G = (V, E, R)$, where V is a set of nodes, E is a set of edges, and R is a set of typed edges that connect pairs of nodes. To construct an ℓ-layer graph, each layer represents a subgroup where a node $n_j^i \in V$ in each layer i represents a predictor f_j^i for subgroup g_i, the value of the node represents its "importance" to the class label Z_j^i. An edge $(n_j^i, n_{j'}^i, r)$ is a triplet, where $n_j^i, n_{j'}^i \in V$ and $r \in E$. We define two types of edges: $E = \{$ *inter-layer edges*, *intra-layer edges* $\}$. Inter-layer edges connect a node from subgroup i to its peer, the node representing the same feature, of subgroup i'. On the other hand, intra-layer edges connect a given node from layer i to its closest counterparts among the nodes of the same subgroup i. An intra-layer edge between two predictors f_j^i and $f_{j'}^i$ represents how much informative the two predictors are to each other for subgroup g_i, and an inter-layer edge between f_j^i and $f_j^{i'}$ represents how informative f_j^i is to learn about $f_j^{i'}$ (or vice versa). The weights of the edges are determined by Mutual Information (MI) [174], as described above (Figure 7.1).

Message propagation. The multiplex graph forms the foundation for employing a GNN model to acquire informativeness. In the context of GNNs, a "layer" refers to the process of receiving messages from neighboring nodes, aggregating these messages, and subsequently applying a fully connected neural network with an activation function. In the multiplex graph setting, a layer specifically denotes a collection of nodes corresponding to the same subgroup.

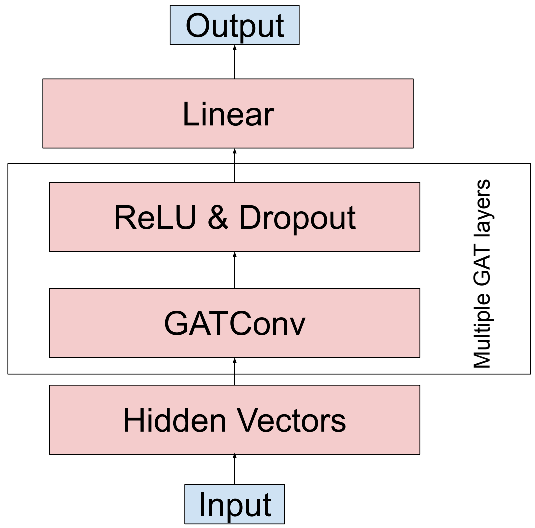


Figure 7.2 Proposed GNN architecture for generating feature importance.

A general GNN architecture is composed of q layers, each producing a hidden state vector, which is generated by aggregating the vectors of adjacent nodes. Using the multi-layer GNN, each node iteratively transmits its current information to itself and its neighboring nodes (connected by outgoing edges). Numerous GNN models have been introduced in recent years. In this work, we follow the model Graph Attention Network (GAT) [49, 190]. Initially, the hidden vectors $h = \{h_1, h_2, \ldots, h_m\}$ of nodes are created using the mutual information of the node and their closest counterparts. The GAT layer produces a new set of hidden vectors $h' = \{h'_1, h'_2, \ldots, h'_m\}$, as its output. A shared linear transformation, parametrized by a weight matrix, W is applied to every node. A shared attentional mechanism $\alpha_{j,j'}$ computes attention coefficients that indicate the importance of node j's features to node j'. The aggregated hidden vector is represented as: $h'_j = \sigma(\sum \alpha_{j,j'} W h_{j'})$, where σ is a nonlinear function.

Training the GNN. Figure 7.2 provides an overview of the GNN model architecture designed to predict the unknown mutual information value. The model comprises multiple GAT layers, each consisting of a GAT convolution, a ReLU activation function, and a dropout layer. The GATConv operation enables each node to attend to its neighbors using its own representation as a query, while the ReLU activation function introduces non-linearity, thereby enhancing the model's expressive capacity. The dropout layer ensures that each node experiences a stochastically sampled neighborhood. Subsequently, a linear layer, acting as a bias-free feed-forward neural network, is integrated into the architecture. This layer learns the average rate of correlation between the hidden vectors from the GAT layers and the target mutual information value, effectively fine-tuning the predictions based on the learned representations to achieve the desired mutual information estimation. During training, the model employs a Mean Square Error (MSE) loss function, defined as $MSE = \sum(\hat{MI}(f_i^j) - MI(f_i^j))^2$, where $\hat{MI}(f_i^j)$ and $MI(f_i^j)$ represent the true and predicted mutual information values of feature f_i^j, respectively. The model's primary goal is to predict the informativeness of unknown features, and as a result, the top-k highest-scored features are selected based on the model's predictions.

