Finding a Fair Scoring Function for Top-k Selection: From Hardness to Practice

Guangya Cai
University of Minnesota, Twin Cities, MN, USA

Abstract

We study the problem of finding a fair linear scoring function over (numerical) attributes for top-k selection, ensuring fairness through a proportional representation constraint on the protected group. Existing algorithms do not scale efficiently, particularly in higher dimensions. Our hardness analysis shows that in more than two dimensions, no algorithm is likely to scale efficiently with respect to dataset size, and the computational complexity is likely to grow rapidly with dimensionality. However, the hardness results also provide key insights guiding algorithm design, leading to our two-pronged solution: (1) For small k, our analysis reveals a gap in the hardness barrier. By addressing various engineering challenges, including achieving efficient parallelism, we turn this potential of efficiency into an optimized geometry-based algorithm delivering substantial performance gains. (2) For large k, where the hardness is robust, we employ a practically efficient optimization-based algorithm which, despite being theoretically worse, achieves superior real-world performance. Experimental evaluations on real-world datasets then explore scenarios where worst-case behavior does not manifest, identifying areas critical to practical performance. Our solution achieves speedups of up to several orders of magnitude compared to the state of the art, an efficiency made possible through a tight integration of hardness analysis, algorithm design, practical engineering, and empirical evaluation.

2012 ACM Subject Classification Information systems → Top-k retrieval in databases

Keywords and phrases Fairness, Top-k, Integration

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.26

Related Version *Full Version*: `https://arxiv.org/abs/2503.11575` [11]

Supplementary Material *Software*: `https://github.com/caiguangya/fair-topk`
archived at `swh:1:dir:71bbb9faeffa0e61b7f5f063e08ec7e80d6ca6ec`

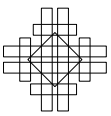
Acknowledgements The author would like to acknowledge the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing computing facilities.

1 Introduction

Selecting a subset of items from a large dataset based on specific criteria is critical for decision-making. Each item is typically assigned a score derived from its attributes, and the top-scoring items are selected. A scoring function determines these scores, typically a linear one that weights and sums the attribute values by importance. The problem of selecting the k highest scoring items from a dataset of n items ($k \leq n$) is called *Top-k Selection*.

Given the rise of automated decision-making software, ensuring fairness in tasks like top-k selection has become crucial. Among different ways of quantifying fairness (e.g., [21, 50, 52]), a common one is based on proportional representation: Each item has categorical attributes (e.g., gender, race, ethnicity), and one or a combination of them is considered *sensitive*. A specific value of this sensitive attribute denotes membership in a minority or historically disadvantaged group, often referred to as a *protected* group. The aim is to select a top-k subset where the protected group's proportion mirrors its proportion in the entire dataset.

Many applications seek top-k selections that are both fair and high-quality, often achieved by applying proportional fairness constraints. Prior work (e.g., [51, 13, 49]) typically applies these constraints after scores are determined, which might result in group-dependent selection



criteria and thus, potential legal risks (e.g., Ricci vs. DeStefano [42]). A fair scoring function from the outset is thus preferable. Moreover, one may also wish to find such a function within a subset of scoring functions. For instance, one may start with an existing, unfair scoring function and seek a "nearby" fair one, where attribute weights do not vary by a lot. Even knowing that no such nearby function exists can be informative, revealing some negligence in the original design. Consider the following variation of the example from [6]:

▶ **Example 1.** *A college admissions officer designs a scoring function based on applicants' normalized GPA (g) and SAT score (s), initially setting $f(c) = 0.5 \times g + 0.5 \times s$ and admitting the top-500 applicants. However, this may yield only 150 women whereas the fairness constraint is that at least 40% women should be admitted, possibly due to a gender disparity in the SAT score [44]. To meet the fairness constraint while preserving the explainability, the officer searches for a nearby fair scoring function with weights in the range of $[0.45, 0.55]$. An algorithm may find a fair function $f(c) = 0.55 \times g + 0.45 \times s$; if no such function exists, it reports failure, indicating the equal-weight design may be unsuitable. The officer may then adjust the design and, with the algorithm's help, obtain a fair function $f(c) = 0.6 \times g + 0.4 \times s$.*

Prior work has explored finding such fair scoring functions [6], but existing algorithms scale poorly with dataset size and worse with dimensionality. Since the general fairness constraints considered in [6] may introduce additional overhead, we focus on the simplest possible one on top-k selection, which ignores the relative rankings within the top-k (as in Example 1). For such a simple fairness constraint, do we have better algorithms that are efficient for large, high-dimensional datasets in practice, preferably running in $O(n \cdot \text{polylog}(n))$ time?

