GENERALIZING FAIR TOP-k SELECTION: AN INTEGRATIVE APPROACH

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the problem of finding a fair (linear) scoring function with multiple protected groups (i.e., minority or disadvantaged groups) while also minimizing the disparity from a reference scoring function, generalizing the prior setup that was restricted to the single-group setting without disparity minimization. Driven by the need for experimental exploration, we find that previous studies overlook a critical issue that may affect the fairness of the outcome. Once it is properly considered, the problem may become computationally intractable even for a two-dimensional dataset and small k. However, our analysis also reveals a gap that enables us to recover the efficiency for small k when the number of protected groups is sufficiently small. Furthermore, we introduce a new disparity measure—utility loss—that may yield a more stable scoring function under small weight perturbations. Through careful engineering trade-offs that balance implementation complexity, robustness, and performance, our augmented two-pronged solution demonstrates strong empirical performance on real-world datasets, with experimental observations also informing algorithm design and implementation decisions.

1 INTRODUCTION

Fair top-k selection, especially the approach that selects the k-best items under proportional fairness constraints, has drawn significant attention (Zehlike et al., 2017; Kleinberg & Raghavan, 2018; Celis et al., 2018; Asudeh et al., 2019; Yang et al., 2019; Campbell et al., 2024; Cai, 2026a). Top-k selection often works as selecting the k highest-scoring items from a dataset with n items. In the fair top-k selection problem, the aim is to select a top-k subset where the proportion of each protected group (i.e., minority or disadvantaged group) mirrors its proportion in the entire dataset, enforced by proportional fairness constraints. In many previous works (e.g., Zehlike et al. (2017); Celis et al. (2018); Yang et al. (2019); Campbell et al. (2024)), the fairness constraints are applied after scores to each item are identified. However, this method may result in a different selection criterion to different groups of people, introducing potential legal risks (e.g., Ricci vs. DeStefano (SCOTUS, 2009)). So, designing a fair scoring function from the outset is preferable. Consider the following variation of the example from (Asudeh et al., 2019; Cai, 2026a):

Example 1. *A college admissions officer is designing a selection process based on applicants' normalized GPA (g) and SAT score (s), using the initial scoring function $f(c) = 0.5 \times g + 0.5 \times s$ to select the top-500 applicants. However, this may yield only 150 female and 30 Black applicants, violating the fairness constraints requiring at least 40% female and 10% Black representation. This violation may be due to gender and race disparities in the SAT score (The College Board, 2025). To preserve explainability, the officer may seek a fair scoring function with weights constrained to $[0.45, 0.55]$. Among multiple feasible options, they may choose the one minimizing the sum of absolute differences from the original weights. An algorithm may find such a fair scoring function $f(c) = 0.54 \times g + 0.46 \times s$. However, it is possible that no fair function exists within this range, indicating the equal-weight design may be unsuitable. The officer may then revise the design, with algorithmic support, to obtain a fair scoring function such as $f(c) = 0.6 \times g + 0.4 \times s$.*

Previous work (Asudeh et al., 2019; Liu et al., 2024; Cai, 2026a) has explored finding such a fair scoring function in a similar setup. However, most of these methods (Asudeh et al., 2019; Liu et al., 2024) do not scale efficiently for high-dimensional datasets and do not address a critical issue: potential ties among candidates. Since tie-breaking can affect the number of selected candidates from

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

protected groups, this issue merits greater attention (Cachel & Rundensteiner, 2025; Cai, 2026a). A more recent approach (Cai, 2026a) improves scalability and explicitly addresses the tie-breaking issue; however, it considers only a single protected group and returns an arbitrary fair scoring function within the allowable scoring function subspace, which may not be desirable in some applications (as illustrated in Example 1). (See Appendix B for further discussions on related work.)

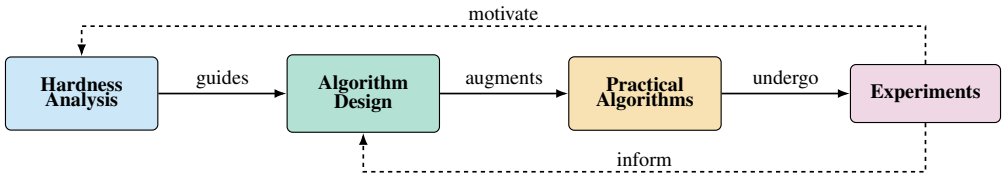


Figure 1: The structure and interplay of key components in this work. Solid arrows denote the primary workflow, while dashed arrows indicate feedback and motivation. (See Appendix A.)

Our contributions. For the fair top-k selection problem with multiple protected groups and disparity minimization, instead of focusing on **locally defined contribution**, we adopt an **integrative framework** encompassing both high-level theoretical reasoning and low-level engineering optimizations (see Figure 1). Previous work (Asudeh et al., 2019; Liu et al., 2024) implies that a larger number of protected groups may not impose significant runtime overhead. However, we find that this implication overlooks the tie-breaking issue. Once this critical issue that may affect the fairness of the outcome is properly considered, we show that the problem becomes NP-hard even for a two-dimensional dataset. Furthermore, we establish a near $\Omega(n^k)$ lower bound for constant values of k under a well-known conjecture in fine-grained complexity (Williams, 2018), invalidating the "Small k Opportunity" identified in (Cai, 2026a) in the worst case. In particular, the hardness analysis is driven by the need for generating inputs for our experimental explorations. Despite these negative findings, the hardness analysis also reveals a gap in the hardness barrier, that when the number of protected groups is sufficiently small, the "Small k Opportunity" can be recovered. With this gap exploited, we further augment the algorithms of the two-pronged solution proposed in (Cai, 2026a), making them efficient in the generalized setup. Specifically, we introduce a new disparity measure—utility loss—that enables a more stable scoring function under small weight perturbations comparing to the distance-based measures in (Asudeh et al., 2019; Liu et al., 2024). To implement our augmented two-pronged solution in practice, we address several practical engineering considerations, balancing implementation complexity, robustness, and performance through careful engineering trade-offs. Experimental results on real-world datasets demonstrate the efficiency of our augmented two-pronged solution, while also informing algorithm design and implementation decisions, including the choice between the two algorithms in our augmented two-pronged solution.

2 Preliminaries and Problem Definition

In this problem, we are given a set of candidates (or items) $\mathcal{C}$, where each candidate $c \in \mathcal{C}$ is described by d scoring attributes and a set of sensitive attribute values. We represent the scoring attribute values by a point $p(c) \in \mathbb{R}^d$ and the sensitive attribute values by an appropriate subset of labels $A(c) \subseteq \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. Among these labels, $\mathcal{G}_j$ for $1 \leq j \leq n_p$ denote protected groups, where n_p is their total number. We also use $\mathcal{G}_j$ to denote the subset of candidates with $\mathcal{G}_j \in A(c)$ when the context is clear. Additionally, "artificial" group labels can be created to represent intersections of multiple protected groups, enabling intersectional fairness constraints (see Appendix D).

We consider the same class of linear scoring functions as in (Cai, 2026a), defined by a non-negative weight vector $w = (w_1, w_2, \ldots, w_d)$ satisfying $||w||_1 = 1$. A top-k subset τ_k^w is defined as follows:

Definition 1. *For a given weight vector w and an integer $k \leq n$, a subset $\tau_k^w \subseteq C$ is a top-k subset of C if $|\tau_k^w| = k$ and for any pair of candidates that $c \in \tau_k^w$ and $c' \in C \setminus \tau_k^w$, $w \cdot p(c) \geq w \cdot p(c')$.*

Since ties may make the top-k subset for a given w non-unique, we consider all top-k subsets and regard w as fair if at least one of its top-k subsets satisfies all fairness constraints. For each protected group $\mathcal{G}_j$, the corresponding lower and upper bounds are given as $L_k^{\mathcal{G}_j}$ and $U_k^{\mathcal{G}_j}$. Vector w is fair if

$$L_k^{\mathcal{G}_j} \leq |\tau_k^w \cap \mathcal{G}_j| \leq U_k^{\mathcal{G}_j}, \tag{1}$$

for all $1 \leq j \leq n_p$. Notably, introducing "artificial" group labels enables independent fairness constraints on intersections of multiple protected groups, addressing the limitation that individual constraints may not ensure adequate representation for such candidates (Yang et al., 2021).

As in (Cai, 2026a), we aim to find the fair weight vector in a subset V of the weight vector space described by l linear inequalities. Furthermore, we aim to find a fair weight vector w^f within V that minimizes the disparity from the reference weight vector w^o under one of two objectives $g^o(w^f)$:

- ***w* difference.** The disparity of w^f is measured by its L_1 distance from w^o, formally as

$$g^o(w^f) = ||w^f - w^o||_1 = \sum_{i=1}^{d} |w_i^f - w_i^o|. \tag{2}$$

  This objective can be regarded as the sum of the absolute differences between vector components.
- **Utility loss.** By the principle in (Zehlike et al., 2022), the utility of τ_k is computed by summing scores of all candidates in τ_k under w^o, that is, $U^o(\tau_k) = \sum_{c \in \tau_k} w^o \cdot p(c)$. The disparity of w^f is measured by the relative utility loss of τ_k, formally as

$$g^o(w^f) = 1 - \frac{U^o(\tau_k^f)}{U^o(\tau_k^o)} = 1 - \frac{\sum_{c \in \tau_k^f} w^o \cdot p(c)}{\sum_{c \in \tau_k^o} w^o \cdot p(c)}, \tag{3}$$

  where τ_k^f and τ_k^o are top-k subsets of w^f and w^o, respectively.

Notably, since ties in scores may yield multiple top-k subsets for a w^f, we choose the subset satisfying fairness constraints with the minimum utility loss to determine $g^o(w^f)$. Notably, minimizing the utility loss is equivalent to maximizing the utility under w^o (Appendix D). One benefit of the utility loss objective is that it enables improvement in the stability of a fair scoring function (Section 4.2).

To conclude, the (Generalized) Fair Top-k Selection problem aims to find a fair vector $w^f \in V$ that satisfies the fairness constraints while minimizing $g^o(w^f)$ (see Appendix D for a formal definition).

3 HARDNESS

Cai (2026a) considered the hardness of a special version of the problem, where $n_p = 1$ and the disparity minimization was not required. While those hardness results are still valid to the generalized version, their analysis also revealed the following gaps in the hardness barrier:

- **Low Dimensions Opportunity.** The problem is NP-hard in arbitrary dimensions but solvable in polynomial time for fixed (constant) dimensions, with upper and lower bounds becoming smaller as the dimensionality d decreases. Notably, the hardness analysis does not cover the $d = 2$ case.
- **Small k Opportunity.** The hardness analysis does not apply to a sufficiently small k. Subsequent algorithmic results confirm that the lower bounds are indeed breakable in such cases.

While earlier work (Asudeh et al., 2019; Liu et al., 2024) implies that these "opportunities" may persist, we show that with multiple protected groups, these "opportunities" in the single-protected-group setting may vanish in the worst case. We establish the results via *Fair Top-k Verification*, which determines whether a given weight vector w induces a top-k subset satisfying all fairness constraints, and becomes computationally hard due to ties among candidates. (See Appendix E.1.)

Notably, the Fair Top-k Verification problem also plays a crucial role in our experimental evaluation, specifically for input generation. Since we sample weight vectors uniformly at random until a target number of *unfair* weight vectors are obtained, an efficient algorithm to determine whether a sampled weight vector is fair or not is essential for our experimental explorations (see Section 6.1).

3.1 HARDNESS RESULT IN LOW DIMENSIONS

We first consider the hardness of the problem in low dimensions, specifically for $d = 2$. By a reduction from the Set Cover problem to the Fair Top-k Verification problem, we show the following:

Theorem 1. *Fair Top-k Verification is NP-hard for an arbitrary n_p.*

Remark 1. *Cai (2026a) established that the Fair Top-k Selection problem is solvable in polynomial-time for any constant d when $n_p = 1$. In contrast, our reduction suggests that a polynomial-time algorithm for general problem instances ($n_p > 1$) is unlikely to exist, even in constant dimensions.*

3.2 Hardness result for small k

Our NP-hardness result considers the case where k can be large. However, k can be small in many applications. If k is a constant, a naive algorithm exists for solving the Fair Top-k Verification problem in polynomial time by testing all possible k-candidate subsets, with a run time of $O(n^k \cdot n_p)$. Moreover, our NP-hardness result only applies to the case where n_p is large. In practice, n_p is small in most datasets. Here, we further consider the case where $k \geq 2$ is a constant and n_p is modest.

Our hardness result relies on a fundamental problem in fine-grained complexity (Williams, 2018): t-Orthogonal Vectors (t-OV, often referred to as k-OV in the literature). The t-OV hypothesis is widely studied due to their connections to the strong exponential time hypothesis (SETH) (Impagliazzo & Paturi, 2001), established in (Williams, 2005). By encoding all protected groups to which a candidate belongs into a binary vector, we are able to show the following:

Theorem 2. *For every constant $\delta > 0$, there is a constant $\alpha > 0$ such that Fair Top-k Verification cannot be solved in $O(n^{k-\delta})$ time for $n_p = \alpha \log n$, assuming the t-OV hypothesis.*

Remark 2. *For a constant $k \geq 2$ and $n_p = \alpha \log n$, the naive algorithm takes $O(n^k \cdot \log n)$ time. Our reduction shows that in such cases, significant improvements over the naive approach are unlikely. Consequently, the Fair Top-k Selection problem may not admit practically efficient algorithms (i.e., an algorithm with a run time of $O(n \cdot \text{polylog}(n))$) for such cases. However, we also note that the reduction only works for n_p being large enough such that the input sets can contain a sufficient number of distinct binary vectors. This observation, in fact, implies a more efficient algorithm when n_p is sufficiently small.*

4 Small k opportunity revisited

Despite negative results in Section 3, Remark 2 implies that when n_p is sufficiently small, the lower bounds may be breakable. In this section, we confirm this implication: for $n_p = O(1)$ and $k = O(\text{polylog}(n))$, the Fair Top-k Verification problem can be solved in linear time when $d = O(1)$. By leveraging the key insights of the resulting algorithm, we largely recover the results from the "Small k Opportunity" for the Fair Top-k Selection problem, with only a polylogarithmic overhead.

4.1 The case of small n_p (and k)

Recall in the proof of Theorem 2, the protected groups to which a candidate belongs, *protected groups membership profile*, is encoded into a binary vector. As noted in Remark 2, the lower bound does not hold when the number of distinct binary vectors is small. This suggests that the Fair Top-k Verification problem may admit a faster algorithm when there are few distinct protected groups membership profiles. Note that when $n_p = O(1)$, the number of distinct profiles is indeed small ($O(1)$), and it may result in many candidates having the same membership profile. Our algorithm exploits the key observation that tied candidates with the same membership profile are "exchangeable" for a top-k subset: swapping a selected candidate with an unselected (and tied) candidate having the same membership profile preserves all protected group member counts. Thus, potentially many top-k subsets are equivalent for the fairness test, so they can be bypassed for runtime improvements.