Currently, we are in the process of implementing and evaluating the effectiveness of the proposed model using different datasets and understanding further data management opportunities.

REFERENCES

[1] Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, and Sepideh Mahabadi. Real-time recommendation of diverse related articles. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1–12, 2013.

[2] Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, Sepideh Mahabadi, and Kasturi R Varadarajan. Diverse near neighbor problem. In *Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry*, pages 207–214, 2013.

[3] Zeinab Abbassi, Vahab S Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 32–40, 2013.

[4] Pankaj K Agarwal, Stavros Sintos, and Alex Steiger. Efficient indexes for diverse top-k range queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 213–227, 2020.

[5] Akanksha Agrawal, Grzegorz Guspiel, Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Connecting the dots (with minimum crossings). In *35th International Symposium on Computational Geometry (SoCG 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[6] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14, 2009.

[7] 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.

[8] Nir Ailon. Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica*, 57(2):284–300, 2010.

[9] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

[10] Airbnb. Dataset, san francisco, ca, 2023. Available at: `http://insideairbnb.com/get-the-data`, retrieved on 4/7/2023.

[11] Mohammed A Ambusaidi, Xiangjian He, Priyadarsi Nanda, and Zhiyuan Tan. Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE Transactions on Computers*, 65(10):2986–2998, 2016.

[12] Sihem Amer-Yahia, Behrooz Omidvar-Tehrani, Senjuti Basu, and Nafiseh Shabib. Group recommendation with temporal affinities. In *International Conference on Extending Database Technology (EDBT)*, 2015.

[13] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *Proceedings of the VLDB Endowment*, 2(1):754–765, 2009.

[14] Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 781–792, 2011.

[15] Anonymous. Git link. `https://anonymous.4open.science/r/selection_queries_using_irv-5AD0/README.md`, 2023.

[16] Robert Armstrong. The long tail: Why the future of business is selling less of more. *Canadian Journal of Communication*, 33(1):127, 2008.

[17] Kenneth J Arrow. A difficulty in the concept of social welfare. *Journal of Political Economy*, 58(4):328–346, 1950.

[18] Abolfazl Asudeh, HV Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1259–1276, 2019.

[19] Martin Aumuller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. Fair near neighbor search via sampling. *ACM SIGMOD Record*, 50(1):42–49, 2021.

[20] Manel Ayadi, Nahla Ben Amor, Jérôme Lang, and Dominik Peters. Single Transferable Vote: Incomplete Knowledge and Communication Issues. In *18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 19)*, pages 1288–1296, Montreal QC, Canada, May 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[21] Pieter Cornelis Baayen and Z Hedrlin. *On the existence of well distributed sequences in compact spaces*. Stichting Mathematisch Centrum. Zuivere Wiskunde, 1964.

[22] Krisztian Balog, Filip Radlinski, and Shushan Arakelyan. Transparent, scrutable and explainable user models for personalized recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 265–274, 2019.

[23] Linas Baltrunas, Tadas Makcinskas, and Francesco Ricci. Group recommendations with rank aggregation and collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, pages 119–126, 2010.

[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] Jacob P. Baskin and Shriram Krishnamurthi. Preference aggregation in group recommender systems for committee decision-making. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, page 337–340, New York, NY, USA, 2009. Association for Computing Machinery.

[29] Senjuti Basu Roy and Kaushik Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 361–372, 2011.

[30] Senjuti Basu Roy, Laks VS Lakshmanan, and Rui Liu. From group recommendations to group formation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1603–1616, 2015.

[31] Rudolf Bayer. The universal b-tree for multidimensional indexing: General concepts. In *International Conference on Worldwide Computing and Its Applications*, pages 198–209. Springer, 1997.

[32] Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141, 1970.

[33] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[34] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975.

[35] S Berchtold, D Keim, and HP Kriegel. The X-tree: An efficient and robust access method for points and rectangles. In *Proc. 1996 International Conference Very Large Data Bases*, pages 28–39, 1996.

[36] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, 2006.