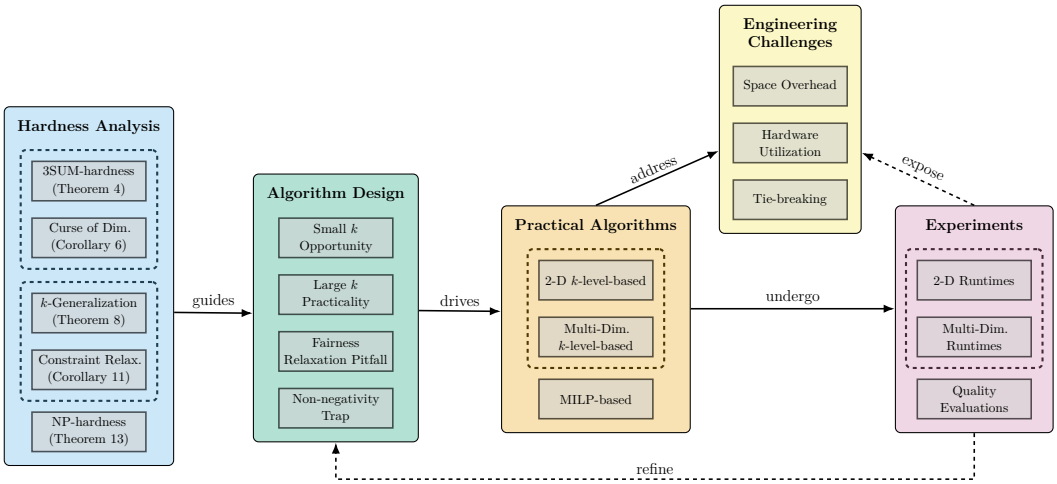


Figure 1 The structure and interplay of key results and components in this work. Solid arrows denote the primary workflow and direct influences, while dashed arrows indicate feedback.

Our contributions. To answer this question, we conduct an end-to-end, integrated study combining hardness analysis, algorithm design, practical engineering and empirical evaluation into a unified framework that yields an efficient solution. While some of the components may be of independent interest, their primary significance in this work lies in their interconnection. Our two-pronged solution, as well as many insights, emerge from the integration (see Figure 1).

Related work. Fair top-k selection under proportional fairness constraints has been widely studied [51, 31, 13, 6, 49] (see [52] for a review). Most existing work uses a single attribute or a fixed scoring function. Ref. [6] instead searched for a linear scoring function (i.e., a weight

vector) satisfying the fairness constraints while minimizing angular distance from a given one. Their method supports various fairness constraints but scales poorly with dataset size and dimensionality (see Section 4.2). Besides applying fairness constraints, ref. [35] introduced alpha-fairness – a metric capturing disparities between protected group proportions in the top-k subset and the entire dataset – and methods to find a weight vector optimizing this metric. However, their approach lacks a formal runtime analysis, still scales poorly, and breaks ties arbitrarily which may be unsuitable for fairness considerations (see Section 5.1).

2 Preliminaries

In the Fair Top-k Selection problem, we are given a set, $\mathcal{C}$, of candidates (or items), where each candidate $c \in \mathcal{C}$ has d scoring attributes and one sensitive attribute. The scoring attributes are represented by a point $p(c) = (p_1, p_2, \ldots, p_d) \in \mathbb{R}^d$, and the sensitive attribute by a label $A(c) \in \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. The label $\mathcal{G}_1$ denotes the protected group. Following [52], we also use $\mathcal{G}_i$ to denote the subset of candidates with $A(c) = \mathcal{G}_i$, particularly for $i = 1$.