Now, let us proceed to the algorithm. Given w, our algorithm first identifies the set of candidates tied at the top-k selection boundary, along with the exact number of positions, t, to be filled from this set. We then determine the number of boundary candidates for each distinct protected groups member profile. Finally, a backtracking subroutine enumerates all valid combinations of selections across these distinct profiles that sum to exactly t (see Appendix F.1). This gives us the following:

Theorem 3. *With $n_p = O(1)$ and a sufficiently small k (e.g., $k = \text{polylog}(n)$), Fair Top-k Verification can be solved in $O(n \cdot d)$ time.*

Remark 3. *While our worst-case runtime analysis requires k to be small since t can be as large as k, the algorithm may remain efficient even for large k, provided that t is sufficiently small. Moreover, there are some heuristics (Section 5.1) for improving its practical run time.*

4.2 AUGMENTING THE k-LEVEL-BASED ALGORITHM

Having developed an efficient algorithm for the Fair Top-k Verification problem when both k and n_p are small, we now turn to the Fair Top-k Selection problem. We show how to augment the theoretically efficient k-level-based algorithm (Cai, 2026a)—which is designed for small k—to handle multiple protected groups and disparity minimization. (See Appendix F.2 for the concept of k-level.)

Multiple protected groups. In (Cai, 2026a), the theoretically efficient k-level-based algorithm for $n_p = 1$ works by traversing the cells of the $(k-1)$-level. While a straightforward adaptation appears to works for $n_p > 1$, it breaks down when ties occur. To address this, our augmentation incorporates the backtracking subroutine (Section 4.1) that handles a top-k subset with ties correctly. To facilitate efficient calls to the backtracking subroutine, we additionally track, for each protected group, the number of hyperplanes that lie strictly above the current $(k-1)$-level cell (i.e., candidates with strictly higher scores). Moreover, each $(k-1)$-level cell also stores auxiliary information about its intersecting hyperplanes (i.e., tied candidates), allowing the number of tied candidates for each protected groups membership profile to be identified efficiently. (See Appendix F.3 for details.)

***w* difference.** For minimizing the w difference, we focus on the *fair cell*, which is a cell that corresponds to a fair top-k subset. Let $\mathcal{F}$ denotes a fair cell that is projected onto the first $d-1$ coordinates, and $\mathcal{H}$ be the set of boundary hyperplanes of $\mathcal{F}$. Using the known technique that reformulates L_1 norm minimization as a linear program, we have the following:

$$\begin{aligned} \min \ & \sum_{i=1}^{d} \phi_i \\ \text{s.t.} \ & \sum_{i=1}^{d-1} h_i w_i + h_d \geq 0, \qquad \forall h \in \mathcal{H} \\ & \phi_i \geq w_i - w_i^o, \qquad i = 1, \ldots, d \\ & \phi_i \geq w_i^o - w_i, \qquad i = 1, \ldots, d \end{aligned} \tag{4}$$

To simplify presentation, constraints on the weight vector (e.g., non-negativity) are omitted from the formulation (see Figure 2 for illustrative example and Appendix F.3 for its justification).

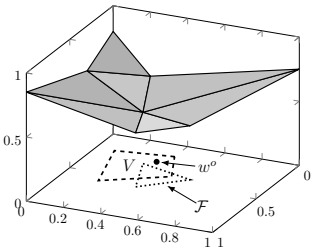


Figure 2: $(k-1)$-level (triangular mesh) and V (dashed region on the x-y plane) in 3-D. The dotted region (on the x-y plane) represents $\mathcal{F}$ and the point inside V represents w^o. When minimizing the w difference, the resulting fair weight vector will lie on the boundary of $\mathcal{F}$. In contrast, minimizing the utility loss enables a stable weight vector by selecting a weight vector inside $\mathcal{F}$ and away from its boundary.

Utility loss. For minimizing the utility loss, one only need to find the fair cell with the minimum utility loss. Rather than recomputing the utility at each fair cell, we update the utility dynamically during the traversal to improve its efficiency. Moreover, to handle a top-k subset with ties, a greedy approach is employed along with an adapted backtracking subroutine that updates the utility dynamically during the search, which only introduces a small runtime overhead (see Appendix F.3).

One may observe that a weight vector minimizing the utility loss may not be unique, as two weight vectors may induce the same top-k subset. In fact, the utility loss optimization objective enables us to consider another important aspect of the top-k selection: the stability of a scoring function (Asudeh et al., 2018). A fair weight vector minimizing the w difference is inherently unstable, as a small perturbation towards w^o would yield top-k subsets violating the fairness constraints. With the utility loss objective, one is able to obtain a stable fair weight vector by solving the following LP:

$$\begin{aligned} \max \ & \xi \\ \text{s.t.} \ & \sum_{i=1}^{d-1} h_i w_i + h_d \geq \xi, \quad \forall h \in \mathcal{H} \end{aligned} \tag{5}$$

Here, the normal vector of each $h \in \mathcal{H}$ is normalized by its L_∞ norm. This linear program enforces a margin of ξ between the weight vector and the cell boundary, ensuring that a weight vector under small perturbations bounded by ξ induces the same top-k subset (with the minimum utility loss). As a result, the resulting fair weight vector is more stable with respect to perturbations (see Figure 2).

To conclude, we show the run time of our augmented k-level-based algorithm:

Theorem 4. *With $n_p = O(1)$, $k = O(\text{polylog}(n))$ and $l = |V| = O(\text{polylog}(n))$, Fair Top-$k$ Selection can be solved in $\tilde{O}(n)$ time for $d = 2, 3$ ($\tilde{O}(\cdot)$ hides a polylog n factor). For constant dimensions $d \geq 4$, it can be solved in $\tilde{O}(n^{\lfloor d/2 \rfloor})$ (expected) time.*

5 Practical algorithms

It is widely known that a theoretically efficient algorithm may not perform well in practice. In this section, we first present an efficient implementation of the backtracking subroutine for tie-breaking (recall Remark 3). We then show how to augment the practical two-pronged solution (Cai, 2026a)—one algorithm for small k and another for large k—to accommodate multiple protected groups and disparity minimization. Key engineering decisions and trade-offs are discussed along the way.

5.1 Engineering the backtracking algorithm for tie-breaking

Recall in Section 4.1, the backtracking subroutine considers all possible protected groups membership profiles. However, some membership profiles may have no associated candidates, and thus can be ignored. In particular, adding "artificial" group labels corresponding to the intersections of multiple protected groups does not increase the number of "non-empty" membership profiles. Thus, the algorithm incurs only a small runtime overhead for handling intersectional fairness constraints. Moreover, additional pruning strategies are used to accelerate the backtracking search by tracking the number of remaining candidates of membership profiles. We also employ a greedy approach as a fast pruning step before invoking the backtracking search. (See Appendix G.1.)

Recall in Section 4.2, an adapted backtracking subroutine that updates the utility dynamically is used to minimize the utility loss. While the method is theoretically more efficient, in our implementation, we instead recompute the utility from scratch each time (with the greedy approach still being employed for the speedup), as this method is numerically more robust (see Section 6.2).

5.2 Practical k-level-based algorithms

For $n_p = 1$ without an optimization objective, two practical k-level-based algorithms for small k have been given in (Cai, 2026a): one for $d = 2$ and one for $d \geq 3$. We show how to augment them.

5.2.1 2-D k-level-based algorithm

For $n_p = 1$ without an optimization objective, the 2-D k-level-based algorithm (Cai, 2026a) is an adaptation of a sweep-line algorithm using kinetic tournament trees in (Chan, 1999) (briefly outlined in (Basch et al., 1996)). The adapted algorithm employs a suite of techniques to address engineering challenges arising in practice, as demonstrated in (Cai, 2026a).

Multiple protected groups. As in the theoretical algorithm, our augmentations focus on top-k subsets with ties, The candidates tied at the top-k boundary are identified as intersecting lines within the two kinetic tournament trees. The member counts for each protected group within one of the trees are also tracked to facilitate efficient calls to the backtracking subroutine. (See Appendix G.2.1.)

***w* difference.** With $d = 2$ and $||w||_1 = 1$, finding a fair weight vector minimizing the w difference is equivalent to finding a fair weight vector minimizing the absolute difference in the first coordinate. In the context of the sweep-line algorithm, it becomes finding the closest vertical line from the one induced by w^o that yields a top-k subset satisfying the fairness constraints. So, a bidirectional sweep-line algorithm can be employed by sweeping from w^o in both directions (see Appendix G.2.1).

This approach is efficient as it only (implicitly) constructs the part of $(k-1)$-level that is relevant to the search. However, in practice, a unidirectional sweeping approach can be more efficient due to

the cost of constructing kinetic tournament trees twice in the bidirectional approach. Nevertheless, The previous observation is used as an early termination criterion in the unidirectional approach.

Utility loss. At first glance, it appears the entire $(k-1)$-level within V must be swept for finding a fair weight vector minimizing the utility loss. However, the bidirectional sweeping approach can also be applied here, due to a monotonicity property of the utility of the top-k subset we establish (see Appendix G.2.1). Of course, a unidirectional sweeping approach is still used due to its better practical performance, with the observation works as an early termination criterion.

To obtain a stable fair weight vector, while it is possible to make it work by directly adapting the sweep-line algorithm, we instead employ a post-processing approach (see Appendix G.2.1) that is independent of the underlying Fair Top-k Selection algorithm. While it may introduce some runtime overhead, the approach is more flexible and can work with other Fair Top-k Selection algorithms.

5.2.2 Multi-dimensional k-level-based algorithm

For $n_p = 1$ without an optimization objective, the multi-dimensional k-level-based algorithm (Cai, 2026a) adapts the algorithm in (Andrzejak & Fukuda, 1999). Notably, via a lockless implementation proposed in (Cai, 2026a), the algorithm can efficiently utilize a multi-core shared-memory system.

Multiple protected groups. Handling multiple protected groups is simple since the algorithm actually enumerates all valid top-k subsets. One can easily test the fairness using a simple counting method. Moreover, we further improve the performance by accounting for V (see Appendix G.2.2).

***w* difference.** Since in this algorithm, the $(k-1)$-level cell is not explicitly determined, the linear program (4) cannot be directly applied. Fortunately, given the knowledge of the top-k subset, it is easy to adapt the linear program to minimize the w difference (see Appendix G.2.2). Notably, we choose the simplex algorithm over the Seidel's LP algorithm (Seidel, 1991) for solving the result LP, despite the theoretically worse performance in low dimensions. The choice was informed by an experimental observation on the number of fair cells (see Section 6.3).

The algorithm is kept lockless via a final reduction step to determine the global minimum among local optima from each thread. While a parallel reduction is possible (Frigo et al., 2009), we employ a serial reduction as it is sufficiently efficient for typical systems (up to a few hundred cores).

Utility loss. To minimize the utility loss, we take the simple approach that directly computes the utility of a found fair top-k subset and keeps track of the one with the minimum utility loss during the traversal. For obtaining a stable fair weight vector, we extend the post-processing approach for the 2-D k-level-based algorithm (Section 5.2.1) to higher dimensions (see Appendix G.2.2).

5.3 Mixed-integer linear programming-based algorithm

For $n_p = 1$ without an optimization objective, a mixed-integer linear programming-based (MILP-based) algorithm (Cai, 2026a) is used for large k, which is practically efficient despite being theoretically suboptimal. The algorithm uses a binary indicator variable, δ_c, to encode whether a candidate c is within a top-k subset or not. This variable satisfies the following inequalities:

$$-1 \leq w \cdot p(c) - \lambda - \delta_c \leq 0, \tag{6}$$

where $\lambda \in [0, 1]$. The pair (w, λ) can be regarded as a separating hyperplane.

Multiple protected groups. We extend the method for encoding the fairness constraint for one protected group to each protected group. The resulting fairness constraints are formulated as

$$L_k^{\mathcal{G}_j} \leq \sum_{c \in \mathcal{G}_j} \delta_c \leq U_k^{\mathcal{G}_j}, \tag{7}$$

for all $1 \leq j \leq n_p$. All other constraints are kept as-is.

***w* difference.** Since minimizing the w difference can be formulated as a linear program, it is easy to incorporate it into the MILP using the same formulation method (see Appendix G.3).

Utility loss. As noted in Section 2, minimizing the utility loss is equivalent to maximizing the utility of the top-k subset. Thus, the optimization objective can be formulated as follows:

$$\sum_{c \in C} \delta_c (w^o \cdot p(c)). \tag{8}$$

The resulting objective function is linear since $w^o \cdot p(c)$ is fixed given w^o. Thus, it can be directly incorporated into the MILP. Notably, the application of LP relaxation in MILP solvers may yield values of δ_c that are not exactly binary (0 or 1), thereby introducing numerical errors into the objective function. This issue can be mitigated by adjusting solver tolerances appropriately. The same post-processing approach (Section 5.2) is also applied here for obtaining a stable fair weight vector.

6 EXPERIMENTS

In this section, we present experimental results and discuss our empirical findings from them.

6.1 EXPERIMENTAL SETUP

In our experiments, an unfair weight vector w^o is provided as an input, along with a parameter ϵ to control allowable deviation. Formally, for a w^o, the goal is to find a fair weight vector w^f such that $|w_i^f - w_i^o| \leq \epsilon$ for all $1 \leq i \leq d$ while minimizing $g^o(w^f)$ (w difference or utility loss). A failure is reported if no such w^f exists. We used two real-world datasets, ***COMPAS*** ($n = 7214$, $d \leq 6$, $n_p \leq 3$) (ProPublica, 2014) and ***IIT-JEE*** ($n = 384\,977$, $d \leq 3$, $n_p \leq 2$) (IIT, 2009), with preprocessing as needed. 2DRAYSWEEP and ATC_+ from (Asudeh et al., 2019) were implemented as baselines, with several enhancements incorporated. Notably, the need for generating input unfair weight vectors drives our hardness analysis (Section 3). See Appendix H.1 for the complete setup.