[37] Arnab Bhattacharyya and Palash Dey. Predicting winner and estimating margin of victory in elections using sampling. *Artificial Intelligence*, 296:103476, 2021.

[38] Asia J Biega, Krishna P Gummadi, and Gerhard Weikum. Equity of attention: Amortizing individual fairness in rankings. In *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 405–414, 2018.

[39] Reuben Binns. On the apparent conflict between individual and group fairness. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 514–524, 2020.

[40] Julian H Blau. The existence of social welfare functions. *Econometrica: Journal of the Econometric Society*, pages 302–313, 1957.

[41] Michelle Blom, Peter J Stuckey, and Vanessa J Teague. Towards computing victory margins in stv elections. *arXiv preprint arXiv:1703.03511*, 2017.

[42] Michelle Blom, Peter J Stuckey, Vanessa J Teague, and Ron Tidhar. Efficient computation of exact irv margins. *arXiv preprint arXiv:1508.04885*, 2015.

[43] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1):245–271, 1997. Relevance.

[44] Anneliese C Bolland, Sara Tomek, John M Bolland, et al. Does missing data in studies of hard-to-reach populations bias results? not necessarily. *Open Journal of Statistics*, 7(02):264, 2017.

[45] JC de Borda. Mémoire sur les élections au scrutin. *Histoire de l'Academie Royale des Sciences pour 1781 (Paris, 1784)*, 1784.

[46] Steven J Brams and Peter C Fishburn. Approval voting. *American Political Science Review*, 72(3):831–847, 1978.

[47] 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.

[48] Gerth Stølting Brodal, Loukas Georgiadis, Kristoffer Arnsfelt Hansen, and Irit Katriel. Dynamic matchings in convex bipartite graphs. In *International Symposium on Mathematical Foundations of Computer Science*, pages 406–417. Springer, 2007.

[49] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[50] Zhi Cai, Georgios Kalamatianos, Georgios J Fakas, Nikos Mamoulis, and Dimitris Papadias. Diversified spatial keyword search on rdf data. *The VLDB Journal*, 29(5):1171–1189, 2020.

[51] Donald E Campbell and Jerry S Kelly. Information and preference aggregation. *Social Choice and Welfare*, 17(1):3–24, 2000.

[52] Donald E Campbell and Jerry S Kelly. Impossibility theorems in the arrovian framework. *Handbook of Social Choice and Welfare*, 1:35–94, 2002.

[53] David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S Jensen. Unsupervised time series outlier detection with diversity-driven convolutional ensembles–extended version. *arXiv preprint arXiv:2111.11108*, 2021.

[54] Da Cao, Xiangnan He, Lianhai Miao, Yahui An, Chao Yang, and Richang Hong. Attentive group recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 645–654, 2018.

[55] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336, 1998.

[56] David Cary. Estimating the margin of victory for instant-runoff voting. In *Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, San Francisco, CA, 2011. USENIX Association.

[57] Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth Vishnoi. Fair and diverse dpp-based data summarization. In *International Conference on Machine Learning*, pages 716–725. PMLR, 2018.

[58] L Elisa Celis, Amit Deshpande, Tarun Kathuria, and Nisheeth K Vishnoi. How to be fair and diverse? *arXiv preprint arXiv:1610.07183*, 2016.

[59] L Elisa Celis, Lingxiao Huang, and Nisheeth K Vishnoi. Multiwinner voting with fairness constraints. *arXiv preprint arXiv:1710.10057*, 2017.

[60] L Elisa Celis, Damian Straszak, and Nisheeth K Vishnoi. Ranking with fairness constraints. *arXiv preprint arXiv:1704.06840*, 2017.

[61] Abhijnan Chakraborty, Aniko Hannak, Asia J Biega, and Krishna Gummadi. Fair sharing for sharing economy platforms. In *Fairness, Accountability and Transparency in Recommender Systems-Workshop on Responsible Recommendation*, 2017.

[62] Abhijnan Chakraborty, Gourab K. Patro, Niloy Ganguly, Krishna P. Gummadi, and Patrick Loiseau. Equality of voice: Towards fair representation in crowdsourced top-k recommendations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 129–138, New York, NY, USA, 2019. Association for Computing Machinery.

[63] Christopher P Chambers and Takashi Hayashi. Preference aggregation under uncertainty: Savage vs. pareto. *Games and Economic Behavior*, 54(2):430–440, 2006.

