Satisfying Complex Top-k Fairness Constraints by Preference Substitutions

Md. Mouinul Islam
Dept. of Computer Science, NJIT, NJ, USA
mi257@njit.edu

Dong Wei
Dept. of Computer Science, NJIT, NJ, USA
dw277@njit.edu

Baruch Schieber
Dept. of Computer Science, NJIT, NJ, USA
sbar@njit.edu

Senjuti Basu Roy
Dept. of Computer Science, NJIT, NJ, USA
senjutib@njit.edu

ABSTRACT

Given m users (voters), where each user casts her preference for a single item (candidate) over n items (candidates) as a ballot, the preference aggregation problem returns k items (candidates) that have the k highest number of preferences (votes). Our work studies this problem considering *complex fairness constraints* that have to be satisfied via proportionate representations of different values of the group protected attribute(s) in the top-k results. Precisely, we study *the margin finding problem under single ballot substitutions*, where a single substitution amounts to removing a vote from candidate i and assigning it to candidate j and the goal is to *minimize the number of single ballot substitutions needed to guarantee that the top-k results satisfy the fairness constraints*. We study several variants of this problem considering how top-k fairness constraints are defined, (i) MFBinaryS and MFMultiS are defined when the fairness (proportionate representation) is defined over a single, binary or multivalued, protected attribute, respectively; (ii) MFMulti2 is studied when top-k fairness is defined over two different protected attributes; (iii) MFMulti3+ investigates the margin finding problem, considering 3 or more protected attributes. We study these problems theoretically, and present a suite of algorithms with provable guarantees. We conduct rigorous large scale experiments involving multiple real world datasets by appropriately adapting multiple state-of-the-art solutions to demonstrate the effectiveness and scalability of our proposed methods.

PVLDB Reference Format:
Md. Mouinul Islam, Dong Wei, Baruch Schieber, and Senjuti Basu Roy. Satisfying Complex Top-k Fairness Constraints by Preference Substitutions. PVLDB, 16(2): 317 - 329, 2022.
doi:10.14778/3565816.3565832

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/MouinulIslamNJIT/BallotChange.git.

Proceedings of the VLDB Endowment, Vol. 16, No. 2 ISSN 2150-8097.
doi:10.14778/3565816.3565832

1 INTRODUCTION

Preference aggregation is important in finding top-k outputs that represent plurality preference [34] and has wide variety of applications in recommender systems, search results listing [7], electoral systems [33, 35], or allocating resources among candidates, such as, in hiring or admission [46]. 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 [13], 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 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 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**

and 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 [25]. 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 [25]. 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 1. Our final contribution is experimental **(Section 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 [22, 42] 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 7**.

Table 1: Summary of technical results

Problem	Protected Attribute	Hardness	Algorithm	Approx Factor	Running Time
MFBinaryS	single attribute binary valued	p-time	Alg1AttBOpt	exact	$O(n \log n)$
MFMultiS	single attribute multi (ℓ) valued	p-time	Alg1AttMOpt	exact	$O(n \log n)$
MFMulti2	2 attributes ℓ possible values	Weak NP-hard	Alg2AttApx	2	$O(n^2 \ell \log m)$
MFMulti3+	3+ attributes	NP-hard			
MFMulti2 MFMulti3+	2+ attributes const size (c) of Cartesian prod	p-time	AlgCartOpt	exact	$O(n^{c+1})$

2 DATA MODEL & PROBLEM DEFINITIONS

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

2.1 A Toy Running Example

Table 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 2: 12 voters, 6 candidates, and a voting outcome

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	$\sum V_i$
C1 (*M,Sr,si*)	1	1	0	1	0	0	1	0	0	0	0	0	4
C2 (*M,Jr,si*)	0	0	1	0	1	0	0	0	1	0	0	0	3
C3 (*M,Jr,ma*)	0	0	0	0	0	1	0	0	0	0	0	1	2
C4 (*F,Jr,si*)	0	0	0	0	0	0	0	0	0	1	1	0	2
C5 (*F,Jr,ma*)	0	0	0	0	0	0	0	1	0	0	0	0	1
C6 (*F,Sr,di*)	0	0	0	0	0	0	0	0	0	0	0	0	0

An Example Complex fairness constraint. Imagine the goal is to select $k = 4$ candidates from the voting outcome described in Table 2 with the following fairness constraints described in Table 3.