6.2 VALIDATION AND STABILITY EXPERIMENTS

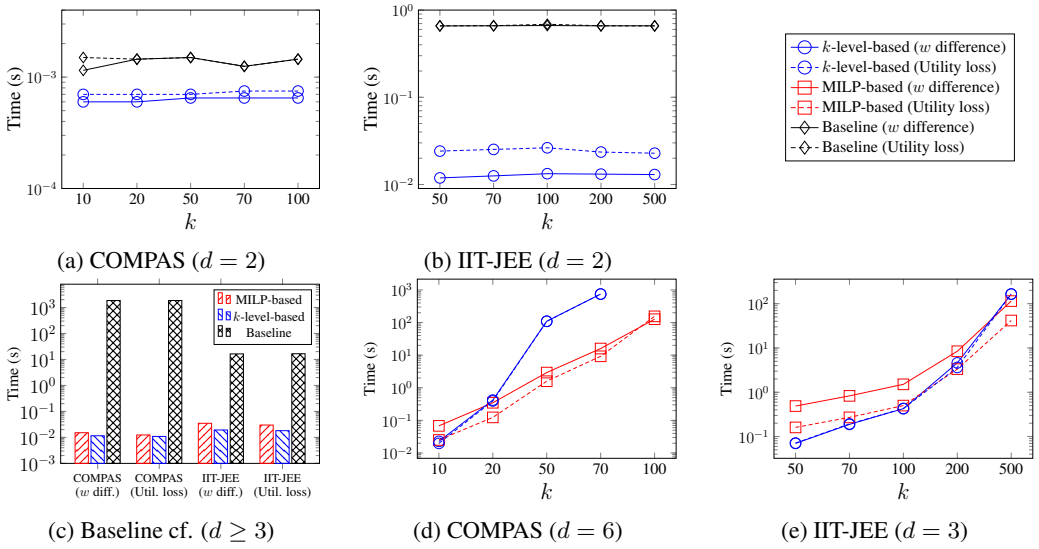
We first validate that our algorithms indeed minimize the disparity, in contrast to prior unaugmented algorithms (Cai, 2026a). Table 1 compares the optimal w differences and utility losses achieved by our augmented algorithms (both achieved the same results) with those reported in (Cai, 2026a) (see Appendix H.3 for further results). Notably, an adapted backtracking subroutine was used to compute utility losses, which was one of the reasons to make it numerically more robust (Section 5.1). Compared to the (unaugmented) MILP-based algorithm, improvements in the two objectives were clear. Compared to the (unaugmented) k-level-based algorithm, the improvement in the w difference was evident; however, the improvement in the utility loss was relatively modest, suggesting its breadth-first search approach is effective in finding a fair weight vector with a small utility loss.

As for stability experiments, we evaluate whether an output fair weight vector remain fair under small perturbations for the 6-D COMPAS dataset with the default setup. For each fair weight vector, four randomly perturbed weight vectors were generated with an L_1 distance of 10^{-4} from it. A fair weight vector passes the stability test if all its perturbed weight vectors remain fair. In our experiments, all 10 fair weight vectors found under the utility loss objective (with stabilization) passed the stability test. In contrast, all fair weight vectors found under the w difference objective failed. These results suggest that optimizing for the utility loss can yield more stable fair weight vectors than optimizing for a distance-based objective, aligning with our theoretical analysis (Section 4.2).

Table 1: Validation results for COMPAS dataset ($k = 50$).

ϵ	Fairness constraint	w difference			Utility loss		
		k-level	MILP	Optimal (ours)	k-level	MILP	Optimal (ours)
0.05	$[34\%, 66\%]$	0.149	0.211	**0.060**	0.476%	1.210%	**0.392%**
	$[40\%, 60\%]$	0.144	0.213	**0.073**	0.449%	1.328%	**0.347%**
	$[44\%, 56\%]$	0.155	0.202	**0.086**	0.718%	1.319%	**0.656%**
0.025	$[40\%, 60\%]$	0.088	0.090	**0.044**	0.121%	0.142%	**0.086%**
0.05		0.144	0.213	**0.073**	0.449%	1.328%	**0.347%**
0.075		0.247	0.308	**0.162**	2.416%	3.187%	**2.161%**

6.3 RUNTIME EXPERIMENTS AND ALGORITHM SELECTION



(a) COMPAS ($d = 2$) (b) IIT-JEE ($d = 2$)

(c) Baseline cf. ($d \geq 3$) (d) COMPAS ($d = 6$) (e) IIT-JEE ($d = 3$)

Figure 3: Representative runtime experimental results.

Figure 3 shows representative runtime experimental results. We summarize our key observations and discuss the algorithm selection. See Appendix H.2 for the complete results and their analysis.

2-D experiments. Our 2-D k-level based algorithm performed consistently better than the baseline algorithm, achieving speedups of up to 50x and 28x for minimizing the w difference and the utility loss, respectively. Its performance trends seen by varying variables were mostly consistent with the previous observations from $n_p = 1$ experiments (Cai, 2026a), which evaluated an unaugmented algorithm. The run times also remained stable across k for both datasets, as the sweeping process accounted for only a small fraction of the total run time. This reinforced our choice of the unidirectional sweeping approach. The utility loss minimization incurred greater overhead than the w difference minimization, partly because the backtracking subroutine for utility loss minimization was engineered more conservatively. The same subroutine was also used in the validation experiments, raising the importance of ensuring its correctness and accuracy. Nevertheless, the augmented k-level-based algorithm was still substantially faster than the baseline for minimizing the utility loss.

Multi-dimensional experiments. Our two algorithms showed speedups of several orders of magnitude comparing to the baseline. The performance trends of both algorithms were also mostly consistent with the observations from $n_p = 1$ experiments. Unlike the 2-D case, the run times of the k-level-based algorithm under the two optimization objectives were nearly identical across all experiment settings. This was because the additional overhead applies only to fair cells, which were few in number in our experiments. In contrast, the MILP-based algorithm was noticeably slower in minimizing the w difference than in minimizing the utility loss due to a larger number of variables.

Regarding selecting between the two algorithms of the two-pronged solution, the general guideline, that one selects the k-level-based algorithm for small k and the MILP-based algorithm for large k, remains valid. However, the threshold value of k is now influenced by the choice of optimization objective, as it is likely to be higher for w difference minimization than for utility loss minimization.

7 CONCLUSION

We presented an integrated study of the fair top-k selection problem in a generalized setting. While one may read our work by extracting the locally defined contribution of each individual component and ignoring its nonlinear structure (Figure 1), we believe the true crux of this work lies in how these components are interconnected. It is these connections that lead to new insights into the problem and enable our superior solution. Ultimately, **the whole is greater than the sum of its parts**.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, specifically to condense text to meet the page limit. We have reviewed all AI-assisted work and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The source code, data, and container have been made available at `https://anonymous.4open.science/r/fair-topk-general-anonymous`. This includes the preprocessed datasets (provided solely for reproducibility) and the data processing code. A command-line interface (CLI) tool is provided for reproducing all experimental results, along with command-line output samples in the repository's README. Furthermore, the CLI tool supports experimental configurations beyond those reported in the paper.

REFERENCES

Artur Andrzejak and Komei Fukuda. Optimization over k-set polytopes and efficient k-set enumeration. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures (WADS)*, pp. 1–12, 1999.

Abolfazl Asudeh, H. V. Jagadish, Gerome Miklau, and Julia Stoyanovich. On obtaining stable rankings. *Proceedings of the VLDB Endowment (VLDB)*, 12(3), 2018.

Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, pp. 1259–1276, 2019.

Julien Basch, Leonidas J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proceedings of the Fourth Annual European Symposium on Algorithms (ESA)*, pp. 302–319, 1996.

Surender Baswana, Partha Pratim Chakrabarti, Sharat Chandran, Yashodhan Kanoria, and Utkarsh Patange. Centralized admissions for engineering colleges in india. *INFORMS Journal on Applied Analytics*, 49(5), 2019.

Kathleen Cachel and Elke Rundensteiner. Group fair rated preference aggregation: Ties are (mostly) all you need. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, pp. 660–678, 2025.

Guangya Cai. Finding a fair scoring function for top-k selection: From hardness to practice. In *Proceedings of the 42nd International Symposium on Computational Geometry (SoCG)*, pp. 26:1–26:17, 2026a.

Guangya Cai. Finding a fair scoring function for top-k selection: From hardness to practice, 2026b. arXiv:2503.11575.

Felix S. Campbell, Alon Silberstein, Julia Stoyanovich, and Yuval Moskovitch. Query refinement for diverse top-k selection. *Proceedings of the ACM on Management of Data (SIGMOD)*, 2(3): 1–27, 2024.

L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. Ranking with fairness constraints. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 28:1–28:15, 2018.

Timothy M. Chan. Remarks on k-level algorithms in the plane, 1999. Manuscript. Available at `https://tmc.web.engr.illinois.edu/lev2d_7_7_99.pdf`.

Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 1032–1041, 2019.

Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. Why not yet: Fixing a top-k ranking that is not fair to individuals. *Proceedings of the VLDB Endowment (VLDB)*, 16(9):2377–2390, 2023.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Matroids, matchings, and fairness. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 2212–2220, 2019.

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 (ITCS)*, pp. 214–226, 2012.

Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1):66–104, 1990.

Seyed Esmaeili, Sharmila Duppala, Davidson Cheng, Vedant Nanda, Aravind Srinivasan, and John P. Dickerson. Rawlsian fairness in online bipartite matching: Two-sided, group, and individual. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 37, pp. 5624–5632, 2023.

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 (CACM)*, 64(4):136–143, 2021.

Matteo Frigo, Pablo Halpern, Charles E. Leiserson, and Stephen Lewin-Berlin. Reducers and other Cilk++ hyperobjects. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures (SPAA)*, pp. 79–90, 2009.

Usman Gohar and Lu Cheng. A survey on intersectional fairness in machine learning: notions, mitigation, and challenges. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6619–6627, 2023.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. `https://www.gurobi.com`, 2024.

Dan Halperin and Micha Sharir. Arrangements. In *Handbook of discrete and computational geometry*, pp. 723–762. Chapman and Hall/CRC, 2017.

Martin Henk, Jürgen Richter-Gebert, and Günter M Ziegler. Basic properties of convex polytopes. In *Handbook of discrete and computational geometry*, pp. 383–413. Chapman and Hall/CRC, 2017.

IIT. IIT-JEE, 2009. Retrieved from `https://indiankanoon.org/doc/1955304`.

Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences (JCSS)*, 62(2):367–375, 2001.

Jon Kleinberg and Manish Raghavan. Selection problems in the presence of implicit bias. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*, pp. 33:1–33:17, 2018.

Matthäus Kleindessner, Michele Donini, Chris Russell, and Muhammad Bilal Zafar. Efficient fair PCA for fair representation learning. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 5250–5270, 2023.

Youjin Kong. Are "intersectionally fair" AI algorithms really fair to women of color? a philosophical analysis. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, pp. 485–494, 2022.

Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLoS ONE*, 12(5):e0177459, 2017.

Hao Liu, Raymond Chi-Wing Wong, Zheng Zhang, Min Xie, and Bo Tang. Fair top-k query on alpha-fairness. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 2338–2350, 2024.

Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM (JACM)*, 31(1):114–127, 1984.

Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2005.

ProPublica. Correctional Offender Management Profiling for Alternative Sanctions, 2014. Retrieved from `https://github.com/propublica/compas-analysis`.

Samira Samadi, Uthaipon Tantipongpipat, Jamie H. Morgenstern, Mohit Singh, and Santosh Vempala. The price of fair PCA: One extra dimension. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

SCOTUS. Ricci v. DeStefano (Nos. 07-1428 and 08-328), 530 F. 3d 87, reversed and remanded. `https://www.law.cornell.edu/supct/html/07-1428.ZO.html`, 2009.

Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry (DCG)*, 6:423–434, 1991.

Cheng Sheng and Yufei Tao. Worst-case I/O-efficient skyline algorithms. *ACM Transactions on Database Systems (TODS)*, 37(4):1–22, 2012.

The College Board. 2025 total group SAT suite of assessments annual report. `https://reports.collegeboard.org/media/pdf/2025-total-group-sat-suite-of-assessments-annual-report.pdf`, 2025.

Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Reverse top-k queries. In *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pp. 365–376, 2010.

Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science (TCS)*, 348(2-3):357–365, 2005.

Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pp. 3447–3487, 2018.

Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 1–6, 2017.

Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. Balanced ranking with diversity constraints. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6035–6042, 2019.

Ke Yang, Joshua R. Loftus, and Julia Stoyanovich. Causal intersectionality and fair ranking. In *2nd Symposium on Foundations of Responsible Computing (FORC)*, volume 192, pp. 7:1–7:20, 2021.

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 (CIKM)*, pp. 1569–1578, 2017.

Meike Zehlike, Ke Yang, and Julia Stoyanovich. Fairness in ranking, part I: Score-based ranking. *ACM Computing Surveys*, 55(6):1–36, 2022.

A FURTHER CLARIFICATION ON THE INTEGRATIVE APPROACH

Given the unconventional narrative of this paper (by a conventional narrative, we mean one centered on "idea(s)" in isolation) and our focus on the connections between its components, we highlight some "concrete" connections between these components as suggested in Figure 1:

- **"motivate"**: The need to generate unfair weight vectors as inputs (as required by the problem formulation) for experimental exploration motivates our hardness analysis via the consideration of the Fair Top-k Verification problem (Section 6.1 $\rightarrow$ Section 3).
- **"guides"**: The design of our efficient algorithm for solving the Fair Top-k Verification problem when the number of protected groups and k are small is guided by a structural insight into the setting where the hardness result for small k does not apply (Section 3.2 (Remark 2) $\rightarrow$ Section 4.1).
- **"inform"**: Several experimental observations, including the number of fair cells and the actual $(k-1)$-level structural complexity in 2-D, inform our algorithm design and engineering decisions for the practical k-level-based algorithms (Section 6.3 $\rightarrow$ Section 5.2). The correctness and accuracy requirements of the validation experiments also inform our implementing decisions for the backtracking algorithm (Sections 6.2 $\rightarrow$ Section 5.1).

Note that the list above does not exhaust all the connections (e.g., the connection between the scoring function stability and the geometric structure of the k-level). While we acknowledge that some of the components may be of independent interest, still, we emphasize that their primary significance in this work lies in their interconnection, rather than their locally defined contribution.

B RELATED WORK

Algorithmic fairness has emerged as a prominent research area in computer science, with particular relevance to data processing systems and machine learning. A broad range of research has addressed fairness across diverse tasks, including clustering (Chierichetti et al., 2017; Chen et al., 2019), dimensionality reduction (Samadi et al., 2018; Kleindessner et al., 2023), and matching (Chierichetti et al., 2019; Esmaeili et al., 2023). Since top-k selection is a fundamental task in information retrieval and decision-making, its fairness has also drawn significant attention. Among many different ways to quantifying its fairness (e.g. Dwork et al. (2012); Yang & Stoyanovich (2017); Friedler et al. (2021); Zehlike et al. (2022)), a common one is based on proportional representation. Thus, fair top-k selection under proportional fairness constraints has been widely studied (Zehlike et al., 2017; Kleinberg & Raghavan, 2018; Celis et al., 2018; Asudeh et al., 2019; Yang et al., 2019; Campbell et al., 2024; Cai, 2026a). See Zehlike et al. (2022) for a survey. Notably, intersectional fairness, which considers candidates belonging to multiple different minority or historically disadvantaged groups, has also drawn considerable attention (Yang et al., 2021; Kong, 2022; Gohar & Cheng, 2023), including in the context of fair top-k selection (Yang et al., 2021).