[64] Danny Z Chen, Xiaomin Liu, and Haitao Wang. Computing maximum non-crossing matching in convex bipartite graphs. *Discrete Applied Mathematics*, 187:50–60, 2015.

[65] Junghoo Cho, Sourashis Roy, and Robert E. Adams. Page quality: In search of an unbiased web ranking. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, page 551–562, New York, NY, USA, 2005. Association for Computing Machinery.

[66] Yong-Gon Cho and Keun-Tae Cho. A loss function approach to group preference aggregation in the ahp. *Computers & Operations Research*, 35(3):884–892, 2008.

[67] Alexandra Chouldechova and Aaron Roth. The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810*, 2018.

[68] Paolo Ciaccia et al. M-tree: An efficient access method for similarity search in metric spaces. In *Very Large Data Bases*, volume 97, pages 426–435, 1997.

[69] Marquis de Condorcet. Essay on the application of analysis to the probability of majority decisions. *Paris: Imprimerie Royale*, 1785.

[70] Vincent Conitzer, Andrew Davenport, and Jayant Kalagnanam. Improved bounds for computing kemeny rankings. In *AAAI*, volume 6, pages 620–626, 2006.

[71] Vincent Conitzer, Tuomas Sandholm, and Jérôme Lang. When are elections with few candidates hard to manipulate? *J. ACM*, 54(3):14:1–14:33, Jun 2007.

[72] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, 2009.

[73] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. Answering top-k queries using views. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 451–462, 2006.

[74] 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.

[75] Gil Delannoi and Oliver Dowlen. *Sortition: Thoery and Practice*, volume 3. Andrews UK Limited, 2016.

[76] Palash Dey and Y. Narahari. Estimating the margin of victory of an election using sampling, 2015.

[77] 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.

[78] Marina Drosou and Evaggelia Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4):49–56, 2009.

[79] Marina Drosou and Evaggelia Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *arXiv preprint arXiv:1208.3533*, 2012.

[80] Marina Drosou and Evaggelia Pitoura. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1102–1116, 2013.

[81] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 214–226, 2012.

[82] Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, pages 613–622, 2001.

[83] Jon Elster and Aanund Hylland. *Foundations of social choice theory*. Cambridge University Press, Cambridge, UK, 1989.

[84] Peter Emerson. The original borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358, 2013.

[85] Ulle Endriss. Lecture notes on fair division. *arXiv preprint arXiv:1806.04234*, 2018.

[86] Mohammadreza Esfandiari, Ria Mae Borromeo, Sepideh Nikookar, Paras Sakharkar, Sihem Amer-Yahia, and Senjuti Basu Roy. Multi-session diversity to improve user satisfaction in web applications. In *Proceedings of the Web Conference 2021*, pages 1928–1936, 2021.

[87] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201, 2009.

[88] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal of Discrete Mathematics*, 20(3):628–648, 2006.

[89] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[90] Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner voting: A new challenge for social choice theory. *Trends in Computational Social Choice*, 74(2017):27–47, 2017.

[91] David M Farrell, Malcolm Mackerras, and Ian McAllister. Designing electoral institutions: Stv systems and their consequences. *Political Studies*, 44(1):24–43, 1996.

[92] Benjamin Fish, Ashkan Bashardoust, Danah Boyd, Sorelle Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. Gaps in information access in social networks? In *The World Wide Web Conference*, pages 480–490, 2019.

[93] Peter C Fishburn. Arrow's impossibility theorem: Concise proof and infinite voters. *Journal of Economic Theory*, 2(1):103–106, 1970.

[94] Bailey Flanigan, Paul Gölz, Anupam Gupta, Brett Hennig, and Ariel D Procaccia. Fair algorithms for selecting citizens' assemblies. *Nature*, 596(7873):548–552, 2021.

[95] Will Fleisher. What's fair about individual fairness? In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 480–490, 2021.

[96] 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.

[97] James R Foulds, Rashidul Islam, Kamrun Naher Keya, and Shimei Pan. An intersectional definition of fairness. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1918–1921. IEEE, 2020.

[98] Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. Top-k bounded diversification. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 421–432, 2012.

[99] Sorelle A Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. The (im) possibility of fairness: Different value systems require different mechanisms for fair decision making. *Communications of the ACM*, 64(4):136–143, 2021.

[100] William A Gale and Geoffrey Sampson. Good-turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995.