Table 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 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, $\Pi_{i=1}^{\ell} \ell_i$

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 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 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 [27, 32, 42] Fairness Constraints. A fairness constraint defined over a single protected attribute containing ℓ different

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

groups $G_1, G_2,..,G_\ell$, 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. Based on this, $\{C1, C3, C5, C6\}$ is a feasible top-4 outcome, as it satisfies all these requirements.

2.2 Problem Definitions

Definition 2.1. 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 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 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 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 2 again and the fairness constraint presented in Table 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 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.

3.1 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 3.1. 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.

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: $B_a = t(b - b_*(t)) - \sum_{i \in I_a \ \&\ L_C(i) \in G_B} L_V(i)$
4: $I_r = \{(i_a + 1), \ldots, (a_*(t) + b_*(t))\}$
5: $B_r = \sum_{i \in I_r \ \&\ L_C(i) \in G_A} L_V(i) - (t-1)(a_*(t) - a)$
6: $S = \max\{B_a, B_r\}$
7: Return S

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

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 3.3. 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 3.5 we prove that the optimal threshold t must be in the interval $[L_V(i_b), L_V(i_a)]$. Thus, from now on we just consider this interval.

For any threshold t in the open interval $(L_V(i_b), L_V(i_a))$ the optimal set of ballot additions and removals is determined in Case 3 of subroutine FindBallotSubB (described below). For a specific t, ballots are added to candidates in group G_B and removed from candidates in group G_A. Later we show how to replace the vote additions and removals by single ballot substitutions. The number of these single ballot substitutions is the maximum between the number of vote additions and vote removals.

The number of votes subtracted from candidates in G_A declines as t grows in this interval and the number of votes added to candidates in G_B grows as t grows in this interval. The optimal t can thus be found using binary search. We can make the binary search even more efficient, and instead of doing it on the interval $(L_V(i_b), L_V(i_a))$ which may be $\Omega(m)$ we can do it on the interval (i_a, i_b). After completing this binary search we identify an index $i \in (i_a, i_b)$, such that the optimal threshold is in the interval $[L_V(i), L_V(i+1))$. In our Technical Report we show how the optimal threshold in this interval can be computed in constant time.

3.2 Subroutine FindBallotSubB

Given a threshold t, FindBallotSubB finds the minimum number of single ballot substitutions that result in a fair outcome with this threshold. For simplicity we first assume that the fair outcome does not have a tie. Later, we show how to remove this assumption.

Let $a_*(t)$ and $b_*(t)$ be the number of candidates from groups G_A and G_B respectively who received at least t votes. Note that $L_V(a_*(t) + b_*(t)) = t$ and $L_V(a_*(t) + b_*(t) + 1) < t$.

This subroutine is designed by distinguishing the following cases.
Case 1: $t \leq L_V(i_b)$ (and $L_V(i_b) > 0$). In this case the numbers of candidates from groups G_A and G_B who got at least t votes are at least a and b, respectively; namely, $a_*(t) \geq a$ and $b_*(t) \geq b$. We decrease the number of votes of the $a_*(t) - a$ candidates from G_A in $L_C(i_a + 1), \ldots, L_C(a_*(t) + b_*(t))$ and the number of votes of the $b_*(t) - b$ candidates from G_B in $L_C(i_b + 1), \ldots, L_C(a_*(t) + b_*(t))$ to $t - 1$. To reconcile for the decrease of these votes, we add votes of the candidate in $L_C(1)$.
Case 2: $t > L_V(i_a)$. In this case the numbers of candidates from groups G_A and G_B who got at least t votes are less than a and b, respectively; namely, $a_*(t) < a$ and $b_*(t) < b$. 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 $O(n)$ time. We can also precompute required ballot additions and removals for $t\ in(i_a, i_b)$ which also requires $O(n)$. As a result Subroutine FindBallotSubB takes constant time. This subroutine is called $O(\log n)$ times in Alg1AttBOpt. Overall running time is $O(n + \log n) = O(n)$. The time complexity is dominated by the $O(n \log n)$ time it takes to sort the lists L_C and L_V.