For the problem of fair top-k selection, most prior work (e.g., Zehlike et al. (2017); Celis et al. (2018); Yang et al. (2019); Campbell et al. (2024)) either scores items on a single attribute or precomputes scores from multiple attributes using a fixed, predefined scoring function. In many applications, however, selecting the top-k items requires considering several attributes simultaneously, and choosing an appropriate scoring function over these attributes could be a principled way to promote fairness. Asudeh et al. (2019) proposed algorithms to find a fair linear scoring function if the initial one does not produce a top-k subset meeting the given fairness constraints. Since a linear scoring function can be presented as a weight vector, they aim at searching for a fair weight vector minimizing the angular distance. Their algorithms can work with various fairness models based on proportional fairness constraints, including proportional fairness constraints considered in this work. However, their algorithms do not scale efficiently with the size of dataset, and their run times increase dramatically with dimensionality. Moreover, the secondary criteria considered in the work, angular distance, requires solving expensive non-linear programs. Besides applying proportional fairness constraints, Liu et al. (2024) introduced a fairness metric, alpha-fairness, which is based on the differences between the proportions of protected group members in the top-k subset and their proportions in the entire dataset. They further proposed methods for finding a fair weight vector optimizing this metric. However, their approach lacks a formal runtime analysis (with only an informal estimate of $O(n^{2d-2})$ that omits the costs of some steps) and still scales poorly. Moreover,

the secondary criteria considered in this work is difficult to interpret, as their weight vectors are normalized by the L_1 norm while the distance between the fair and the reference weight vectors is measured by their L_2 distance. In particular, neither work (Asudeh et al., 2019; Liu et al., 2024) properly addresses the potential ties among candidates, even though the way to break ties may affect the fairness of the outcome (Cachel & Rundensteiner, 2025; Cai, 2026a). Moreover, both use a distance metric to measure the disparity, which produces a scoring function that is inherently unstable (see Section 4.2). Cai (2026a) adopted an integrative framework to systematically study the problem of finding a fair weight vector under proportional constraints, resulting in a more efficient two-pronged solution while also offering valuable insights into the problem structure and algorithmic behavior. They also identified and addressed the tie-breaking issue overlooked by the previous work. However, they only considered a single protected group and did not use a secondary criterion for choosing among multiple possible fair scoring functions (returning an arbitrary one among them instead). This limits the applicability of their method in settings where multiple protected groups are common and practical users may wish to choose among multiple feasible fair scoring functions.

C SUMMARY OF NOTATION

Table 2: Summary of notation

Symbol	Definition
$\mathcal{C}$	Set of candidates to be selected
c	Candidate in $\mathcal{C}$
n	Number of candidates, i.e., $\lvert\mathcal{C}\rvert$
d	Number of scoring attributes
$p(c)$	Values of the scoring attributes for c, $p(c) \in \mathbb{R}^d$
$A(c)$	Values of the sensitive attribute for c, an appropriate subset of group labels
$\mathcal{G}_j$	Group label / Subset of $\mathcal{C}$ with $\mathcal{G}_j \in A(c)$
n_p	Number of protected groups
w	Weight vector
w^o	Reference weight vector
w^f	Fair weight vector
τ_k^w	Top-k subset of $\mathcal{C}$ given w and k
$L_k^{\mathcal{G}_j}, U_k^{\mathcal{G}_j}$	Lower and upper bounds on the number of candidates in a top-k subset that are in $\mathcal{G}_j$
V	Weight vector subspace described by a set of linear inequalities
l	Number of linear inequalities describing V, i.e., $\lvert V\rvert$
$g^o(w)$	Disparity of w given w^o
$U^o(\tau_k^w)$	Utility of the top-k subset τ_k^w under w^o
ϵ	Parameter to control allowable deviation from w^o in experiments

D OMITTED DETAILS OF SECTION 2

Our definition of the sensitive attribute is slightly different from that in (Zehlike et al., 2022), where protected groups may be values of different sensitive attributes. Our formulation can be viewed as concatenating all relevant attributes into a single composite sensitive attribute. An illustrative example is given below.

Example 2. *Consider a dataset with gender and race information for each candidate. The sensitive attribute concatenates the gender and race together, such that for each candidate c, its sensitive attribute values, $A(c)$, are represented by an appropriate subset of $\{$Male, Female, $\ldots$, Caucasian, Black, $\ldots\}$. Thus, the sensitive attribute values of a black female candidate are $A(c) = \{$Female, Black$\}$. Furthermore, if both "Female" and "Black" are protected groups, an "artificial" group label, "Black Female" can be constructed and added to the attribute domain. In this case, the sensitive attribute values of a black female candidate become $A(c) = \{$Female, Black, Black Female$\}$.*

Also, as noted in Section 2, there may be multiple top-k subsets for a weight vector w^f due to ties in scores. In such a case, the top-k subset that satisfies the fairness constraints and minimizes the relative utility loss is used to determine the value of $g^o(w^f)$. In fact, minimizing the utility loss is equivalent to maximizing the utility (under w^o), as $U^o(\tau_k^o)$ stays the same regardless of the tie-breaking outcome. An illustrative example is given below.

Example 3. *Consider a dataset of* 5 *candidates with scoring attribute values being* $\{(0.4, 0.7), (0.5, 0.6), (0.7, 0.35), (0.8, 0.2), (0.9, 0.9)\}$ *(sensitive attribute values are omitted). Let* $k = 2$ *and suppose that the given reference weight vector is* $w^o = (0.5, 0.5)$. *Tie-breaking leads to two different top-2 subsets:* $\{(0.9, 0.9), (0.4, 0.7)\}$ *and* $\{(0.9, 0.9), (0.5, 0.6)\}$. *Both top-2 subsets have the same utility* 1.45 *under* w^o. *Now, suppose we obtain a fair weight vector* $w^f = (0.6, 0.4)$, *then we have two possible top-2 subsets due to ties:* $\{(0.9, 0.9), (0.7, 0.35)\}$ *and* $\{(0.9, 0.9), (0.8, 0.2)\}$. *Their utilities under* w^o *are* 1.425 *and* 1.4 *respectively. Assuming both satisfy the fairness constraints, the former subset is used to determine the utility loss of* w^f.

One may note that the utility defined in Section 2 may not be non-negative, which may cause issues in computing the relative utility loss. In practice, however, values of scoring attributes are typically normalized to $[0, 1]^d$, as they are equivalent via uniform translation and scaling (Cai, 2026a) and the normalization makes weights comparable across different scoring attributes. With this normalization, the utility of a top-k subset becomes non-negative.

Finally, we formally define our problem as follows:

Problem 1 ((Generalized) Fair Top-k Selection)**.** *We are given a set of* n *candidates with* d *scoring attributes, one sensitive attribute, and* n_p *protected groups. Each candidate,* c*, is represented by a point* $p(c)$ *in* $\mathbb{R}^d$ *and a subset* $A(c) \subseteq \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. *We are also given a non-negative integer* $k \leq n$*, two sets of non-negative integers* $\{L_k^{\mathcal{G}_j} \mid 1 \leq j \leq n_p\}$ *and* $\{U_k^{\mathcal{G}_j} \mid 1 \leq j \leq n_p\}$*, a subset* V *of the weight vector space described by* l *linear inequalities, and a reference weight vector* w^o *with its corresponding disparity measure* $g^o : V \to \mathbb{R}$ *(w difference or utility loss). The goal is to find a real-valued weight vector* $w = (w_1, w_2, \ldots, w_d) \in V$*, where* $w_i \geq 0$ *for all* i *and* $||w||_1 = 1$*, such that*

$$L_k^{\mathcal{G}_j} \leq |\tau_k^w \cap \mathcal{G}_j| \leq U_k^{\mathcal{G}_j},$$

for all $1 \leq j \leq n_p$*, while also minimizing the value of* $g^o(w)$.

E Omitted details of Section 3

E.1 Fair Top-k Verification

The formal definition of the Fair Top-k Verification problem is given as follows:

Problem 2 (Fair Top-k Verification)**.** *We are given a set of* n *candidates with* d *scoring attributes, one sensitive attribute and* n_p *protected groups. Each candidate,* c*, is represented by a point* p *in* $\mathbb{R}^d$ *and a subset* $A(c) \subseteq \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. *We are also given a non-negative integer* $k \leq n$*, two sets of non-negative integers* $\{L_k^{\mathcal{G}_j} \mid 1 \leq j \leq n_p\}$ *and* $\{U_k^{\mathcal{G}_j} \mid 1 \leq j \leq n_p\}$*, and a weight vector* w*. The goal is to determine whether there exists a top-*k *subset* τ_k^w *of* w *such that*

$$L_k^{\mathcal{G}_j} \leq |\tau_k^w \cap \mathcal{G}_j| \leq U_k^{\mathcal{G}_j},$$

for all $1 \leq j \leq n_p$.

At first glance, the problem appears trivial: one can use w to compute a score for each candidate, apply a selection algorithm to identify the top-k candidates, and then count the number of selected candidates of each protected group to verify whether the fairness constraints are satisfied. However, as noted in Example 3, the top-k subset may not be unique due to ties in scores. In such cases, it is no longer obvious how to efficiently determine whether there exists a top-k subset satisfying the fairness constraints. The problem is computationally easier than the Fair Top-k Selection problem, as an algorithm to the latter problem can be used to solve the former. In fact, this problem can be regarded as the Fair Top-k Selection problem when $d = 1$, treating the value $w \cdot p(c)$ as the single scoring attribute value. While the specific weight assigned to this attribute becomes irrelevant, the algorithm can effectively determine whether there exists a top-k subset that satisfies the fairness constraints. Consequently, any lower bounds established for the Fair Top-k Verification problem can be applied to the Fair Top-k Selection problem.

E.2 Proof of Theorem 1

Proof. For an arbitrary instance of the Set Cover problem, we are given a universe $\mathcal{U} = \{u_1, u_2, \ldots, u_{n_p}\}$ and a collection $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ where each s_ℓ ($1 \leq \ell \leq n$) is a subset of $\mathcal{U}$. Our goal is to find the smallest subset of $\mathcal{S}$ whose union equals $\mathcal{U}$.

Now, we construct an instance of Fair Top-k Verification with the given instance of Set Cover. For each element u_j in U, a group $\mathcal{G}_j$ is created that is also a protected group in the Fair Top-k Verification problem. Then, for each subset s_ℓ in S such that $s_\ell = \{u_x, u_y, u_z, \ldots\}$, we create a candidate c with $A(c) = \{\mathcal{G}_x, \mathcal{G}_y, \mathcal{G}_z, \ldots\}$. These give us a set of n candidates. The values of the scoring attributes are assigned to be the same across different candidates (e.g., $p(c) = (0.5, 0.5)$ for all c). Now we have an instance of the Fair Top-k Verification problem in 2-D, with n candidates and n_p groups, and all groups are protected groups. To solve the given instance of the Set Cover problem, we set $L_k^{\mathcal{G}_j} = 1$ and $U_k^{\mathcal{G}_j} = k$ for all $1 \leq j \leq n_p$, and run the algorithm for the Fair Top-k Verification problem with an arbitrary w (e.g., $w = (0.5, 0.5)$) for all $1 \leq k \leq n$. The smallest k that successfully verifies the fairness of the weight vector is reported. Clearly, this reduction takes polynomial time.

To show the correctness of this reduction, note that for the given fairness constraints, if one successfully verifies the fairness of the given weight vector for a specific k value, we will be able to derive a top-k subset τ_k^w of the size k such that each protected group, $\mathcal{G}_j$, must be covered by at least one $A(c)$ of $c \in \tau_k^w$. Moreover, consider the one-to-one correspondence between an element u_j in the universe and a protected group $\mathcal{G}_j$, and between a subset s_ℓ of S and a candidate c. A sub-collection of S of size k whose union covers all elements in $\mathcal{U}$ can be easily derived from τ_k^w. In such cases, the smallest sub-collection of S which covers $\mathcal{U}$ corresponds to a solution to the Fair Top-k Selection problem with the smallest possible k.

There is one subtlety that we need to discuss. An algorithm for the Fair Top-k Verification problem can only confirm that a weight vector is fair if there exists a τ_k^w satisfying the fairness constraints. However, for this input w, it is not obvious how to derive a solution of the Set Cover problem as the τ_k^w satisfying the fairness constraints is still unknown. Fortunately, the decision version of the Set Cover problem, which only considers the size of the smallest sub-collection of S that covers U, is known to be NP-hard, which is enough for our purpose. □

E.3 Proof of Theorem 2

To begin with, let us introduce two problems, Orthogonal Vectors (OV) and its generalization t-OV (often referred to as k-OV in the literature (Williams, 2018)), whose hardness assumptions play key roles in fine-grained complexity and which we will use in our reductions.

The OV problem is defined as follows: Let $n_p = \alpha \log n$. Given two sets $A, B \subseteq \{0,1\}^{n_p}$ with $|A| = |B| = n$, the goal is to determine whether there exist $a \in A, b \in B$ so that $a \cdot b = 0$ where $a \cdot b = \sum_{i=1}^{n_p} a_i \cdot b_i$. For such a problem, we have the following conjecture (Williams, 2005; 2018):

Hypothesis 1 (OV Hypothesis)**.** *For every constant $\delta > 0$, there is a constant $\alpha > 0$ such that OV cannot be solved in $O(n^{2-\delta})$ time on instances with $n_p = \alpha \log n$.*

An extension of the OV problem, t-OV problem, is defined as follows: Let t be a constant and $n_p = \alpha \log n$. Given t sets $A_1, A_2, \ldots, A_t \subseteq \{0,1\}^{n_p}$ with $|A_i| = n$ for all i, the goal is to determine whether there exist $a^1 \in A_1, a^2 \in A_2, \ldots a^t \in A_t$ so that $a^1 \cdot a^2 \cdot \ldots \cdot a^t = 0$ where $a^1 \cdot a^2 \cdot \ldots \cdot a^t = \sum_{i=1}^{n_p} \prod_{\ell=1}^{t} a_i^\ell$. For such a problem, we have the conjecture as follows:

Hypothesis 2 (t-OV Hypothesis)**.** *For every constant $\delta > 0$, there is a constant $\alpha > 0$ such that t-OV cannot be solved in $O(n^{t-\delta})$ time on instances with $n_p = \alpha \log n$.*

With the two hypotheses introduced, we begin by establishing the following (conditional) lower bound based on the OV hypothesis.