[101] David García-Soriano and Francesco Bonchi. Maxmin-fair ranking: individual fairness under group-fairness constraints. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 436–446, 2021.

[102] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[103] Bar Genossar, Roee Shraga, and Avigdor Gal. Flexer: Flexible entity resolution for multiple intents. *Proc. ACM Manag. Data*, 1(1), may 2023.

[104] Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. Fairness-aware ranking in search and recommendation systems with application to linkedin talent search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2221–2231, 2019.

[105] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th International Conference on World Wide Web*, pages 381–390, 2009.

[106] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[107] Goetz Graefe et al. Modern b-tree techniques. *Foundations and Trends® in Databases*, 3(4):203–402, 2011.

[108] Bernard Grofman, Guillermo Owen, and Scott L Feld. Thirteen theorems in search of the truth. *Theory and Decision*, 15(3):261–278, 1983.

[109] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems (TODS)*, 28(2):140–174, 2003.

[110] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[111] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[112] Jiawei Han, Micheline Kamber, and Jian Pei. Data mining concepts and techniques third edition. *The Morgan Kaufmann Series in Data Management Systems*, 5(4):83–124, 2011.

[113] Wm. H. Hare. Application of mr. hare's system of voting to the nomination of overseers of harvard college. *Journal of Social Science: Containing the Transactions of the American Social Science Association*, 3-4:192–198, 1871.

[114] Hao He, Haixun Wang, Jun Yang, and Philip S Yu. Blinks: ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 305–316, 2007.

[115] Corinna Hertweck, Christoph Heitz, and Michele Loi. On the moral justification of statistical parity. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 747–757, 2021.

[116] Steven Hill and Robert Richie. Success for instant runoff voting in san francisco. *National Civic Review*, 94(1):65–69, 2005.

[117] Tom Hope, Joel Chan, Aniket Kittur, and Dafna Shahaf. Accelerating innovation through analogy mining. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 235–243, 2017.

[118] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):1–58, 2008.

[119] IMDB. Dataset, 2023. Available at: `https://www.kaggle.com/datasets/isaactaylorofficial/imdb-10000-most-voted-feature-films-041118`, Retrieved: 4/7/2023.

[120] Md Mouinul Islam, 2023. Codes and data are available at: `https://anonymous.4open.science/r/FairSelectionInsideTopk-2F4F/README.md`, Retrieved on 4/7/2023.

[121] Md Mouinul Islam, Dong Wei, Baruch Schieber, and Senjuti Basu Roy. Satisfying complex top-k fairness constraints by preference substitutions. *Proceedings of the VLDB Endowment*, 16(2):317–329, 2022.

[122] Alborz Jelvani and Amelie Marian. Identifying possible winners in ranked choice voting elections with outstanding ballots. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 10(1):114–123, Oct. 2022.

[123] Zhongjun Jin, Mengjing Xu, Chenkai Sun, Abolfazl Asudeh, and HV Jagadish. Mithracoverage: a system for investigating population bias for intersectional fairness. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2721–2724, 2020.

[124] Kaggle. Top-1000 IMDB Movies. `https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows,Retrievedon4/7/2023`.

[125] Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. *ACM SIGMOD Record*, 26(2):369–380, 1997.

[126] Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. An empirical study of rich subgroup fairness for machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. ACM, jan 2019.

[127] John G Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[128] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[129] Jae Kyeong Kim, Hyea Kyeong Kim, Hee Young Oh, and Young U Ryu. A group recommendation system for online communities. *International Journal of Information Management*, 30(3):212–219, 2010.

[130] Craig W Kirkwood and Rakesh K Sarin. Ranking with partial information: A method and an application. *Operations Research*, 33(1):38–48, 1985.

[131] Donald E. Knuth. *The Art of Computer Programming*, volume 1 of *Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., 3rd edition, 1998. (book).

[132] Yehuda Koren and Joe Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 117–124, 2011.

[133] Iordanis Koutsopoulos and Maria Halkidi. Efficient and fair item coverage in recommender systems. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 912–918. IEEE, 2018.

[134] Caitlin Kuhlman and Elke Rundensteiner. Rank aggregation algorithms for fair consensus. *Proceedings of the VLDB Endowment*, 13(12), 2020.

[135] Neeraj Kumar et al. What is a good nearest neighbors algorithm for finding similar patches in images? In *European Conference on Computer Vision*, pages 364–378. Springer, 2008.