We consider the class of linear scoring functions defined by a weight vector $w = (w_1, w_2, \ldots, w_d)$, where each $w_i \geq 0$ (a common assumption [46, 12, 6]). The score of a candidate c is computed as $f(p(c)) = w \cdot p(c) = \sum_{i=1}^{d} w_i p_i$. Since ties may occur, the top-k subset for a given w may not be unique. We consider all top-k subsets and regard w as fair if any of its top-k subsets satisfies the fairness constraint. This is embodied as follows:

▶ **Definition 2.** *For a given w and an integer $k \leq n$, a subset $\tau_k^w \subseteq C$ is a top-k subset if $|\tau_k^w| = k$ and for any pair of candidates that $c \in \tau_k^w$ and $c' \in C \setminus \tau_k^w$, $w \cdot p(c) \geq w \cdot p(c')$.*

We now formalize our notion of fairness. Let τ_k^w denotes a top-k subset for w. Let $L_k^{\mathcal{G}_1}$ (resp., $U_k^{\mathcal{G}_1}$) denote the desired lower (resp. upper) bound on the number of protected group candidates in τ_k^w, ideally near $k \cdot (|\mathcal{G}_1|/n)$. Then, w is a fair scoring weight vector if

$$L_k^{\mathcal{G}_1} \leq |\tau_k^w \cap \mathcal{G}_1| \leq U_k^{\mathcal{G}_1}. \tag{1}$$

Finally, one may wish to limit the search for a fair weight vector to a subset of the weight vector space. We denote this subset by V and describe it via a set of l linear inequalities. Without loss of generality, we further assume $||w||_1 = \sum_{i=1}^{d} w_i = 1$ (recall $w_i \geq 0$). Then, V can be effectively regarded as a subset of $\mathbb{R}^{d-1}$. Formally, we define our problem as follows:

▶ **Problem 3.** *We are given a set of n candidates with d scoring attributes and one sensitive attribute. Each candidate, c, has a point $p(c)$ in $\mathbb{R}^d$ and a label $A(c) \in \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. We are also given a non-negative integer $k \leq n$, non-negative integers $L_k^{\mathcal{G}_1}$ and $U_k^{\mathcal{G}_1}$, and a subset V of the weight vector space described by l linear inequalities. 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*

$$L_k^{\mathcal{G}_1} \leq |\tau_k^w \cap \mathcal{G}_1| \leq U_k^{\mathcal{G}_1}.$$

Although our formulation allows $p(c)$ to be in $\mathbb{R}^d$, in practice, scoring attribute values are typically normalized to $[0, 1]^d$, as they are equivalent via uniform translation and scaling.

3 Hardness

Known algorithms for the Fair Top-k Selection problem [6] do not scale well with the dataset size, especially in higher dimensions. We establish (conditional) lower bounds that shed light on this limitation. In particular, we focus here more on the **practical implications** of these hardness results (Remarks within the section), rather than on their proofs ([11, Section 3]).

3.1 Hardness results in fixed dimensions

We show that for any fixed dimension, $d \geq 3$, the Fair Top-k Selection problem is 3SUM-Hard [47, 22], hence is unlikely to be solvable in $O(n^{2-\delta})$ time for any constant $\delta > 0$. Furthermore, based on a result in [20], we show that the problem is unlikely to be solvable in $O(n^{d-1})$ time for any $d \geq 3$. All our conditional lower bounds are established for a special case of the problem, where k is approximately $n/2$, $\mathcal{G}_1$ contains only one candidate, and $L_k^{\mathcal{G}_1} = U_k^{\mathcal{G}_1}$.