Theorem 5. *For every constant $\delta > 0$, there is a constant $\alpha > 0$ such that Fair Top-k Verification cannot be solved in $O(n^{2-\delta})$ time for $n_p = \alpha \log n$, assuming the OV hypothesis.*

Proof. Given the two sets $A, B \subseteq \{0,1\}^{n_p}$, we first create a set of groups for our instance of the Fair Top-k Verification problem, denoted as $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{n_p}, \mathcal{G}_{n_p+1}, \mathcal{G}_{n_p+2}\}$. All groups are

protected groups in our instance. For each element a in A such that $a = (a_1, a_2, \ldots, a_{n_p})$, we create a candidate c with $A(c) = \{a_1 \cdot \mathcal{G}_1, a_2 \cdot \mathcal{G}_2, \ldots, a_{n_p} \cdot \mathcal{G}_{n_p}, \mathcal{G}_{n_p+1}\}$, where $a_i \cdot \mathcal{G}_i$ means $\mathcal{G}_i$ is an element of $A(c)$ if $a_i = 1$ and is not an element of $A(c)$ if $a_i = 0$. Similarly, for each element b in B such that $b = (b_1, b_2, \ldots, b_{n_p})$, we create a candidate c with $A(c) = \{b_1 \cdot \mathcal{G}_1, b_2 \cdot \mathcal{G}_2, \ldots, b_{n_p} \cdot \mathcal{G}_{n_p}, \mathcal{G}_{n_p+2}\}$. Again, as was done in the proof of Theorem 1, the values of scoring attributes are assigned to be the same across different candidates (e.g., $p(c) = (0.5, 0.5)$ for all c). Then, we set $L_k^{\mathcal{G}_j} = 0$ and $U_k^{\mathcal{G}_j} = 1$ for all $1 \leq j \leq n_p$, with $L_k^{\mathcal{G}_{n_p+1}} = 1$ and $L_k^{\mathcal{G}_{n_p+2}} = 1$ while $U_k^{\mathcal{G}_{n_p+1}} \geq 1$ and $U_k^{\mathcal{G}_{n_p+2}} \geq 1$ are set arbitrarily. Finally, we set $k = 2$ and take an arbitrary input weight vector (e.g. $w = (0.5, 0.5)$). This gives us an instance of the Fair Top-k Verification problem with $2n$ candidates and $n_p + 2$ protected groups. Clearly, the reduction takes $O(n \log n)$ time with $n_p = O(\log n)$, as the time is bounded by $O(n \cdot n_p)$ for creating candidates.

To show the correctness of this reduction, note that if one successfully verifies the fairness of the weight vector for such an instance of the Fair Top-k Selection problem (with $k = 2$), one must also be able to derive a set of two candidates, c_1 and c_2, such that $A(c_1) \cap A(c_2) = \emptyset$. Moreover, $\mathcal{G}_{n_p+1}$ and $\mathcal{G}_{n_p+2}$ must be an element of either $A(c_1)$ or $A(c_2)$. Assume without the loss of the generality that $\mathcal{G}_{n_p+1} \in A(c_1)$ and $\mathcal{G}_{n_p+2} \in A(c_2)$. By our construction, this means that c_1 corresponds to an element in A and c_2 corresponds to an element in B. Next, one can recover the corresponding element $a \in A$ by $A(c_1)$ by setting $a_i = 1$ if $\mathcal{G}_i \in A(c_1)$ and $a_i = 0$ otherwise. The element $b \in B$ can also be recovered in the same way. Given $A(c_1) \cap A(c_2) = \emptyset$, we now have $a \cdot b = 0$ as $a_i \cdot b_i = 0$ for all $1 \leq i \leq n_p$.

Conversely, if the algorithm reports a failure, that means there does not exist $a \in A, b \in B$ such that $a \cdot b = 0$, as each candidate corresponds exactly one element in either A or B by our construction. This completes the proof. □

Now, using a similar approach to the one in Theorem 5, we are able to prove Theorem 2.

Proof of Theorem 2. Given an arbitrary instance of the t-OV problem, we set $t = k$. For the k sets $A_1, A_2, \ldots, A_k \subseteq \{0,1\}^{n_p}$, we first create a set of groups for our instance of the Fair Top-k Selection problem, denoted as $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{n_p}, \mathcal{G}_{n_p+1}, \mathcal{G}_{n_p+2}, \ldots, \mathcal{G}_{n_p+k}\}$, where all groups are protected. For each element a in A_ℓ such that $a = (a_1, a_2, \ldots, a_{n_p})$, we create a candidate c with $A(c) = \{a_1 \cdot \mathcal{G}_1, a_2 \cdot \mathcal{G}_2, \ldots, a_{n_p} \cdot \mathcal{G}_{n_p}, \mathcal{G}_{n_p+\ell}\}$. Then, we set $L_k^{\mathcal{G}_j} = 0$ and $U_k^{\mathcal{G}_j} = k - 1$ for all $1 \leq j \leq n_p$, as well as $L_k^{\mathcal{G}_{n_p+\ell}} = 1$ for all $1 \leq \ell \leq k$. Finally, similar to the proof of Theorem 5, $U_k^{\mathcal{G}_{n_p+\ell}} \geq 1$ are arbitrarily set for all $1 \leq \ell \leq k$, values of scoring attributes are assigned to be the same across different candidates, and an arbitrary weight vector is used as an input. Clearly, the reduction takes $O(nk \cdot n_p) = O(n \log n)$ time, as $k = O(1)$ and $n_p = O(\log n)$. Note that in our reduction, there are $n_p + k$ protected groups rather than n_p, but since $k = O(1)$, the extra k protected groups can be absorbed by choosing a slightly larger α.

To show the correctness of this reduction, note that if one successfully verifies the fairness of the weight vector for such an instance of the Fair Top-k Verification problem, one must be able to derive a set of k candidates, C', such that for all $1 \leq j \leq n_p$, $\mathcal{G}_j$ can only be an element of $A(c)$ of at most $k - 1$ candidates in C'. Moreover, for each $1 \leq \ell \leq k$, $\mathcal{G}_{n_p+\ell}$ can only be an element of $A(c)$ of exactly one candidate in C', as the t-OV problem requires selecting exactly one binary vector from each binary vector set. By the same method applied in Theorem 5, one can recover k elements $a^1 \in A_1, a^2 \in A_2, \ldots a^k, \in A_k$ such that $a^1 \cdot a^2 \cdot \ldots \cdot a^k = 0$. This is because for all $1 \leq i \leq n_p$, we have $\prod_{\ell=1}^{k} a_i^\ell = 0$ as there are at most $k-1$ $a_i^\ell = 1$ among all possible $1 \leq \ell \leq k$. Conversely, if the algorithm reports a failure, there must not exist such a set of k binary vectors, given the one-to-one correspondence between a candidate and an element in the given k sets $A_1, A_2, \ldots, A_k$. □

F Omitted details of Section 4

F.1 Omitted details of Section 4.1

Before describing the algorithm in detail, let us revisit its main idea. To break the established lower bound, we leverage the key observation that when $n_p = O(1)$, many candidates become

indistinguishable with respect to their protected groups membership profile, which represents all protected groups to which a candidate belongs. These candidates are "exchangeable" for a top-k subset that satisfies the fairness constraints, as swapping a selected candidate with an unselected (and tied) one of the same protected groups membership profile does not change the protected group member counts of the top-k subset. Thus, potentially many top-k subsets are equivalent for the fairness test, so they can be bypassed for runtime improvements.

Now, let us proceed to the algorithm. First, an arbitrary top-k subset is identified using a selection algorithm, with the score of each candidate under the given w being computed. Let τ_k denotes this top-k subset, and let $c_k \in \tau_k$ be the candidate with the lowest score in τ_k, that is, the kth candidate under w in the candidate set C. Next, two sets, $M_1 \subseteq \tau_k$ and $M_2 \subseteq C \setminus \tau_k$, are identified such that every candidate c in M_1 or M_2 has the same score as c_k (i.e., $w \cdot p(c) = w \cdot p(c_k)$ for all $c \in M_1$ and $c \in M_2$). Notably, for every candidate in $\tau_k \setminus M_1$, their scores are strictly higher than that of c_k so that they must appear in the top-k subset regardless of the tie-breaking outcome. Therefore, we initialize the protected group member counts by $\tau_k \setminus M_1$.

Since there are $k-|M_1|$ candidates that are determined, our next step is to select $t = |M_1|$ candidates among $M_1 \cup M_2$ such that the fairness constraints are satisfied. Recall our observation that two candidates with the same protected groups membership profile are equivalent for a fair top-k subset. With n_p protected groups, there are 2^{n_p} distinct protected groups membership profiles. We encode each membership profile using a binary vector converted into an integer, denoted by the function $I(\cdot)$. For example, assuming $n_p = 5$, a candidate with membership profile $A(c) = \{\mathcal{G}_1, \mathcal{G}_3, \mathcal{G}_5\}$ can be encoded into a binary vector 10101, which is further converted as $I(A(c)) = 21$. Specially, if a candidate does not belong to any protected group, $I(A(c)) = 0$. The inverse of this operation, which coverts an integer to a protected groups member profile, is denoted as $I^{-1}(\cdot)$. In fact, a similar operation that converts a binary vector into a membership profile is used in the proof of Theorem 5. Since candidates with the same protected groups membership profile are equivalent for the fairness test, only the number of candidates selected from each membership profile matters. Let m_j denote the number of candidates with $I(A(c)) = j$, for a top-k subset, one must have values of n_j such that $\sum_{j=0}^{2^{n_p}-1} m_j = t$. All valid value assignments to m_j whose sum is t can be enumerated using a backtracking algorithm.

Algorithm 1 provides the pseudocode for our algorithm, while Algorithm 2 provides the pseudocode for the backtrack subroutine. We use a dictionary N to maintain the member counts for each protected group during the tree search. The array S stores the total number of candidates for each protected groups membership profile, while S' maintains the number of selected candidates for each profile during the search. Whenever the tree search reaches a leaf node, that is, when exactly t candidates are selected, we use N to test against the input fairness constraints $L = \{L_k^{\mathcal{G}_j} \mid 1 \le j \le n_p\}$ and $U = \{U_k^{\mathcal{G}_j} \mid 1 \le j \le n_p\}$. The tree search backtracks whenever the leaf node does not yield a top-k subset satisfying the fairness constraint, or the number of candidates to be selected for a membership profile exceeds the limit specified in S.

F.1.1 Proof of Theorem 3

Proof. Our algorithm starts by using a selection algorithm to obtain an arbitrary top-k subset, which takes $O(n \cdot d)$ time. In the backtracking subroutine, each node of the search tree has at most $t + 1$ children, as there are at most t candidates to be selected. The depth of the search tree is bounded by 2^{n_p}, as each level corresponds to a unique protected groups membership profile.

At each node, updating N takes $O(n_p)$ time, and if the node is a leaf, checking the fairness constraints also takes $O(n_p)$. Since $n_p = O(1)$, these are constant-time operations. Therefore, the total runtime of the backtracking subroutine is $O(t^{2^{n_p}})$. Given that $n_p = O(1)$, 2^{n_p} is also a constant. Since $t \le k$, for a sufficiently small k (e.g., $k = \text{polylog}(n)$), the term $O(t^{2^{n_p}})$ is $o(n \cdot d)$. (E.g., when $k = \text{polylog}(n)$, we have $O(t^{2^{n_p}}) = O(\text{polylog}(n)) = o(n \cdot d)$ since $t \le k$ and $2^{n_p} = O(1)$.) As all other operations take $O(n)$ time (with $n_p = O(1)$), the initial selection step dominates the computational cost, resulting in an overall time complexity of $O(n \cdot d)$. □

Algorithm 1 Algorithm for Fair Top-k Verification

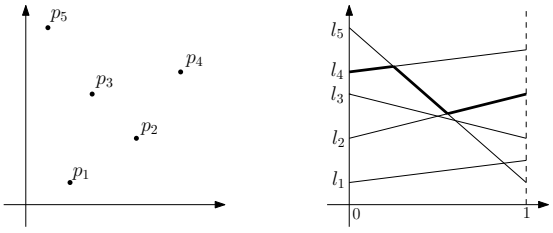
procedure FAIRTOPKVERIFICATION(C, k, n_p, L, U, w)
  $\tau_k \leftarrow$ SELECT(C, k, w)
  $c_k \leftarrow$ the kth candidate under w
  $M_1 \leftarrow \{c \mid c \in \tau_k$ and $w \cdot p(c) = w \cdot p(c_k)\}$
  $M_2 \leftarrow \{c \mid c \in C \setminus \tau_k$ and $w \cdot p(c) = w \cdot p(c_k)\}$
  Let N be a dictionary storing protected group member counts, initialized to 0
  for $c \in \tau_k \setminus M_1$ **do**
    for $g \in A(c)$ **do**
      if g is protected **then** $N[g] \leftarrow N[g] + 1$
    end for
  end for
  Let S be an integer array of size 2^{n_p}, initialized to 0
  for $c \in M_1 \cup M_2$ **do**
    $S[I(A(c))] \leftarrow S[I(A(c))] + 1$
  end for
  return BACKTRACKTIEBREAKING($|M_1|$, n_p, L, U, N, S)
end procedure

Algorithm 2 Backtracking algorithm for tie-breaking

procedure BACKTRACKTIEBREAKING(t, n_p, L, U, N, S)
  Let T be a stack to store pairs of integers, initialized to empty
  Let S' be an array of size 2^{n_p}, initialized to -1
  T.PUSH($\{t, 0\}$)
  while T is not empty **do**
    $\{r, i\} \leftarrow T$.TOP()
    if $i \leq 2^{n_p} - 1$ **then**
      if $S'[i] < S[i]$ **then**
        $s' \leftarrow$MAX($S'[i]$, 0); $S'[i] \leftarrow S'[i] + 1$; $r \leftarrow r - S'[i]$
        for $g \in I^{-1}(i)$ **do**
          $N[g] \leftarrow N[g] + (S'[i] - s')$
        end for
      else
        REVERT(N, S', i); $S'[i] \leftarrow -1$; T.POP() ▷ Restore to the previous state
        continue
      end if
    else T.POP(); **continue**
    end if
    if $r = 0$ **then**
      if ISFAIR(L, U, N) **then**
        return True
      else
        REVERT(N, S', i); $S'[i] \leftarrow -1$; T.POP() ▷ Restore to the previous state
      end if
    else T.PUSH($\{r, i + 1\}$)
    end if
  end while
  return False
end procedure

F.2 DUAL TRANSFORMATION AND k-LEVEL

We give a brief overview of the computational geometry concepts and techniques (e.g., k-level) used in the k-level-based algorithm. The materials are from (Cai, 2026b) and one may refer to it for a better context.



(a) Primal space (b) Dual space

Figure 4: Dual transformation in 2-D, mapping a point p_i to a line l_i. Each (open) line segment is a cell in 2-D and bold line segments in (b) show the k-level for $k = 1$ in 2-D. The figure is obtained from (Cai, 2026b).

For the k-level-based algorithm, a dual transformation is employed which transforms points in $\mathbb{R}^d$ into $(d-1)$-dimensional hyperplanes in $\mathbb{R}^d$. Let $p = (p_1, p_2, \ldots, p_d)$ be values of the scoring attributes in $\mathbb{R}^d$ of a candidate $c \in C$. We transform p into a hyperplane $H(p)$ defined by

$$(p_1 - p_d)x_1 + (p_2 - p_d)x_2 + \cdots + (p_{d-1} - p_d)x_{d-1} - x_d + p_d = 0, \tag{9}$$

where $x_1, x_2, \cdots, x_d$ are variables. One can rewrite it as

$$x_d = p_1x_1 + p_2x_2 + \cdots + p_{d-1}x_{d-1} + (1 - \sum_{i=1}^{d-1} x_i)p_d. \tag{10}$$

An example of this primal-to-dual transformation for a set of points in 2-D is provided in Figure 4.