[136] Jean-François Laslier. And the loser is… plurality voting. In *Electoral Systems*, pages 327–351. Springer, 2012.

[137] David D. Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, page 212–217, USA, 1992. Association for Computational Linguistics.

[138] Chang Li, Haoyun Feng, and Maarten de Rijke. Cascading hybrid bandits: Online learning to rank for relevance and diversity. In *RecSys 2020: The ACM Conference on Recommender Systems*, pages 33–42. ACM, September 2020.

[139] Yunqi Li, Hanxiong Chen, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. User-oriented fairness in recommendation. In *Proceedings of the Web Conference 2021*, pages 624–632, 2021.

[140] Pasi Luukka. Feature selection using fuzzy entropy measures with similarity classifier. *Expert Systems with Applications*, 38(4):4600–4607, 2011.

[141] Rischan Mafrur, Mohamed A Sharaf, and Hina A Khan. Dive: diversifying view recommendation for visual data exploration. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1123–1132, 2018.

[142] Thomas Magrino, Ronald Rivest, Emily Shen, and David Wagner. Computing the margin of victory in irv elections. In *2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 11)*, pages 4–4, San Francisco, CA, 08 2011. USENIX Association.

[143] Sepideh Mahabadi and Ali Vakilian. Individual fairness for k-clustering. In *International Conference on Machine Learning*, pages 6586–6596. PMLR, 2020.

[144] Makeblobs. Dataset, 2023. Available at: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`.

[145] Colin L Mallows. Non-null ranking models. i. *Biometrika*, 44(1/2):114–130, 1957.

[146] Federico Malucelli, Thomas Ottmann, and Daniele Pretolani. Efficient labelling algorithms for the maximum noncrossing matching problem. *Discrete Applied Mathematics*, 47(2):175–179, 1993.

[147] Stella Maropaki, Sean Chester, Christos Doulkeridis, and Kjetil Nørvåg. Diversifying top-k point-of-interest queries via collective social reach. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, pages 2149–2152, 2020.

[148] Christopher McComb, Kosa Goucher-Lambert, and Jonathan Cagan. Impossible by design? fairness, strategy, and arrow's impossibility theorem. *Design Science*, 3, 2017.

[149] Eamon McGinn. Rating rankings: Effect of instant run-off voting on participation and civility. *Unpublished manuscript. Retrieved from http://eamonmcginn.com/papers/IRV_in_Minneapolis. pdf*, 2020.

[150] Iain McLean. The borda and condorcet principles: three medieval applications. *Social Choice and Welfare*, 7(2):99–108, 1990.

[151] Reshef Meir. Plurality voting under uncertainty. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[152] Reshef Meir, Maria Polukarov, Jeffrey Rosenschein, and Nicholas Jennings. Convergence to equilibria in plurality voting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 823–828, 2010.

[153] Alberto O Mendelzon and Tova Milo. Formal models of web queries. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 134–143, 1997.

[154] Samuel Merrill III. A comparison of efficiency of multicandidate electoral systems. *American Journal of Political Science*, pages 23–48, 1984.

[155] Ardalan Mirzaei, Stephen R Carter, Asad E Patanwala, and Carl R Schneider. Missing data in surveys: Key concepts, approaches, and applications. *Research in Social and Administrative Pharmacy*, 18(2):2308–2316, 2022.

[156] Kyriakos Mouratidis. Geometric aspects and auxiliary features to top-k processing. In *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, volume 2, pages 1–3. IEEE, 2016.

[157] Sepideh Nikookar, Mohammadreza Esfandiari, Ria Mae Borromeo, Paras Sakharkar, Sihem Amer-Yahia, and Senjuti Basu Roy. Diversifying recommendations on sequences of sets. *The Very Large Data Bases Journal*, pages 1–22, 2022.

[158] Francisco Parreño, Ramón Álvarez-Valdés, and Rafael Martí. Measuring diversity. a review and an empirical analysis. *European Journal of Operational Research*, 289(2):515–532, 2021.

[159] Gourab K Patro, Arpita Biswas, Niloy Ganguly, Krishna P Gummadi, and Abhijnan Chakraborty. Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms. In *Proceedings of The Web Conference 2020*, pages 1194–1204, 2020.

[160] David M. Pennock, Eric Horvitz, and C. Lee Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, page 729–734. AAAI Press, 2000.