Lemma 3.2. 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. □

Lemma 3.3. *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. □

Lemma 3.4. *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. □

Lemma 3.5. *The optimal threshold is in interval $[L_V(i_b), L_V(i_a)]$.*

Proof. Consider a threshold $t > L_V(i_a)$, to satisfy fairness, the number of votes of the $a - a_*(t)$ candidates from G_A in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_a)$ and the number of votes of the $b - b_*(t)$ candidates from G_B in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_b)$ need to be increased to at least t. On the other hand vote removals are not needed. It follows that the number of ballot substitutions equals the total number of ballot additions. Clearly, the number of vote additions required to guarantee fairness in case the threshold is $L_V(i_a)$ is lower, and thus t cannot be optimal. Similarly, for threshold $t < L_V(i_b)$, the number of vote removals required to guarantee fairness is more than this number when the threshold is $L_V(i_b)$. Hence, the optimal threshold t must be in the interval $[L_V(i_b), L_V(i_a)]$. □

Lemma 3.6. *The minimum number of ballot additions to G_B increases and the minimum number of ballot removals from G_A decreases monotonically with t in the interval $[L_V(i_b), L_V(i_a)]$.*

Proof. We get minimum number of ballot additions when we increase votes of the $b - b_*(t)$ candidates from G_B in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_b)$ to t. When t increases, both $b - b_*(t)$ and distance from t to votes of candidates from G_B in $L_C(a_*(t) + b_*(t) + 1), \ldots, L_C(i_b)$ increases. Hence, the minimum number of ballot additions to G_B increases monotonically with t in the interval $[L_V(i_a), L_V(i_b)]$. Similarly, we can prove that the minimum number of ballot removals from G_A decreases monotonically with t in the interval $[L_V(i_a), L_V(i_b)]$.

□

Theorem 3.7. 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 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 $O(n \log n)$ 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 $O(n)$ time, and each iteration takes $O(\log \ell)$ time. Thus the overall running time of Alg1AttMOpt (excluding the sorting) is $O(n + \log n \log \ell) = O(n)$.

Theorem 3.8. Alg1AttMOpt *always produces a fair outcome.*

Proof. The proof is similar to the proof of Theorem 3.7. □

4 MULTIPLE PROTECTED ATTRIBUTES

In this section we assume that there are ℓ attributes, denoted $A_1, \ldots, A_\ell$. For $i \in [1..\ell]$, attribute A_i has ℓ_i possible values, denoted $A[i, j]$, for $j \in [1..\ell_i]$. Each candidate is associated with a specific value from each attribute. In addition, we are given target quantities $a[i, j]$, for $i \in [1..\ell]$, and $j \in [1..\ell_i]$, with property that all marginals some to k. Namely, for every $i \in [1..\ell]$, $\sum_{j=1}^{\ell_i} a[i, j] = k$. A fair election outcome should satisfy the fairness condition that for $i \in [1..\ell]$, and $j \in [1..\ell_i]$, exactly $a[i, j]$ candidates whose A_i attribute value is $A[i, j]$ are elected.

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.

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:

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

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 $O(n^c)$, we can solve the c attribute configurations by $O(n^c)$ calls to the single attribute case and then choosing the call that produces the smallest margin. Alg1AttMOpt has a running time of $O(n)$. Overall running time is $O(n^{c+1})$. Clearly, when c is a constant, AlgCartOpt takes polynomial time to run.