Recall that for any weight vector $w = (w_1, w_2, \ldots, w_d)$ with $w_i \geq 0$ and $|w||_1 = 1$, we have $f(p) = \sum_{i=1}^{d} w_i p_i = \sum_{i=1}^{d-1} p_i w_i + (1 - \sum_{i=1}^{d-1} w_i)p_d$. Observe from Equation (10) that $f(p)$ is actually the d-th coordinate of the projection of point $(w_1, w_2, \ldots, w_{d-1})$ on $H(p)$. Now, consider two points p and p'. If $f(p) > f(p')$ for some weight vector w, then in the dual space, the downward-directed ray from $(w_1, w_2, \ldots, w_{d-1}, +\infty)$ hits $H(p)$ before $H(p')$. The transformation thus preserves, in dual space, the relative order between each pair of points in primal space, with respect to their scores for any weight vector. Notably, a tie in scores, where $f(p) = f(p')$, corresponds to an intersection between $H(p)$ and $H(p')$ in the dual space.

Next, consider the subdivision of $\mathbb{R}^d$ induced by a collection of hyperplanes, i.e., an *arrangement*. We say that a point of $\mathbb{R}^d$ is at *level* k if exactly k hyperplanes lie strictly above it in the positive x_d direction. A *cell* on a hyperplane is a maximal connected region of the hyperplane that is not intersected by other hyperplanes. The *k-level* consists of the closure of all cells whose interior points have level k. It is a monotone, piecewise-linear surface. (See Figure 4b for an example in 2-D.) The *$\leq k$-level* is the part of the arrangement lying on or above the k-level. Finally, a *j-dimensional cell* ($0 \leq j \leq d-1$) is a maximal connected region of the hyperplane of dimension j in the intersection of a subset of the hyperplanes that is not intersected by any other hyperplane (see (Halperin & Sharir, 2017)). One important property of this k-level is that if a hyperplane has a cell that is part of the $(k-1)$-level, then the primal point corresponding to the hyperplane is the kth point for some weight vectors, since there are exactly $k-1$ points with higher scores. Moreover, if two weight vectors whose corresponding downward-directed rays in the dual space hit the same cell of the $(k-1)$-level, their top-k subsets are the same. Finally, the top-k subsets of adjacent cells differ in at most one point since there is at most one hyperplane that enters the $\leq (k-1)$-level and becomes part of the $(k-1)$-level.

F.3 Omitted details of Section 4.2

Multiple protected groups. In (Cai, 2026a), the theoretically efficient k-level-based algorithm for $n_p = 1$ (without an optimization objective) works by traversing the cells of the $(k-1)$-level (see Appendix F.2 for the computational concepts and techniques used in the algorithm). During the traversal, it keeps track of the number of protected group members in the top-k subset, and uses this count to test against the fairness constraint. With multiple protected groups, we adapt the method

for the single protected group to keep track of the numbers of members for each protected group, and use them to test against the fairness constraints whenever one of the numbers changes.

In the presence of ties, note that a top-k subset with ties typically corresponds to a low-dimensional cell of the $(k-1)$-level, which is formed by the intersection of hyperplanes (see Appendix F.2). To facilitate efficient calls to the backtracking subroutine for finding the top-k subset satisfying fairness constraints in the presence of ties, we additionally track, for each protected group, the number of hyperplanes that lie strictly "above" the current $(k-1)$-level cell in the dth dimension, as such a hyperplane corresponds to a candidate with a strictly higher score than the kth one (see Appendix F.2). Moreover, each $(k-1)$-level cell also stores auxiliary information about its intersecting hyperplanes, allowing the number of tied candidates for each protected groups membership profile to be identified efficiently.

w difference. Note that the fair cell $\mathcal{F}$ can be regarded as the intersection of the positive halfspaces induced by the hyperplanes in $\mathcal{H}$. Recall in Section 4.2, we have the following LP formulation:

$$\begin{aligned} \min \ & \sum_{i=1}^{d} \phi_i \\ \text{s.t.} \ & \sum_{i=1}^{d-1} h_i w_i + h_d \geq 0, \qquad \forall h \in \mathcal{H} \\ & \phi_i \geq w_i - w_i^o, \qquad i = 1, \ldots, d \\ & \phi_i \geq w_i^o - w_i, \qquad i = 1, \ldots, d \end{aligned} \tag{11}$$

Notably, constraints on the weight vector (non-negativity and $||w||_1 = 1$) are omitted from the formulation to simplify the presentation. In this linear program, each ϕ_i can be regarded as the absolute difference between w and w^o at the coordinate i, and the objective function seeks to minimize the sum of the ϕ_i. Notably, since $\mathcal{F}$ is a projection of a cell onto the first $d-1$ coordinates, a hyperplane of $\mathcal{H}$ is of dimension $d-1$. For an $\mathcal{F}$ that only partially overlaps V, the weight vector space constraints implied by V can also be easily incorporated into the linear program. A fair cell of lower dimensions can also be handled similarly.

Utility loss. Recall that for minimizing the utility loss, one only need to find the fair cell with the minimum utility loss. To do this, alongside the numbers of members for each protected group, we keep track of the utility of the top-k subset, which is a summation of scores of candidates under w^o within the top-k subset. Since minimizing the utility loss is equivalent to maximizing the utility of the top-k subset, the maximum utility of top-k subset satisfying the fairness constraints is maintained during the traversal.

Special attention is needed for handling top-k subsets in the presence of ties, as different tie-breaking outcomes may result in top-k subsets with different utility (recall Example 3). Since candidates of the same protected groups membership profile are exchangeable, a greedy strategy can be employed: once the backtracking algorithm identifies a fair tie-breaking outcome, for each protected groups membership profile, candidates with the highest scores are selected to achieve the maximum utility. To do so efficiently, for each protected groups membership profile, we sort the candidate by its score under w^o with a non-increasing order. In fact, only partial sorting is needed since at most $t \leq k$ candidates can be selected for each protected groups membership profile. Then, an array of prefix sums over these sorted scores is computed, which allows fast retrieval of the maximum utility when selecting any given number of candidates from that protected groups membership profile. The utility of selected candidates is maintained using the same method for maintaining the protected group membership counts during the backtracking search.

As an additional note, the process of obtaining a stable fair weight vector cannot be applied to a fair weight vector with the minimum w difference since the resulting stable fair weight vector may have a larger w difference (see Figure 2 for an illustration). In contrast, for a fair weight vector with the minimum utility loss, the stabilization does not change the utility loss.

F.3.1 Proof of Theorem 4

Proof. It is known that for $n_p = 1$ and $k = O(\text{polylog}(n))$, Fair Top-k Selection can be solved in $\tilde{O}(n)$ time for $d = 2, 3$, and in $\tilde{O}(n^{\lfloor d/2 \rfloor})$ (expected) time for a constant $d \geq 4$ (Cai, 2026a). Consider the additional overhead introduced by the augmentations. For the multiple protected groups, the additional time spent for maintaining additional variables is $O(n_p)$, which are constant-time operations as $n_p = O(1)$. For the backtracking subroutine, the additional overhead is $O(j + k^{2^{n_p}})$, where j is the number of intersecting hyperplanes at a cell. Since each additional intersecting hyperplane (making the configuration divert from a general position one) decreases the number of cells, the first part of the cost only adds $O(1)$ per cell in the worst case. Since $k = O(\text{polylog}(n))$ and $n_p = O(1)$, the second part of the cost only adds a polylog n factor of run time to each cell. For minimizing w difference and the utility loss, an additional linear program has to be solved. Since the number of variables is at most $2d$, this can be achieved in $O(|\mathcal{H}| + l)$ time using a linear-time algorithm for fixed-dimensional linear programming (Megiddo, 1984; Seidel, 1991). Since each cell can be subdivided into simplices (Henk et al., 2017), each with $O(1)$ boundary hyperplanes in fixed dimensions, and the number of cells remains bounded by the worst-case number after the subdivision, the first part of the cost only adds $O(1)$ per cell in the worst case. The additional cost of solving the linear program for each cell is then $O(\text{polylog}(n))$, as $l = O(\text{polylog}(n))$. Minimizing the utility loss also introduces additional overheads to the backtracking subroutine. However, the overhead per cell is only $O(\log k) = O(\log \log n)$ due to the partial sorting. □

G Omitted details of Section 5

G.1 Omitted details of Section 5.1

Recall in Section 4.1, the backtracking algorithm regards each distinct protected groups membership profile as a level in the search tree. However, it is possible that some protected group membership profiles may have no candidates associated with them, and thus can be ignored. Let $\beta \leq 2^{n_p}$ be the number of distinct protected groups membership profiles with at least one candidate. The backtracking search then becomes one of enumerating all possible value assignments to β non-negative integer variables whose sum is exactly t. By applying the "stars and bars" technique, one can show that the total number of such assignments is $\binom{t+\beta-1}{\beta-1}$, which is $O(t^{\beta-1})$ when $\beta = O(1)$. Notably, since adding "artificial" group labels corresponding to the intersections of multiple protected groups does not increase β, the algorithm incurs only a small overhead for handling intersectional fairness constraints.

To accelerate the backtracking, we also prune the search tree by considering the total number of remaining candidates during the search. At each level of the search tree, one can check the total number candidates available in all subsequent levels. The number of candidates selected at this level must be large enough such that the total number of selected candidates can reach t if all remaining candidates in subsequent levels are selected. This information, that the total number candidates in the remaining levels, can be easily retrieved from an array of prefix sum over the numbers of candidates of each protected groups membership profile.

Furthermore, notice that when $n_p = 1$, the fairness test can be done with a greedy approach, which computes the tight lower and upper bounds on the number of protected group candidates for all possible top-k subsets (Cai, 2026a). These bounds are used to test against the fairness constraint. This approach is also employed in our implementation to deal with the case of $n_p = 1$. When $n_p > 1$, this greedy approach is no longer sufficient to guarantee correctness. However, it can still be employed as a fast pruning step before invoking the backtracking search. For each protected group, we use the greedy approach to compute the corresponding lower and upper bounds on the number of candidates. If the interval defined by these bounds does not overlap the given fairness constraint for that group, one can safely conclude all top-k subsets violate the fairness constraints.

Finally, to minimize the utility loss, an approach that tracks the utility of the top-k subset during the backtracking search was proposed in Section 4.2 (see Appendix F.3 for details). In our implementation, we take a simpler approach that, whenever the search reaches a leaf corresponding to a fair top-k subset, its utility is computed from scratch using the prefix sum array of candidate scores for each protected groups membership profile, where the number of candidates of each member-

ship profile is stored in S' during the backtracking search (recall Algorithm 2). Even though this approach increases the runtime overhead from $O(n_p)$ to $O(\beta)$, where β denotes the number of distinct protected groups membership profiles with at least one candidate, the value of β is also small in practice, and utility computation is only executed at fair leaves. Moreover, this approach is numerically more robust, as dynamically updating the utility during the search is more susceptible to round-off errors than recomputing it from scratch. Notably, when $n_p = 1$, a greedy approach, which selects a candidate with the highest possible score while complying with the fairness constraint, is sufficient for the correctness. This method is implemented for handling the $n_p = 1$ case.

G.2 Omitted details of Section 5.2

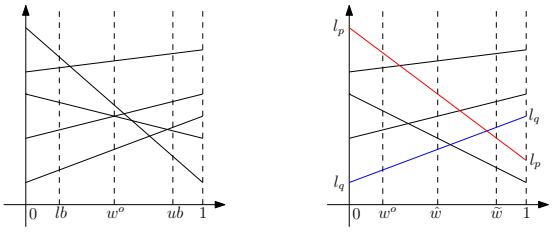
G.2.1 Omitted details of Section 5.2.1

Recall that for $n_p = 1$ without an optimization objective, the 2-D k-level-based algorithm (Cai, 2026a) is an adaptation of an algorithm in (Chan, 1999) (briefly outlined in (Basch et al., 1996)). The algorithm is a sweep-line algorithm using two kinetic tournament trees (Basch et al., 1996), S_1 and S_2, to keep track of the k-level cell. Viewing the x-coordinate of the sweep-line as "time", S_1 (resp. S_2) contains the k (resp. $n - k$) lines lying above (resp. below) at the current time instant. The algorithm proceeds by processing internal queue events or updating queues during the sweep. During the sweep, the number of protected group member is maintained for the fairness test. And the tie-breaking is resolved with the following techniques: (1) Symbolic perturbation (Edelsbrunner & Mücke, 1990) for event processing of kinetic tournament trees. (2) Carefully identifying pairs of intersecting lines to exchange between S_1 and S_2. (3) Guided tree traversal to collect all intersecting lines. (4) Greedy approach to compute tight lower and upper bounds on the number of protected group candidates for all possible top-k subsets.

Multiple protected groups. Just as was done in the theoretical k-level-based algorithm, to accommodate multiple protected groups, we keep track of the member counts for each protected group within S_1 during the sweep. For the tie-breaking with $n_p > 1$, after collecting the intersecting lines found by the guided tree traversal of S_1 and S_2, the dictionary N, which maintains the member counts for each protected group during the backtrack search, is initialized using the protected group member counts within S_1 and the intersecting lines from S_1. The array S can also be easily initialized by the set of intersecting lines (from S_1 and S_2). These steps together enable an invocation of the backtracking subroutine.

***w* difference.** Since $d = 2$ and $||w||_1 = 1$, for a given reference weight vector $w^o = (w_x^o, 1 - w_x^o)$ and any weight vector $w = (w_x, 1 - w_x)$, the w difference can be written as $g^o(w) = ||w - w^o||_1 = 2|w_x - w_x^o|$. Consequently, finding a fair weight vector minimizing the w difference is equivalent to finding a fair weight vector minimizing the absolute difference in the first coordinate. In the context of the sweep-line algorithm, regarding w^o as a vertical line $x = w_x^o$, the goal becomes finding the closest vertical line that yields a top-k subset satisfying the fairness constraints. Based on this observation, one can employ a bidirectional sweep-line algorithm to find the fair weight vector minimizing the w difference. Starting from a weight vector that is to the right (resp. left) of w^o and is in V (including w^o itself), the sweep-line moves from left to right (resp. from right to left) and stops at the first position that yields a fair top-k subset. The fair weight vector that minimizes the w difference is obtained from the two resulting fair weight vectors produced by the bidirectional line-sweeping (see Figure 5a).

This approach is efficient as it only (implicitly) constructs the part of $(k-1)$-level that is relevant to the search. However, it requires a kinetic tournament tree to be constructed twice, especially given that different sweeping directions affect the outcomes of symbolic perturbations. In the experiments for $n_p = 1$ (Cai, 2026a) (as well as those in Section 6.3), it was observed that the time spent on the line-sweeping process only accounted for only a small fraction of the total run time, making the construction of a kinetic tournament tree relatively expensive in practice. In our implementation, we take a simpler and more efficient approach which sweeps along one direction, starting from the left end-point defined by V while tracking the minimum w difference among the fair weights vectors. The previous observation is used as an early termination criterion, that if the sweep-line passes w^o and encounters a fair weight vector, the algorithm terminates immediately.