For the 3SUM-hardness, it is known that deciding whether three points in a given set in $\mathbb{R}^2$ lie on a line (a.k.a point collinearity problem) is 3SUM-Hard [22]. By a reduction from the collinearity problem to the Fair Top-k Selection problem, we show the following:

▶ **Theorem 4** (3SUM-hardness). *For any $d \geq 3$, Fair Top-k Selection cannot be solved in $O(n^{2-\delta})$ time for any constant $\delta > 0$, assuming the 3-SUM hypothesis.*

▶ **Remark 5.** *For large datasets, practical efficiency typically requires algorithms with run times $O(n \cdot \text{polylog}(n))$. However, our reduction suggests that such an efficient algorithm is unlikely to exist for general problem instances in dimensions $d \geq 3$.*

The point collinearity problem is a special case of the Affine Degeneracy problem in $\mathbb{R}^d$: deciding whether $d+1$ points lie on a hyperplane. Using a similar approach we have:

▶ **Corollary 6** (Curse of Dimensionality). *For any constant $d \geq 3$, Fair Top-k Selection in $\mathbb{R}^d$ is at least as hard as the Affine Degeneracy problem in $\mathbb{R}^{d-1}$.*

▶ **Remark 7.** *In a certain decision tree model, all algorithms for the Affine Degeneracy problem require $\Omega(n^d)$ time [20]. This suggests a lower bound of $\Omega(n^{d-1})$ for Fair Top-k Selection in $\mathbb{R}^d$ for $d \geq 3$. However, the reduction only applies if d is a constant, since the number of constructed instances also grows exponentially with d (see Section 3.3).*

3.2 Hardness results for more general cases

The conditional lower bounds in fixed dimensions are established using a special case of the problem. Here, we show that our conditional lower bounds also hold when the value of k, the number of candidates of $\mathcal{G}_1$, and the fairness constraint are more general and realistic.

We begin with definitions of key concepts. For a point set $S \subseteq \mathbb{R}^d$, a point $q \in \mathbb{R}^d$ is a *dominated* (resp. *dominating*) point of S if for any point $p \in S$, $q_i < p_i$ (resp. $q_i > p_i$) for all $1 \leq i \leq d$. For any non-negative weight vector, a dominated (resp. dominating) point has a lower (resp. higher) score than any point in S. We say a candidate c is a *dominated* (resp. *dominating*) candidate of a candidate set C if $p(c)$ is a dominated (resp. dominating) point.

With these concepts, we first consider a general k value. Suppose that we have an efficient algorithm that can only handle $k = \beta n$, where $0 < \beta < 1$ is a fixed constant. We show that it can also be applied to solve the instance in Corollary 6 efficiently.

▶ **Theorem 8** (k-Generalization). *Given a polynomial-time algorithm for Fair Top-k Selection with $k = \beta n$, where $0 < \beta < 1$ is a fixed constant, one can solve the Fair Top-k Selection with $k = (n-1)/2 - d + 1$ in the same asymptotic run time.*

Proof. Since k is expected to be small in practice, we consider the case $\beta n < (n-1)/2 - d + 1$ first. We construct C' from C by adding dominated candidates, making the size of C' to be $(n-1)/2\beta - (d-1)/\beta$. Then, we run the algorithm for Fair Top-k Selection with $k = \beta n$ on C'. Given the size of C', we have $k = (n-1)/2 - d + 1$. Since all candidates in $C' \setminus C$

are dominated ones, none of them will be in the top-k subset for any $k \leq n$. So, the top-k selection result will only contain candidates in C. The reduction takes $O(n)$ time with β being a constant. The case $\beta n > (n-1)/2 - d + 1$ can also be similarly handled. ◀

▶ **Remark 9.** *This reduction extends the lower bounds in Section 3.1 to a broad range of k where $k = \beta n$. However, for a sufficiently small k (e.g., $k = O(\text{polylog}(n))$), the reduction breaks down as C' requires adding too many candidates, causing it to take more than $O(n)$ time. This reveals that the previous lower bounds may not be applicable for a small k.*

Next, we consider a larger number of candidates of $\mathcal{G}_1$ and more general fairness constraints. Consider a dataset with two groups, where n_1 and n_2 denote the number of candidates of $\mathcal{G}_1$ and the other group. By adding both dominated and dominating candidates, we have:

▶ **Theorem 10** (Group-balancing). *Given a polynomial-time algorithm for Fair Top-k Selection with $n_1 = n_2 = n/2$ and $L_k^{\mathcal{G}_1} = U_k^{\mathcal{G}_1} = k \cdot (n_1/n) = k/2$, one can solve the Fair Top-k Selection with $n_1 = 1$ and $L_k^{\mathcal{G}_1} = U_k^{\mathcal{G}_1} = 1$ in the same asymptotic run time.*

Theorem 10 still has a restriction on the fairness constraint that $L_k^{\mathcal{G}_1} = U_k^{\mathcal{G}_1}$. By carefully manipulating the number of dominated and dominating candidates, we are able to show:

▶ **Corollary 11** (Constraint Relaxation). *Given a polynomial-time algorithm for Fair Top-k Selection with $L_k^{\mathcal{G}_1} = k \cdot (n_1/n) - \sigma_L$ and $U_k^{\mathcal{G}_1} = k \cdot (n_1/n) + \sigma_U$, where σ_L and σ_U are non-negative integers, one can solve the Fair Top-k Selection with $n_1 = 1$ and $L_k^{\mathcal{G}_1} = U_k^{\mathcal{G}_1} = 1$ in the same asymptotic run time.*

▶ **Remark 12.** *Intuitively, relaxing the fairness constraint should make the problem computationally easier. However, our reduction suggests that this is unlikely to be the case, as better algorithms imply a violation of the lower bounds established in Section 3.1.*

3.3 Hardness result in arbitrary dimensions

Corollary 6 suggests a possible extension to arbitrary dimensions, since Affine Degeneracy is NP-hard in arbitrary dimensions [20]. However, a direct extension fails because the reduction requires generating 2^{d-1} Fair Top-k Selection instances to circumvent the non-negativity of the weight vector. To resolve this issue, we use an alternative problem, Maximum Independent Set, for the reduction, starting from its binary integer programming formulation [36].

▶ **Theorem 13** (NP-hardness). *Fair Top-k Selection is NP-hard in arbitrary dimensions.*

▶ **Remark 14.** *The failure to extend Corollary 6 under non-negativity may appear to reveal a gap in the hardness barrier. Moreover, this non-negativity appears to permit efficient mathematical optimization techniques (Section 5.2), which typically operate outside the decision-tree model (Corollary 6). However, our NP-hardness result shows that such methods may not work efficiently for the worst case. Notably, the relaxation technique, essential to many efficient solvers for our formulation (Section 5.2), also plays a key role in the reduction.*

4 From hardness results to algorithm design

As stated, our two-pronged solution is guided by the hardness results. Here, we examine these insights in details. Also, we examine the case of small k that is critical to our solution.

4.1 Implications for algorithm design

Our hardness results, starting from restricted cases, show that the problem becomes hard to solve for $d \geq 3$ (Remark 5) and demonstrate the curse of dimensionality (Remark 7). Extending these results to more general cases yields key insights for algorithm design:

- **Small k Opportunity.** The reduction technique used in Theorem 8 breaks down for a sufficiently small k (Remark 9). This suggests the potential of a more efficient algorithm by explicitly leveraging k in the design.
- **Large k Practicality.** The hardness of the problem is robust for a large k, making the existence of worst-case efficient algorithms unlikely. Therefore, designing efficient algorithms for large k should focus on real-world performance.
- **Fairness Relaxation Pitfall.** Relaxing the fairness constraint may appear to make the problem easier, making an algorithm that starts with a loose constraint and gradually tightens it seems attractive. However, such relaxation does not reduce worst-case computational hardness (Remark 12), limiting its value.
- **Non-negativity Trap.** With the failure to extend Corollary 6, non-negativity of variables (weights) and most coefficients (scoring attribute values) appears to open the door to efficient mathematical programming techniques (Remark 14). However, our NP-hardness result shows that this non-negativity may not yield runtime improvements.

Drawing on these insights, we arrive at a two-pronged solution: a theoretically grounded algorithm for a small k, and a practically efficient algorithm for a large k. Next, we demonstrate that the lower bounds are indeed breakable for a small k.