Theorem 4.1. 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 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. □

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 4.2. *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. □

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 4.3. 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. □

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.

4.4.1 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

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 ∈ A_1}
Y = {y_j : y_j ∈ A_2}
E = {e_c = (x_i, y_j) : c ∈ L_C 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 = Σ_{e∈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)
```

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 4.4. *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 $L_V(n)$.

4.4.2 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 4.5. *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 $O(n\ell \log m)$ time [1]. The subroutine FindBallotA+R is called $O(n)$ times from Alg2AttApx. Thus, the running time of Alg2AttApx is $O(n^2 \ell \log m)$.

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

```
1: OPT_{A+R} = k · m
2: U = {L_V(i), L_V(i)+1, L_V(i+1)−1 : i ∈ [1...n−1]} ∪ {L_V(n)}
3: for t ∈ U do
4:     (AplusR, C*_t) = FindBallotA+R( L_C, L_V, A, t)
5:     if OPT_{A+R} > AplusR then
6:         OPT_{A+R} = AplusR
7:         C* = C*_t
8:     end if
9: end for
10: 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
```

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.

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.

5.1.1 Datasets Description. Algorithms are evaluated using multiple real world and a synthetic datasets. The real world datasets are described in Table 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 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 [36]. 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.

5.1.2 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 [22] + 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 [42] + 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.

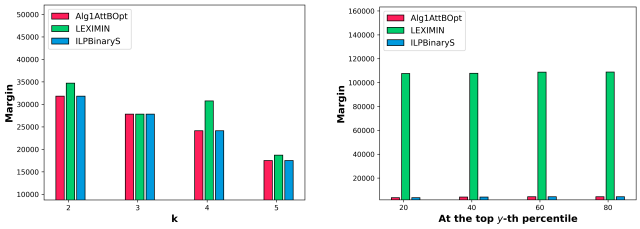
5.2 Quality Experiments Results

5.2.1 Results for MFBinaryS. Figure 1a shows the results from the election for 2021 Bronx Justice of the Supreme Court. There are 6 candidates(5 Democrats and 1 Republican), in which the candidate from Republican receives the least votes. We set the Republican must be included in the top-k (otherwise, the margin would be zero). We can observe that the margin decreases with increasing k.

In Figure 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.

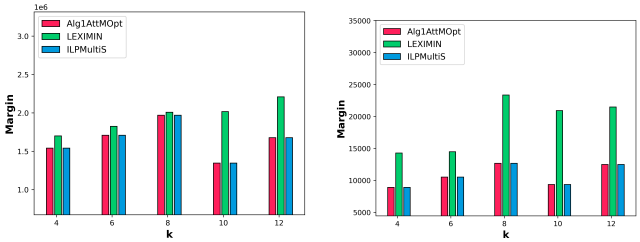
5.2.2 Results for MFMultiS. We present the results of MFMultiS in Figure 2. We consider political parties (36 different parties) of the candidates in 2019 NSW Senate Election, and the movie genres of the Movielens datasets (18 different genres) as the protected attribute. These results demonstrate similar pattern as that of MFBinarySresults.

5.2.3 Results for MFMulti2. Figure 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

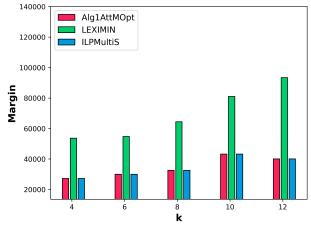


(a) 2021 Bronx Justice Election **(b) 2019 NSW Senate Election**

Figure 1: Results for MFBinaryS