(a) (b)

Figure 5: **(a)** w difference minimization in 2-D, where the interval $[lb, ub]$ represents V and the vertical dashed line w^o represents the reference weight vector. The bidirectional sweep-line algorithm works by sweeping from w^o to ub and to lb. **(b)** For the weight vectors $\hat{w}$ and $\widetilde{w}$ with $k = 3$, l_p (resp. l_q) is in the top-k subset of $\hat{w}$ (resp. $\widetilde{w}$) but not in that of $\widetilde{w}$ (resp. $\hat{w}$). Intuitively, l_p lies above l_q at $x = w^o_x$ since the two lines intersect at a point between $x = \hat{w}_x$ and $x = \widetilde{w}_x$, where their orders change.

Utility loss. At first glance, it appears that for finding a fair weight minimizing the utility loss, the entire $(k-1)$-level within V must be swept. However, the bidirectional sweep-line algorithm can also be applied here, due to the following observation.

Lemma 1. *In 2-D, consider a given a reference weight vector $w^o = (w^o_x, 1 - w^o_x)$ and two weight vectors $\hat{w} = (\hat{w}_x, 1 - \hat{w}_x)$ and $\widetilde{w} = (\widetilde{w}_x, 1 - \widetilde{w}_x)$, where $w^o_x < \hat{w}_x < \widetilde{w}_x$ or $w^o_x > \hat{w}_x > \widetilde{w}_x$. The utilities of any pair of their corresponding top-k subsets, $\tau^{\hat{w}}_k$ and $\tau^{\widetilde{w}}_k$, must satisfy $U^o(\tau^{\hat{w}}_k) \geq U^o(\tau^{\widetilde{w}}_k)$.*

Proof. The proof is given only for the $w^o_x < \hat{w}_x < \widetilde{w}_x$ case, as the $w^o_x > \hat{w}_x > \widetilde{w}_x$ case is symmetric. Let $\tau^{\hat{w}}_k$ and $\tau^{\widetilde{w}}_k$ be an arbitrary top-k subset of $\hat{w}$ and $\widetilde{w}$, respectively. If $\tau^{\hat{w}}_k \neq \tau^{\widetilde{w}}_k$ (otherwise, $U^o(\tau^{\hat{w}}_k) \geq U^o(\tau^{\widetilde{w}}_k)$ trivially holds), one can find a pair of points such that each top-k subset contains a distinct point from the pair. More formally, we have a pair of points, p and q, such that $p \in \tau^{\hat{w}}_k$ and $q \in \tau^{\widetilde{w}}_k$ but $p \notin \tau^{\widetilde{w}}_k$ and $q \notin \tau^{\hat{w}}_k$. By the dual transformation, we have two lines $l_p : y = (p_x - p_y)x + p_y$ and $l_q : y = (q_x - q_y)x + q_y$, in which the value of the y-coordinate is a score under a weight vector. By $p \in \tau^{\hat{w}}_k$ and $q \notin \tau^{\hat{w}}_k$, we have $(p_x - p_y)\hat{w}_x + p_y \geq (q_x - q_y)\hat{w}_x + q_y$, since p is among top-k under $\hat{w}$ and q is not. Similarly, by $q \in \tau^{\widetilde{w}}_k$ and $p \notin \tau^{\widetilde{w}}_k$, we have $(p_x - p_y)\widetilde{w}_x + p_y \leq (q_x - q_y)\widetilde{w}_x + q_y$. With $\hat{w}_x < \widetilde{w}_x$, we have that $(p_x - p_y)\Delta x \geq (q_x - q_y)\Delta x$ for any $\Delta x \leq 0$, since the direction of the inequality changes as the value of x decreases from $\widetilde{w}_x$ to $\hat{w}_x$. Then, given $w^o_x < \hat{w}_x$, we have that $(p_x - p_y)(w^o_x - \hat{w}_x) \geq (q_x - q_y)(w^o_x - \hat{w}_x)$, and thus $(p_x - p_y)w^o_x + p_y \geq (q_x - q_y)w^o_x + q_y$. Because for any such pair of p and q, the score of p is at least as large as that of q under w^o, the utility of $\tau^{\hat{w}}_k$, which is the sum of all candidate scores under w^o, is at least as large as that of $\tau^{\widetilde{w}}_k$ (see Figure 5b). □

In other words, Lemma 1 tells us that if two weight vectors lie on the same side of w^o, the one that is closer to w^o along the x-coordinate has a utility loss that is lower than or equal to that of the other one, as the utility of its corresponding top-k subset is at least as large. Based on this observation, the bidirectional line sweeping process can also stop at the first position that yields a fair top-k subset, since further sweeping does not yield a top-k subset with a higher utility. Of course, for practical efficiency, we use a unidirectional sweeping approach by tracking the highest utility of top-k subsets during the sweep in our implementation. Lemma 1 can still be leveraged as an early termination criterion, analogous to the case of w difference minimization.

To obtain a stable fair weight vector, with the bidirectional sweep-line algorithm, one can continue the sweeping to identify the other end-point of the $(k-1)$-level cell (i.e. line segment), and then obtain the weight vector via the midpoint of the cell. Here, we introduce an alternative approach that, while potentially being less efficient, is easy to implement and more flexible. After obtaining the fair weight vector w^f that minimizes the utility loss, we run a variant of our algorithm for the Fair Top-k Verification problem (Algorithm 1) to obtain its corresponding top-k subset, τ^f_k, that minimizes the utility loss. Then, the kth candidate, c_k, is obtained from τ^f_k. (The tie-breaking involved here will be discussed later.) For any candidate $c \in \tau^f_k \setminus \{c_k\}$, its corresponding line in the dual space, l_c,

must lie above l_{c_k} associated with c_k. Consider the intersection point of l_c and l_{c_k}, if the intersection point falls within the interval defined by V. It partitions V into two intervals, and only one of them is valid as the corresponding downward-directed ray of the stable fair weight vector intersects l_c first. Similarly, for any candidate $c \in C \setminus \tau_k^f$, its corresponding line in the dual space must lie below the corresponding line of c_k, and a valid interval can be derived in the same manner. The intersection of these intervals define a weight vector subspace corresponding to the top-k subset τ_k^f, and we take the midpoint of the interval as the stable fair weight vector.

Apart from the time spent on backtracking search (which the sweep-line algorithm would also incur), this approach takes $O(n \cdot t)$ time, as there are t candidates that are possibly the kth one due to ties in scores (see Section 4.1 and Appendix F.1). In the worst case, t can be as large as k, though this rarely happens in practice. Moreover, the approach can be applied as a post-processing step to any algorithm that finds a fair weight vector minimizing the utility loss, making it both flexible and decoupled from the specific Fair Top-k Selection algorithm. Finally, it is easy to extend this method to higher dimensions, as will be shown shortly.

G.2.2 Omitted details of Section 5.2.2

Recall that for $n_p = 1$ without an optimization objective, the multi-dimensional k-level-based algorithm (Cai, 2026a) adapts the algorithm in (Andrzejak & Fukuda, 1999). Starting from a cell and its top-k subset, the algorithm explores adjacent cells by testing potential top-k subsets, formed by swapping one top-k element with an outsider. The test can be done by finding a separating hyperplane via a linear program. This enables enumeration of all valid top-k subsets in a breadth-first manner. Although the original algorithm (Andrzejak & Fukuda, 1999) is not as fast as the best theoretical algorithm in terms of the asymptotic run time, via a lockless implementation proposed in (Cai, 2026a), the adapted algorithm can efficiently utilize a multi-core shared-memory system.

Multiple protected groups. We further improve the performance of the practical multi-dimensional k-level-based algorithm by incorporating a technique from (Cai, 2026a), which was used to improve the baseline algorithm therein. When considering a swap between a current top-k element and an outsider candidate, we examine the intersection of their corresponding hyperplanes in the dual space. If this intersection lies outside the region V (after projecting the intersection onto the first $d-1$ coordinates), it implies that the element being swapped out always has a higher score than the element being swapped in for all weight vectors within V. In this case, the swapping can be skipped. Since l (the number of linear inequalities of V) is typically small, extreme points of V are precomputed to speed up intersection tests. This preprocessing incurs small overhead, especially when the dimensionality d is low.

***w* difference.** In our practical multi-dimensional k-level-based, the $(k-1)$-level cell is not explicitly determined, thus its set of boundary hyperplanes is not known. This issue can be addressed by utilizing the knowledge of the top-k subset and adding a cut-off variable, which gives us the following linear program:

$$\begin{aligned} \min \ & \sum_{i=1}^{d} \phi_i \\ \text{s.t. } & p(c) \cdot w \geq \lambda, && \forall c \in \tau_k^w \\ & p(c) \cdot w \leq \lambda, && \forall c \in C \setminus \tau_k^w \\ & \phi_i \geq w_i - w_i^o, && i = 1, \ldots, d \\ & \phi_i \geq w_i^o - w_i, && i = 1, \ldots, d \end{aligned} \tag{12}$$

Notice that in this linear program, the top-k subset, τ_k^w, is fixed and determined by the previous step. Our goal is to find a weight vector that minimizes the w difference within the cell. Thus, the top-k subset remains unchanged throughout the optimization process. There are $2d+1$ variables in this linear program. While the equation $\sum_{i=1}^{d} w_i = 1$ can be utilized to eliminate one dimension, the linear program still has $2d$ variables. Due to the factor of two in the number of variables, Seidel's LP algorithm (Seidel, 1991) that we used to solve linear programs for a small number of variables may quickly become inefficient as the dimensionality increases. The run time of Seidel's LP algorithm includes a multiplicative term that grows as the factorial of the number of variables, so the algorithm is efficient only when the number of variables is a small constant. Moreover, our experimental

results (Section 6.3 and Appendix H.2) suggested that the number of fair cells is typically small, thus these linear programs may only account for only a small fraction of the total run time. In our implementation, we use the simplex algorithm to solve linear programs for minimizing the w difference, as the algorithm is practically efficient and its performance is less sensitive to the number of variables.

Utility loss. To find a stable fair weight vector, we extend the approach used in the 2-D k-level-based algorithm. Again, for a fair weight vector w^f that minimizes the utility loss, its corresponding top-k subset τ_k^f is obtained, and the kth candidate, c_k, is obtained from τ_k^f. For any candidate $c \in \tau_k^f \setminus \{c_k\}$, the intersection of the hyperplane corresponding to c with that of c_k can be determined by $(p(c) - p(c_k)) \cdot w = 0$, as a hyperplane intersection corresponds to a tie in scores. Projecting this intersection onto the first $d-1$ coordinates partitions the weight space into two subspaces, and only one of them is valid as the downward-directed ray of the stable fair weight vector intersects the hyperplane associated with c first. For any candidate $c \in C \setminus \tau_k^f$, a valid subspace can be derived in the same manner. The set of bounding hyperplanes of these subspaces is actually a superset of $\mathcal{H}$, which is the set of boundary hyperplanes of the (protected) fair cell $\mathcal{F}$. Moreover, the intersection of these subspaces is $\mathcal{F}$ itself. Thus, the linear program (5) can also be applied here, by replacing $\mathcal{H}$ with the hyperplanes determined in the previous steps. Since the linear program contains $d+1$ variables (which can be reduced to d), Seidel's LP algorithm is employed. Solving the linear programs takes $O(t \cdot (l+n))$ time, with a linear time algorithm for linear programming in fixed dimensions being employed.

G.3 Omitted details of Section 5.3

Recall that for $n_p = 1$ without an optimization objective, a mixed-integer linear programming-based (MILP-based) algorithm (Cai, 2026a) is used for large k. The algorithm uses a binary indicator variable, δ_c, to encode whether a candidate c is within a top-k subset or not. Assume without loss of generality that each scoring attribute value is in the range of $[0, 1]$ (as discussed in Appendix D), this variable satisfies the following inequalities:

$$-1 \leq w \cdot p(c) - \lambda - \delta_c \leq 0, \tag{13}$$

where $\lambda \in [0, 1]$ is a cut-off value such that the pair (w, λ) can be regarded as a separating hyperplane. Then, the problem is formulated into a MILP to find a weight vector whose top-k subset satisfying the fairness constraint, using the indicator variables to enforce it. The resulting MILP is solved using a state-of-the-art MILP solver (Gurobi Optimization, LLC, 2024).

***w* difference.** In Section 4.2, minimizing the w difference is formulated as a linear program, making it easy to incorporate it into the mixed-integer linear program. By using the same formulation method, we have the following MILP:

$$\begin{aligned}
\min \ & \sum_{i=1}^{d} \phi_i \\
\text{s.t.} \ & \sum_{c \in C} \delta_c = k \\
& \sum_{c \in \mathcal{G}_j} \delta_c \geq L_k^{\mathcal{G}_j}, && j = 1, \ldots, n_p \\
& \sum_{c \in \mathcal{G}_j} \delta_c \leq U_k^{\mathcal{G}_j}, && j = 1, \ldots, n_p \\
& w \cdot p(c) - \lambda - \delta_c \leq 0, && \forall c \in C \\
& w \cdot p(c) - \lambda - \delta_c \geq -1, && \forall c \in C \\
& \phi_i \geq w_i - w_i^o, && i = 1, \ldots, d \\
& \phi_i \geq w_i^o - w_i, && i = 1, \ldots, d
\end{aligned} \tag{14}$$

with constraints on the weight vector w, the cut-off variable λ (i.e., $\lambda \in [0, 1]$), and indicator variables δ_c (i.e., $\delta_c \in \{0, 1\}$ for all $c \in C$) also being incorporated accordingly (omitted from the above formulation to simplify the presentation).

H OMITTED DETAILS AND ADDITIONAL EXPERIMENTAL RESULTS OF SECTION 6

H.1 EXPERIMENTAL SETUP

Experiments Design. Although the definition of the problem allows for more flexibility, we consider a more specific setting that finds a fair weight vector within the allowable deviation, as described in 6.1. We conducted three set of experiments: the first to compare runtime performance, the second to validate the algorithm's output under the optimization objectives, and the third to test the stability of the output fair weight vectors under small perturbations. Note that different methods were used to generate input weight vectors for these sets of experiments. For the runtime comparison, weight vectors were sampled uniformly at random until 20 *unfair* weight vectors were found, and the run time was averaged over the same set of 20 samples (excluding the time for sampling). Notably, Algorithm 1 (with engineering optimizations for the backtrack subroutine) was used to determine whether a sampled weight vector was fair. A limit of 20 hours was set on the overall run time of an algorithm, and the time for obtaining a stable fair weight vector (under the utility loss disparity objective) was omitted in our runtime comparison, as the same method was applied to all algorithms (including baselines). For the output validation, 50 weight vectors were sampled uniformly at random and used as inputs, regardless of whether they were fair or not. For the stability experiments, 50 unfair weight vectors were sampled and used as input.