4.2 The case of small k

The Fair Top-k Selection problem can be solved by the algorithms in [6], which run in $O(n^2(k + \log n))$ (improvable to $O(n^2 \log n)$) for $d = 2$ and $O(n^{2d+2})$ for a constant $d \geq 3$. In contrast, our algorithm explicitly incorporates k, allowing it to run fast when k is small.

Like [6], our algorithm employs a dual transformation, but we adopt a different one given in [12, 48] that is better suited to our setting. Let $p = (p_1, p_2, \ldots, p_d)$ be $p(c)$ of a candidate $c \in C$. We transform p into a hyperplane $H(p)$ defined by ($x_1, x_2, \cdots x_d$ are variables)

$$(p_1 - p_d)x_1 + (p_2 - p_d)x_2 + \cdots + (p_{d-1} - p_d)x_{d-1} - x_d + p_d = 0. \tag{2}$$

Given the dual transformation, *k-level* and *cell* can be defined accordingly ([11, Section 4.2]). One important property of the 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 $k-1$ hyperplanes lying "above" in the x_d dimension.

We now proceed to the algorithm. Assume the $(k-1)$-level is constructed and cells adjacencies are identified, with hyperplanes in general position (i.e, no more than d hyperplanes intersect at a point), and ties in top-k selection broken arbitrarily. Our algorithm first finds an arbitrary weight vector $t \in V$ and locates a $(k-1)$-level cell intersected by the downward ray induced by t. Next, we obtain the corresponding top-k subset and count the number of candidates of $\mathcal{G}_1$ in the top-k subset, $n_k^{\mathcal{G}_1}$, and test it against $L_k^{\mathcal{G}_1}$ and $U_k^{\mathcal{G}_1}$. If the fairness constraint is satisfied, we output t. If not, we traverse the $(k-1)$-level by visiting adjacent cells. For each newly visited cell, we test if its projection onto the first $d-1$ coordinates intersects V. If it is, we update the $n_k^{\mathcal{G}_1}$ as the top-k subsets of adjacent cells differ in at most one candidate. We maintain the $n_k^{\mathcal{G}_1}$ during the traversal and whenever the top-k subset changes, test $n_k^{\mathcal{G}_1}$ against $L_k^{\mathcal{G}_1}$ and $U_k^{\mathcal{G}_1}$. The algorithm stops when either a fair weight vector is obtained from a $(k-1)$-level cell or all relevant $(k-1)$-level cells have been visited.

For the runtime analysis, leveraging known results of the k-level [37, 1, 24, 15], along with an application of an efficient convex polyhedra intersection algorithm [7], we have:

▶ **Theorem 15.** *In fixed dimensions, Fair Top-k Selection can be solved in $O(nk^{1/3}\log n)$ time for $d = 2$ and in $O(nk^{3/2}\log^4 n)$ time for $d = 3$, both with $l = O(n)$. In $d \geq 4$ dimensions, it can be solved in $O(n^{\lfloor d/2\rfloor}k^{\lceil d/2\rceil})$ (expected) time with a sufficiently small l.*

▶ **Remark 16.** *Notice that the algorithm relies on assumptions that do not fully align with the problem setting (Section 5.1). With careful refinements and thorough analysis, one can show that the bounds still hold. However, as the algorithm chiefly illustrates the key component for breaking the lower bounds, its full resolution is deferred to the practical context (Section 5.1).*

5 Practical algorithms

The k-level-based algorithm above builds on theoretically efficient algorithms for constructing the k-level, yet few have been empirically evaluated. Moreover, existing studies of a related algorithm show worse empirical performance than simpler heuristics [2, 25]. To make it practical, we identify key engineering challenges and present corresponding solutions. For large k, our algorithm design principle shifts from theoretical guarantees to practical efficiency. As a result, we present an algorithm based on mixed-integer linear programming (MILP).