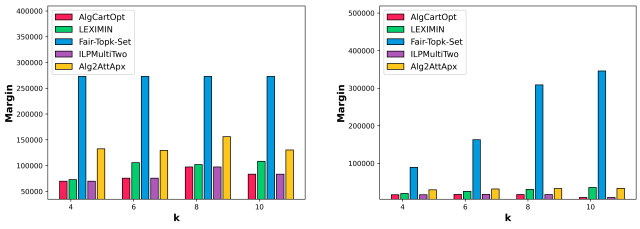


(a) 2019 NSW Senate Election **(b) MovieLens5Star**

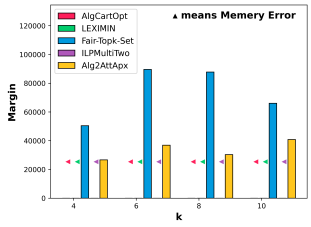


(c) MovieLens3StarMore

Figure 2: Results for MFMultiS



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

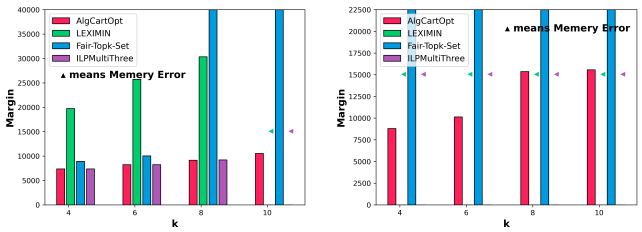


(c) MovieLens3StarMore

Figure 3: Results for MFMulti2

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.

5.2.4 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 4, which is consistent with our previous observations.



(a) Movielens5Star **(b) Movielens3StarMore**

Figure 4: Results for MFMulti3+

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.

5.3.1 Results for MFBinaryS, MFMultiS. Figure 5 shows that our proposed algorithms Alg1AttBOpt and Alg1AttMOpt are scalable up to millions of candidates and voters. In Figure 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 5b, 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.

Figures 7a and 7b show running times varying standard deviation *sd* and *mean*. We observe non-significant change in running time due to varying *mean* and *sd*.

5.3.2 Results for MFMulti2. The running time of Alg2AttApx w.r.t n, m, k are shown in Figure 6. The running time of Alg2AttApx is sub-quadratic w.r.t number of candidates n as shown in Figure 6a, while it increases linearly w.r.t number of voters m (shown in Figure 6b) and does not change significantly with the size of the result set k (Figure 6c). As before, Fair-Topk-Set is the fastest, while LEXIMIN does not scale.

5.3.3 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.

In Figures 8a and 8b, we see that the running time of AlgCartOpt increases exponentially with n and c, as expected, whereas Fair-Topk-Set, LEXIMIN run faster.

5.4 Summary of results

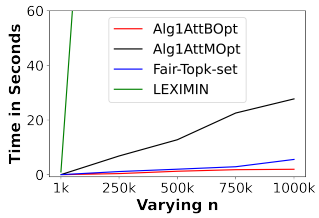
Our first and foremost observation is, consistent with our theoretical analysis Alg1AttBOpt, Alg1AttMOpt, AlgCartOpt produce exact solutions of the underlying problems, i.e., they satisfy the fairness constraints, while minimizing the margin values, whereas, Alg2AttApx demonstrates better approximation factors compared to the theoretical bound 2. Our second observation is that the implemented state-of-the-art solutions LEXIMIN [22] and Fair-Topk-Set [42], 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 [42] 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 solutions as well as LEXIMIN [22] give memory error when run on very large dataset. Overall, our designed solutions turn out to be the unanimous choice.

6 RELATED WORK

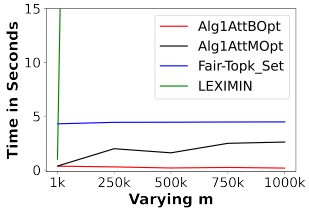
We primarily discuss three types of existing work that are related to our proposed problem.

Preference Elicitation and Aggregation. Preference of the individual users could be elicited as pairwise comparison [18], in form of a binary vector [39] known as Approval Voting [10], in an ordinal scale [2, 31], or considering Arrowian social choice, where users provide partial or complete preference order over the items [8, 12, 30, 41]. Similarly, The properties of social welfare functions for aggregating preferences have been studied by mathematicians since the 18th century [11, 15, 16]. Different preference aggregation methods are proposed, including majority voting, plurality voting [33, 35, 37], their weighted versions, as well as aggregation methods that consider positional preference [8, 12, 41], such as Kemeny rule [19, 29], Condorcet rule [17], or Borda Count [20]. 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.