Datasets. Two real-world datasets were used in our experiments. Details are given as follows:

- ***COMPAS*** (ProPublica, 2014): This dataset consists of 7,214 defendants from Broward County in Florida between 2013 and 2014. It is collected and published by ProPublica for their investigation of racial bias in the criminal risk assessment software. `juv_other_count`, `c_days_from_compas`, `priors_count`, `start`, `end` and `c_jail_out` − `c_jail_in` (the number of days staying in jail) were used as scoring attributes (6-D), and `juv_other_count` and `c_days_from_compas` were used for 2-D experiments. "African-American" (51.2% in the entire dataset), "Male" (80.7% in the entire dataset), and their intersections, "African-American Male" (42.2% in the entire dataset), were the protected groups.
- ***IIT-JEE*** (IIT, 2009): This dataset consists of scores of 384,977 students in the Mathematics, Physics, and Chemistry sections of IIT-JEE 2009 along with their genders and other personal info. All subject scores were used as scoring attributes (3-D), and the Physics and Chemistry scores were used for 2-D experiments. "Female" (25.5% in the entire dataset) and "reserved-category" (Baswana et al., 2019) (39.6% in the entire dataset) were the protected groups, with the latter consisting students from socially and economically backward or otherwise disadvantaged sections.

As in the experiments for $n_p = 1$ (Cai, 2026a), all values of scoring attributes were normalized to the range of $[0, 1]$. For our multi-dimensional experiments ($d \geq 3$), datasets were also preprocessed to shrink the input size using the k-skyband (Papadias et al., 2005; Sheng & Tao, 2012) and the reverse top-k query (Vlachou et al., 2010; Chen et al., 2023), as in the $n_p = 1$ experiments. Since the size of the IIT-JEE dataset was larger, we used larger values of k for it (ranging from 50 to 500), while smaller values of k were used for the COMPAS dataset (ranging from 10 to 100).

Synthetic data were deliberately excluded due to challenges in "meaningfully" reflecting real-world bias, as naive synthetic datasets (same distribution for all groups) diminish the need for fair top-k selection. See (Cai, 2026b) for a detailed discussion.

Baselines. For comparison, 2DRAYSWEEP and ATC$_+$ from (Asudeh et al., 2019) were implemented as baselines, with the former (2DRAYSWEEP) for 2-D experiments and the latter (ATC$_+$) for multi-dimensional experiments. The improvements that enhanced their performance for $n_p = 1$ were also applied (as in (Cai, 2026a)), along with additional refinements regarding the size of V (see Section 5.2.2 and Appendix G.2.2) incorporated into ATC$_+$. The tie-breaking was handled in the implementation of 2DRAYSWEEP, which used our method in Section 4.1 to accounts for tied candidates in the fairness test. To accelerate this test, candidates with identical scoring attribute values were pre-identified. However, the tie-breaking was ignored in the implementation of ATC$_+$, as it was not obvious how to do so without introducing significant runtime overhead for ATC$_+$.

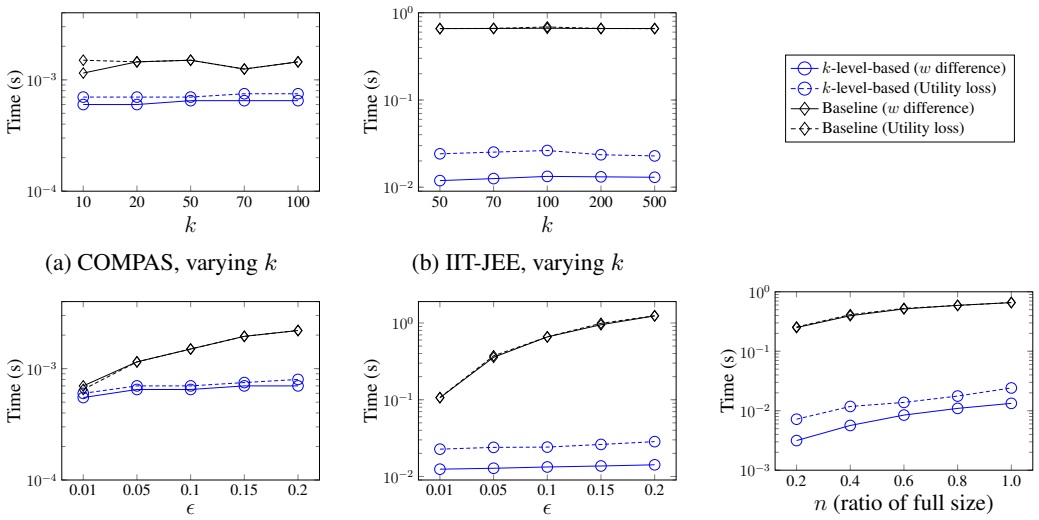
Environment and Implementation. All our implementations were in C++ on a 64-core AMD EPYC 7763 (2.45 GHz) machine with 512GB RAM, and our code was compiled and run inside an Apptainer (Kurtzer et al., 2017) container with a Rocky Linux 8.10 host OS.

Default values. For experiments in 2-D, $k = 50$ and $\epsilon = 0.1$. For experiments in higher dimensions ($d \geq 3$), $k = 50$ and $\epsilon = 0.05$. For the fairness constraints, the proportion of the protected group candidates in top-k selection results was constrained to $[40\%, 60\%]$, $[70\%, 90\%]$ and $[30\%, 55\%]$ for protected groups of COMPAS ("African-American", "Male" and "African-American Male", respectively), and was constrained to $[10\%, 40\%]$ and $[15\%, 65\%]$ for protected groups of IIT-JEE ("Female" and "reserved-category", respectively).

H.2 Complete runtime experiments

Since the main focus of this work is the speed of execution, we present a detailed analysis of all our runtime experimental results, starting with the 2-D experiments followed by results in higher dimensions. For better readability, the subfigures in Figure 3 are regrouped together with additional experimental results.

H.2.1 Runtime experiments for 2-D datasets



(a) COMPAS, varying k (b) IIT-JEE, varying k

(c) COMPAS, varying ϵ (d) IIT-JEE, varying ϵ (e) IIT-JEE, varying n

Figure 6: Runtime experimental results for 2-D datasets.

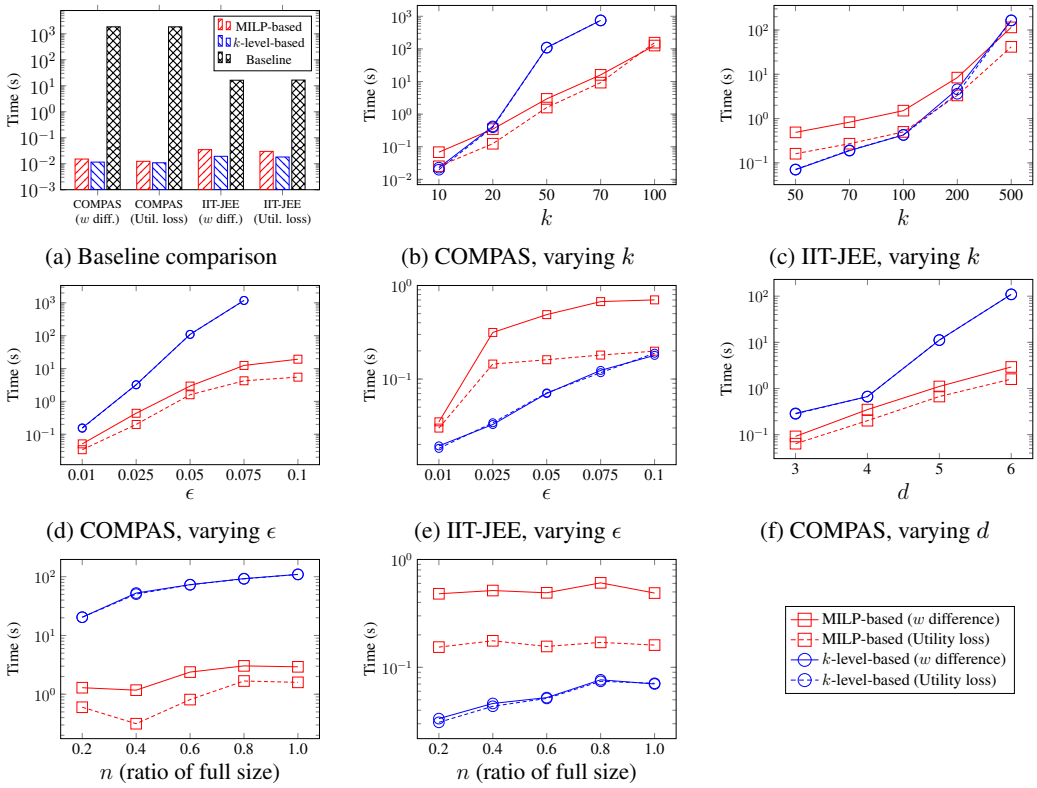
Varying k. Figures 6a and 6b show the average run times as k varies. The k-level-based algorithm performed consistently better than the baseline algorithm, achieving speedups of up to 50x and 28x for minimizing the w difference and the utility loss, respectively. The performance of the k-level-based algorithm remained stable as k increased since the actual k-level structural complexity was low. Consequently, the sweeping process accounted for only a small fraction of the total run time, as also noted in the analysis of $n_p = 1$ experiments (Cai, 2026a).

Varying ϵ. Figures 6c and 6d show the average run times as ϵ varies. The k-level-based algorithm still performed consistently better than the baseline algorithm. The performance trend with respect to ϵ was consistent with the observations from $n_p = 1$ experiments.

Varying n. Figure 6e shows average run times as n varies. We controlled n by randomly selecting 20%, 40%, 60%, 80%, and 100% of the IIT-JEE dataset. The k-level-based algorithm again consistently outperformed the baseline algorithm in all cases, with the performance trend being consistent with the observations from $n_p = 1$ experiments.

***w* difference vs. utility loss.** Across all experiments, utility loss minimization incurred greater overhead than w difference minimization in the k-level-based algorithm. This increase stemmed from additional steps required to compute the utility of a top-k subset in the backtracking algorithm and the cost of prerequisite computations (see Section 5.1 and Appendix G.1). Nevertheless, the k-level-based algorithm with utility loss minimization remained efficient compared to the baseline algorithm. While utility loss minimization introduced similar overhead to the baseline algorithm, it accounted for only a small fraction of the total run time, given the baseline algorithm's inherently long execution time. Therefore, the performance of the baseline algorithm was nearly identical under the two optimization objectives.

H.2.2 Runtime experiments for multi-dimensional datasets



(a) Baseline comparison (b) COMPAS, varying k (c) IIT-JEE, varying k

(d) COMPAS, varying ϵ (e) IIT-JEE, varying ϵ (f) COMPAS, varying d

(g) COMPAS, varying n (h) IIT-JEE, varying n

Figure 7: Runtime experimental results for multi-dimensional datasets ($3 \leq d \leq 6$).

Baseline comparison. Figure 7a shows a comparison of the two algorithms in the two-pronged solution against the baseline. For the COMPAS dataset, we set $k = 10$ and $\epsilon = 0.001$, while for the IIT-JEE dataset, we used $k = 50$ and $\epsilon = 0.01$. Despite the optimization applied to the k-level-based algorithm regarding the size of V (Section 5.2.2 and Appendix G.2.2) also improved the runtime efficiency of the baseline, particularly for the lower dimensional dataset (IIT-JEE), our two algorithms still achieved speedups of several orders of magnitude.

Varying *k*. Figures 7b and 7c show the average run times as k varies. The k-level-based algorithm outperformed the MILP-based algorithm when k was small, especially for the lower dimensional dataset (IIT-JEE). For the higher dimensional dataset (COMPAS), the MILP-based algorithm performed better. Despite the runtime improvement on the k-level-based algorithm due to the optimization regarding the size of V (Section 5.2.2 and Appendix G.2.2), it still failed to finish within the time limit when $k = 100$. The general performance trends of the algorithms were consistent with the observations from $n_p = 1$ experiments (Cai, 2026a).

Varying ϵ. Figures 7d and 7e show the average run times as ϵ varies. In both datasets, the run times of both algorithms increased as ϵ increased, which were consistent with the observations from $n_p = 1$ experiments.

Varying n. Figures 7g and 7h show the average run times as n varies. We controlled n by randomly selecting a subset of data before preprocessing. In general, the run times of both algorithms increased as n increased. For the k-level algorithm, its run time showed greater dependence on n compared to experimental results with $n_p = 1$, as it cannot terminate early given the optimization objective. Some anomalies still existed but were also consistent with the observations from $n_p = 1$ experiments.

Varying d. Figure 7f shows the average run times as d varies. We controlled d by selecting a subset of scoring attributes of the COMPAS dataset after preprocessing. The run times of both algorithm increased dramatically as the dimensionality d increased, with the k-level-algorithm increasing more rapidly, partly due to the linear programming algorithm (Seidel, 1991) used in our implementation. The general performance trends of the algorithms are also consistent with the observations from $n_p = 1$ experiments.

w difference vs. utility loss. For the k-level based algorithm, minimizing the w difference is expected to incur greater overhead than minimizing the utility loss, as the former requires solving a complex linear program while the latter only involves summing the scores of k elements. However, their empirical performance was nearly identical across all experiments. Since this additional overhead applies only to fair cells, this observation suggests that the number of fair cells was small such that the extra computational costs incurred by fair cells may be negligible compared to the time spent on $(k-1)$-level cell traversal. In contrast, for the MILP-based algorithm, minimizing the w difference does introduce noticeable overhead compared to minimizing the utility loss, with the former generally being slower than the latter.

H.3 Additional validation experiments

We further present validation experiments on the IIT-JEE dataset (Table 3), comparing our results with those reported in (Cai, 2026b), as well as experiments on the COMPAS dataset with $n_p > 1$ (Table 4). The results were consistent with what we observed in Section 6.2.

Table 3: Validation results for IIT-JEE dataset ($k = 100$).

ϵ	Fairness constraint	w difference			Utility loss		
		k-level	MILP	Optimal	k-level	MILP	Optimal
0.05	$[5\%, 45\%]$	0.055	0.089	**0.039**	0.033%	0.067%	**0.031%**
	$[7\%, 43\%]$	0.062	0.089	**0.055**	0.025%	0.043%	**0.019%**
	$[9\%, 41\%]$	0.089	0.092	**0.072**	0.028%	0.027%	**0.021%**
0.025	$[9\%, 41\%]$	0.050	0.050	**0.039**	0.022%	0.020%	**0.020%**
0.05		0.089	0.092	**0.072**	0.028%	0.027%	**0.021%**
0.075		0.102	0.134	**0.083**	0.036%	0.096%	**0.031%**

Table 4: Validation results for COMPAS dataset with default fairness constraints ($n_p = 3$) and $k = 50$.

ϵ	Found/Unfair Ratio[1]	w difference			Utility loss		
		k-level	MILP	Optimal	k-level	MILP	Optimal
0.025	2/45	0.068	0.133	**0.025**	0.075%	0.310%	**0.032%**
0.05	7/45	0.184	0.203	**0.113**	0.093%	1.227%	**0.080%**
0.075	16/45	0.273	0.300	**0.191**	2.635%	3.338%	**2.458%**

[1] The ratio between the number of input unfair weight vectors and found fair ones.