[161] Lionel S Penrose. The elementary statistics of majority voting. *Journal of the Royal Statistical Society*, 109(1):53–57, 1946.

[162] Evaggelia Pitoura, Kostas Stefanidis, and Georgia Koutrika. Fairness in rankings and recommendations: an overview. *The VLDB Journal*, pages 1–28, 2021.

[163] Evaggelia Pitoura, Panayiotis Tsaparas, Giorgos Flouris, Irini Fundulaki, Panagiotis Papadakos, Serge Abiteboul, and Gerhard Weikum. On measuring bias in online information. *ACM SIGMOD Record*, 46(4):16–21, 2018.

[164] R Porkodi. Comparison of filter based feature selection algorithms: An overview. *International journal of Innovative Research in Technology & Science*, 2(2):108–113, 2014.

[165] Abinash Pujahari and Dilip Singh Sisodia. Aggregation of preference relations to enhance the ranking quality of collaborative filtering based group recommender system. *Expert Systems with Applications*, 156:113476, 2020.

[166] Shameem A Puthiya Parambath, Nicolas Usunier, and Yves Grandvalet. A coverage-based approach to recommendation diversity on similarity graph. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 15–22, 2016.

[167] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *arXiv preprint arXiv:1208.0076*, 2012.

[168] Davood Rafiei, Krishna Bharat, and Anand Shukla. Diversifying web search results. In *The Web Conference*, 2010.

[169] Pengjie Ren, Zhumin Chen, Zhaochun Ren, Furu Wei, Jun Ma, and Maarten de Rijke. Leveraging contextual sentence relations for extractive summarization using a neural attention model. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 95–104, 2017.

[170] John T Robinson. The KDB-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, 1981.

[171] Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Aggregating preferences cannot be fair. *Intelligenza Artificiale*, 2(1):30–38, 2005.

[172] Senjuti Basu Roy. Returning top-k: Preference aggregation or sortition, or is there a better middle ground? *SIGMOD Blog*, 2022.

[173] Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, Gautam Das, and Cong Yu. Exploiting group recommendation functions for flexible preferences. In *2014 IEEE 30th International Conference on Data Engineering*, pages 412–423, Chicago, IL, USA, 2014. IEEE.

[174] Md Abdus Salam, Senjuti Basu Roy, and Gautam Das. Efficient approximate top-k mutual information based feature selection. *Journal of Intelligent Information Systems*, pages 1–33, 10 2022.

[175] Babak Salimi, Bill Howe, and Dan Suciu. Database repair meets algorithmic fairness. *ACM SIGMOD Record*, 49(1):34–41, 2020.

[176] 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.

[177] Amartya Sen. Social choice theory. *Handbook of Mathematical Economics*, 3:1073–1181, 1986.

[178] Ashudeep Singh and Thorsten Joachims. Fairness of exposure in rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2219–2228, 2018.

[179] Robinson Spencer, Fadi Thabtah, Neda Abdelhamid, and Michael Thompson. Exploring feature selection and classification methods for predicting heart disease. *Digital health*, 6:2055207620914777, 2020.

[180] Peter Stone. Sortition, voting, and democratic equality. *Critical Review of International Social and Political Philosophy*, 19(3):339–356, 2016.

[181] Julia Stoyanovich, Ke Yang, and HV Jagadish. Online set selection with fairness and diversity constraints. In *Proceedings of the EDBT Conference*, 2018.

[182] Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. In *Data Classification: Algorithms and Applications*, 2014.

[183] Nicolaus Tideman. The single transferable vote. *Journal of Economic Perspectives*, 9(1):27–38, 1995.

[184] Robert Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32(3):323–330, 1980.

[185] Chun-Hua Tsai and Peter Brusilovsky. Beyond the ranked list: User-driven exploration and diversification of social recommendation. In *23rd International Conference on Intelligent User Interfaces*, pages 239–250, 2018.

[186] Anke Van Zuylen and David P Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *International Workshop on Approximation and Online Algorithms*, pages 260–273. Springer, 2007.

[187] 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.

[188] Saúl Vargas, Linas Baltrunas, Alexandros Karatzoglou, and Pablo Castells. Coverage, redundancy and size-awareness in genre diversity for recommender systems. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 209–216, 2014.

[189] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM Conference on Recommender Systems*, pages 109–116, 2011.