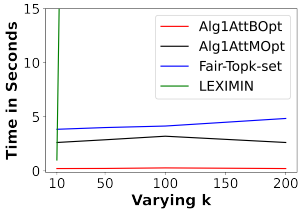
Fairness in Preference Aggregation and Top-k. In [23, 38], authors study the fairness of preference aggregation in the context of Arrow's Impossibility Theorem. In a very recent work of ours [44], we study how to ensure proportionate fairness[5, 6] in aggregating preferences that are provided as ranked orders. Earlier, existing works study proportionate representation considering group fairness in the top-k ranked order [26, 32]. Authors in [24] study how to strike a balance between individual and group fairness in selecting top-k order. In [3], 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 [22, 42] to maximize



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

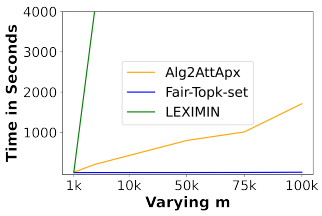


(b) Vary m, $k = 50, n = 100k$

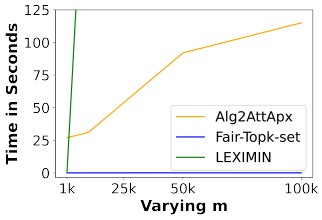


(c) Vary k, $n = 100k, m = 10000k$

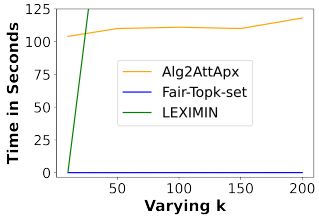
Figure 5: Running Time for Alg1AttBOpt **&** Alg1AttMOpt



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

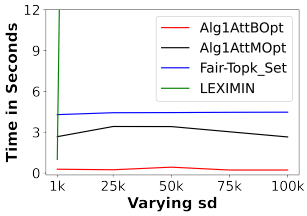


(b) Vary m, $n = 10k, k = 50$

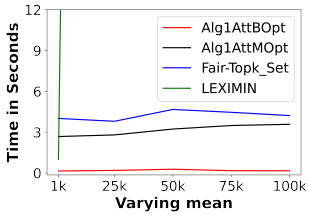


(c) Vary k, $n = 10k, m = 100k$

Figure 6: Running Time for Alg2AttApx

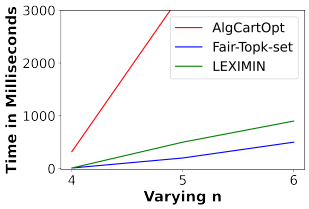


(a) Vary sd, $m = 1000k, n = 100k, mean = 50k$

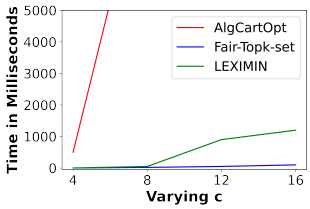


(b) Vary $mean$, $m = 1000k, n = 100k, sd = 30k$

Figure 7: Varying distribution Alg1AttBOpt **&** Alg1AttMOpt



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



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

Figure 8: Running Time for MFMulti3+

fairness or diversity. In a recent paper [14], 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 [28]. 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) [21, 43]. In [45], 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 [9, 13, 40] margin is calculated in STV setting. In [4], 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.

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.

ACKNOWLEDGMENTS

The work of Md. Mouinul Islam, Dong Wei, and Senjuti Basu Roy are supported by the NSF CAREER Award #1942913, IIS #2007935, IIS #1814595, PPoSS:Planning #2118458, and by ONR Grants: #N000141812838, #N000142112966.

REFERENCES

[1] Ravindra K. Ahuja, Andrew V. Goldberg, James B. Orlin, and Robert Endre Tarjan. 1992. Finding minimum-cost flows by double scaling. *Math. Program.* 53 (1992), 243–266.

[2] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. 2009. Group recommendation: Semantics and efficiency. *Proceedings of the VLDB Endowment* 2, 1 (2009), 754–765.

[3] Abolfazl Asudeh, HV Jagadish, Julia Stoyanovich, and Gautam Das. 2019. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data*. 1259–1276.

[4] John J Bartholdi and James B Orlin. 1991. Single transferable vote resists strategic voting. *Social Choice and Welfare* 8, 4 (1991), 341–354.

[5] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* 15, 6 (1996), 600–625.