5.1 Practical k-level-based algorithms

We identify three key engineering challenges for k-level-based algorithms to become practical:

- **Space Overhead.** The k-level-based algorithm constructs the entire k-level, whose structural complexity may grow quickly with n and d. For large datasets, it may exhaust internal memory, requiring much slower external-memory solutions. Yet users may seek a fair weight vector within a small, explainable region, making a full k-level unnecessary.
- **Hardware Utilization.** A prior theoretically efficient algorithm [25] exhibits bad memory performance, which contributes to its slow run time in practice. Moreover, many algorithms are not naturally parallelizable and cannot exploit modern multi-core systems. Bridging the theory-practice gap for parallel algorithms is even more challenging [9].
- **Tie-breaking.** Most theoretical algorithms for constructing the k-level assume general position (limiting ties among candidates) and break ties arbitrarily for simplicity. However, for the fair top-k selection, tie-breaking can affect the number of protected group candidates selected, impacting fairness. In our problem setting, an algorithm must be designed to account for all possible tie-breaking outcomes when evaluating fairness.

Next, we present practical k-level-based algorithms, focusing on our methods addressing these challenges. (Tie-breaking will be revisited in Section 5.2 for the MILP-based algorithm.)

5.1.1 2-D k-level-based algorithm

Our algorithm in 2-D is an adaptation of an algorithm in [14] (briefly outlined in [8]). The algorithm is a sweep-line algorithm using two kinetic priority queues, 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 time instant. The algorithm proceeds by processing internal queue events or updating queues during the sweep.

Space overhead. We note that in this algorithm, S_1 actually maintains the top-k subset during the sweep. Instead of explicitly constructing the $(k-1)$-level, one can simply check whether the top-k subset induced by S_1 satisfies the fairness constraint. Even better, following the theoretical algorithm (Section 4.2), one can use a variable to keep track of the number of lines of $\mathcal{G}_1$ in S_1. The space usage reduces to $O(n)$ by avoiding the explicit construction.

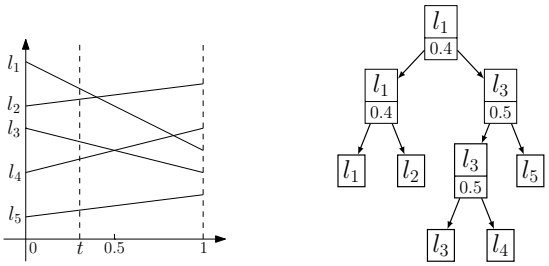


Figure 2 Maximum kinetic tournament tree (right) for lines (left) at $x = t$. We augment the tree with the replacement operation, which can be implemented by substituting the target line (i.e., l_1 in this example) at the leaf level and propagating updates along the path to the root.

Hardware utilization. Our low-level optimizations focus on the kinetic priority queue, a critical component for the real-world performance (see discussions in Section 6.2.1). In [14], fully dynamic kinetic priority queues are used, which may exhibit bad memory performance. However, observe that the sizes of S_1 and S_2 are fixed throughout the sweep; to correctly update them, one only has to replace the minimum (resp. maximum) element of S_1 (resp. S_2) with the maximum (resp. minimum) element of S_2 (resp. S_1) at the "time" instant of crossing. We use the kinetic tournament tree [8] for implementing the kinetic priority queue, in which this replacement can be done in $O(\log n)$ time (see Figure 2). Since the tree configuration is fixed, we further flatten trees into linear arrays for better cache performance, using a pre-order traversal layout. (Other layouts [29] might work better but are not explored.)

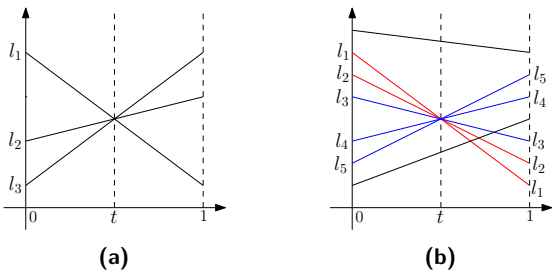


Figure 3 **(a)** Three lines intersect at $x = t$. The tie is broken by perturbing x to $x = t + \epsilon$. This, in fact, sorts the lines by their slopes, and orders l_3 ahead of l_1 and l_2 for a max-queue. **(b)** For $k = 3$, we have $l_1, l_2 \in S_1$ (red) and $l_3, l_4, l_5 \in S_2$ (blue) before the intersection. For correct configurations of S_1 and S_2 at $x = t + \epsilon$, one can exchange l_1 for l_5 and l_2 for l_4 between S_1 and S_2.