[190] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[191] Jorge R Vergara and Pablo A Estévez. A review of feature selection methods based on mutual information. *Neural Computing and Applications*, 24:175–186, 2014.

[192] Sanne Vrijenhoek, Gabriel Bénédict, Mateo Gutierrez Granada, Daan Odijk, and Maarten de Rijke. Radio – rank-aware divergence metrics to measure normative diversity in news recommendation. In *RecSys 2022: The ACM Conference on Recommender Systems*. ACM, September 2022.

[193] Dongjing Wang, Shuiguang Deng, and Guandong Xu. Sequence-based context-aware music recommendation. *Information Retrieval Journal*, 21(2-3):230–252, 2018.

[194] Haixun Wang and Charu C Aggarwal. A survey of algorithms for keyword search on graph data. *Managing and Mining Graph Data*, pages 249–273, 2010.

[195] Lina Wang, Xuyun Zhang, Tian Wang, Shaohua Wan, Gautam Srivastava, Shaoning Pang, and Lianyong Qi. Diversified and scalable service recommendation with accuracy guarantee. *IEEE Transactions on Computational Social Systems*, 8(5):1182–1193, 2020.

[196] Yue Wang, Alexandra Meliou, and Gerome Miklau. Rc-index: Diversifying answers to range queries. *Proceedings of the VLDB Endowment*, 11(7):773–786, 2018.

[197] 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.

[198] Bruce Western, Anthony Braga, David Hureau, and Catherine Sirois. Study retention as bias reduction in a hard-to-reach population. *Proceedings of the National Academy of Sciences*, 113(20):5477–5485, 2016.

[199] David A White and Ramesh Jain. Similarity indexing with the ss-tree. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 516–523. IEEE, 1996.

[200] Wen Wu, Li Chen, and Yu Zhao. Personalizing recommendation diversity based on user personality. *User Modeling and User-Adapted Interaction*, 28(3):237–276, 2018.

[201] Yingying Wu, Yiqun Liu, Fei Chen, Min Zhang, and Shaoping Ma. Beyond greedy search: pruned exhaustive search for diversified result ranking. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 99–106, 2018.

[202] Lirong Xia. Computing the margin of victory for various voting rules. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC '12)*, page 982–999, New York, NY, USA, 2012.

[203] Himank Yadav, Zhengxiao Du, and Thorsten Joachims. Fair learning-to-rank from implicit feedback. In *SIGIR*, 2020.

[204] Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 344–353, 2004.

[205] Howard Hua Yang and John Moody. Data visualization and feature selection: New algorithms for nongaussian data. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 687–693, Cambridge, MA, USA, 1999. MIT Press.

[206] Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. Balanced ranking with diversity constraints. *arXiv preprint arXiv:1906.01747*, 2019.

[207] Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *Proceedings of The 29th International Conference on Scientific and Statistical Database Management*, pages 1–6, 2017.

[208] Jin-ge Yao, Xiaojun Wan, and Jianguo Xiao. Recent advances in document summarization. *Knowledge and Information Systems*, 53(2):297–336, 2017.

[209] Yelp. Dataset, 2023. Available at: `https://www.yelp.com/dataset/documentation/main`.

[210] Peyton Young. Optimal voting rules. *Journal of Economic Perspectives*, 9(1):51–64, 1995.

[211] Cong Yu, Laks Lakshmanan, and Sihem Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 368–378, 2009.

[212] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness beyond disparate treatment and disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1171–1180, 2017.

[213] Michele Zanitti, Sokol Kosta, and Jannick Sørensen. A user-centric diversity by design recommender system for the movie application domain. In *Companion Proceedings of the The Web Conference 2018*, pages 1381–1389, 2018.

[214] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. FA*IR: A fair top-k ranking algorithm. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1569–1578, 2017.

[215] Meike Zehlike, Ke Yang, and Julia Stoyanovich. Fairness in ranking, part ii: Learning-to-rank and recommender systems. *ACM Computing Surveys*, 55(6):1–41, 2022.

[216] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *International Conference on Machine Learning*, pages 325–333. PMLR, 2013.

[217] Hantian Zhang, Xu Chu, Abolfazl Asudeh, and Shamkant B Navathe. Omnifair: A declarative system for model-agnostic group fairness in machine learning. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2076–2088, 2021.

[218] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. Crowdsourced top-k algorithms: An experimental evaluation. *Proceedings of the VLDB Endowment*, 9(8):612–623, 2016.