[6] Sanjoy K Baruah, Johannes E Gehrk, C Greg Plaxton, Ion Stoica, Hussein Abdel-Wahab, and Kevin Jeffay. 1997. Fair on-line scheduling of a dynamic set of tasks on a single resource. *Inform. Process. Lett.* 64, 1 (1997), 43–51.

[7] Senjuti Basu Roy and Kaushik Chakrabarti. 2011. Location-aware type ahead search on spatial databases: semantics and efficiency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 361–372.

[8] Julian H Blau. 1957. The existence of social welfare functions. *Econometrica: Journal of the Econometric Society* (1957), 302–313.

[9] Michelle Blom, Peter J Stuckey, and Vanessa J Teague. 2017. Towards computing victory margins in STV elections. *arXiv preprint arXiv:1703.03511* (2017).

[10] Steven J Brams and Peter C Fishburn. 1978. Approval voting. *American Political Science Review* 72, 3 (1978), 831–847.

[11] Donald E Campbell and Jerry S Kelly. 2000. Information and preference aggregation. *Social Choice and Welfare* 17, 1 (2000), 3–24.

[12] Donald E Campbell and Jerry S Kelly. 2002. Impossibility theorems in the Arrovian framework. *Handbook of social choice and welfare* 1 (2002), 35–94.

[13] David Cary. 2011. Estimating the Margin of Victory for Instant-Runoff Voting. In *2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 11)*.

[14] L Elisa Celis, Lingxiao Huang, and Nisheeth K Vishnoi. 2017. Multiwinner voting with fairness constraints. *arXiv preprint arXiv:1710.10057* (2017).

[15] Christopher P Chambers and Takashi Hayashi. 2006. Preference aggregation under uncertainty: Savage vs. Pareto. *Games and Economic Behavior* 54, 2 (2006), 430–440.

[16] Yong-Gon Cho and Keun-Tae Cho. 2008. A loss function approach to group preference aggregation in the AHP. *Computers & Operations Research* 35, 3 (2008), 884–892.

[17] Marquis de Condorcet. 1785. Essay on the Application of Analysis to the Probability of Majority Decisions. *Paris: Imprimerie Royale* (1785).

[18] Luca de Alfaro and B. Thomas Adler. 2013. Content-Driven Reputation for Collaborative Systems. In *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8358)*, Martín Abadi and Alberto Lluch-Lafuente (Eds.). Springer, 3–13. https://doi.org/10.1007/978-3-319-05119-2_1

[19] Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. 2001. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*. 613–622.

[20] Peter Emerson. 2013. The original Borda count and partial voting. *Social Choice and Welfare* 40, 2 (2013), 353–358.

[21] David M Farrell, Malcolm Mackerras, and Ian McAllister. 1996. Designing electoral institutions: STV systems and their consequences. *Political studies* 44, 1 (1996), 24–43.

[22] Bailey Flanigan, Paul Gölz, Anupam Gupta, Brett Hennig, and Ariel D Procaccia. 2021. Fair algorithms for selecting citizens' assemblies. *Nature* 596, 7873 (2021), 548–552.

[23] Marc Fleurbaey, Kotaro Suzumura, and Koichi Tadenuma. 2005. The informational basis of the theory of fair allocation. *Social Choice and Welfare* 24, 2 (2005), 311–341.

[24] David García-Soriano and Francesco Bonchi. 2021. Maxmin-fair ranking: individual fairness under group-fairness constraints. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 436–446.

[25] Michael R Garey and David S Johnson. 1979. *Computers and intractability*. Vol. 174. freeman San Francisco.

[26] Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. 2019. Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*. 2221–2231.

[27] Corinna Hertweck, Christoph Heitz, and Michele Loi. 2021. On the moral justification of statistical parity. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 747–757.

[28] Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. 2019. An Empirical Study of Rich Subgroup Fairness for Machine Learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. ACM. https://doi.org/10.1145/3287560.3287592

[29] John G Kemeny. 1959. Mathematics without numbers. *Daedalus* 88, 4 (1959), 577–591.

[30] Craig W Kirkwood and Rakesh K Sarin. 1985. Ranking with partial information: A method and an application. *Operations Research* 33, 1 (1985), 38–48.

[31] Yehuda Koren and Joe Sill. 2011. Ordrec: an ordinal model for predicting personalized item rating distributions. In *Proceedings of the fifth ACM conference on Recommender systems*. 117–124.

[32] Caitlin Kuhlman and Elke Rundensteiner. 2020. Rank aggregation algorithms for fair consensus. *Proceedings of the VLDB Endowment* 13, 12 (2020).

[33] Jean-François Laslier. 2012. And the loser is… plurality voting. In *Electoral systems*. Springer, 327–351.

[34] Reshef Meir. 2015. Plurality voting under uncertainty. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[35] Reshef Meir, Maria Polukarov, Jeffrey Rosenschein, and Nicholas Jennings. 2010. Convergence to equilibria in plurality voting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 24. 823–828.

[36] Samuel Merrill III. 1984. A comparison of efficiency of multicandidate electoral systems. *American Journal of Political Science* (1984), 23–48.

[37] Lionel S Penrose. 1946. The elementary statistics of majority voting. *Journal of the Royal Statistical Society* 109, 1 (1946), 53–57.

[38] Francesca Rossi, Kristen Brent Venable, and Toby Walsh. 2005. Aggregating preferences cannot be fair. *Intelligenza Artificiale* 2, 1 (2005), 30–38.

[39] Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, Gautam Das, and Cong Yu. 2014. Exploiting group recommendation functions for flexible preferences. In *2014 IEEE 30th international conference on data engineering*. IEEE, 412–423.

[40] Anand D Sarwate, Stephen Checkoway, and Hovav Shacham. 2013. Risk-limiting audits and the margin of victory in nonplurality elections. *Statistics, Politics, and Policy* 4, 1 (2013), 29–64.

[41] Amartya Sen. 1986. Social choice theory. *Handbook of mathematical economics* 3 (1986), 1073–1181.

[42] Julia Stoyanovich, Ke Yang, and HV Jagadish. 2018. Online set selection with fairness and diversity constraints. In *Proceedings of the EDBT Conference*.

[43] Nicolaus Tideman. 1995. The single transferable vote. *Journal of Economic Perspectives* 9, 1 (1995), 27–38.

[44] Dong Wei, Md Mouinul Islam, Schieber Baruch, and Senjuti Basu Roy. 2022. Rank Aggregation with Proportionate Fairness. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*.

[45] Lirong Xia. 2012. Computing the margin of victory for various voting rules. In *Proceedings of the 13th ACM conference on electronic commerce*. 982–999.

[46] Xiaohang Zhang, Guoliang Li, and Jianhua Feng. 2016. Crowdsourced top-k algorithms: An experimental evaluation. *Proceedings of the VLDB Endowment* 9, 8 (2016), 612–623.