Tie-breaking. The algorithm in [14] assumes no more than two lines intersect at a point. To resolve this issue, we apply the symbolic perturbation [19], which provides a tie-breaking rule that compares the slopes of lines (see Figure 3a). Also, pairs of lines to exchange are carefully identified for correctly updating S_1 and S_2 (see Figure 3b). However, the top-k subset induced by the resulting S_1 is only one of the tie-breaking outcomes. To correctly

account for all tie-breaking outcomes, we collect all intersecting lines at this "time" instant (including duplicates) by a guided tree traversal. Each intersecting line can be found with an additional $O(\log n)$ cost. To avoid enumerating all possible top-k subsets, we compute tight lower and upper bounds on the number of $\mathcal{G}_1$ members and use these bounds to test fairness.

5.1.2 Multi-dimensional k-level-based algorithm

Our multi-dimensional algorithm adapts the method in [4], later implemented in [5] for rank-regret minimization. Although not as fast as the best theoretical algorithm, it may be better at exploiting modern multi-core systems as suggested in [4]. However, no parallel implementation currently exists. 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.

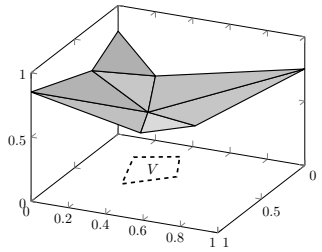


Figure 4 $(k-1)$-level (triangular mesh) and V (dashed region on the x-y plane) in 3-D. Testing cell intersection is equivalent to finding a point w in V whose downward-directed ray from $(w_1, w_2, +\infty)$ hits the cell (triangle). The constraints (points in V) can be incorporated into the linear program.

Space overhead. We incorporate weight vector space constraints (i.e., V) into each linear program, restricting it to find only weight vectors within V when testing a potential top-k subset (see Figure 4). This approach reduces the number of cells (i.e., top-k subsets) visited during the traversal, and thus space usage, especially when V only defines a small region.

Hardware utilization. Notice that the algorithm can be regarded as a breadth-first search algorithm on a graph, where each cell is a node and each edge connects a pair of adjacent cells. Therefore, a method for parallel BFS can be applied. However, since the graph is not known explicitly and finding an adjacent node can be expensive, efficient but sophisticated level-synchronous parallel algorithms (e.g. [34, 41]) may be difficult and less beneficial to adapt. Here we take a simpler approach by distributing jobs for finding adjacent nodes. Each thread independently takes a job for testing whether the given potential top-k subset is valid. If it is, it generates subsequent jobs. We use a global job queue for parallel job distribution. To reduce synchronization costs, a lock-free queue with relaxed FIFO ordering [30] (and a memory reclamation scheme [10]) is used, since a strict FIFO order is not needed. To keep track of the processed top-k subsets, we use a lock-free hash table [40]. Since the table size increases monotonically, we use an insertion-only table with memory reclamation disabled for further reducing runtime overhead. Notably, our lockless approach improves efficiency but introduces subtle implementation issues. For example, preventing premature thread termination requires carefully accounting for dequeue and emptiness-check linearization points [27] (specifically, under the restricted out-of-order specification [26]) of the job queue.

Tie-breaking. We modify linear programs to allow points lying on the hyperplane rather than enforcing strict separation. This approach enables the algorithm to enumerate all possible top-k subsets, as a top-k subset with ties (typically) corresponds to a lower-dimensional cell (see [11, Section 5.1.2]). In the extreme case, the run time will increase by a lot, but this rarely occurs in practice. Nevertheless, we still trade some efficiency for ease of implementation.

5.2 Mixed-integer linear programming-based algorithm

For a large k value, our focus becomes designing a practically efficient algorithm, which results in an algorithm based on MILP. We note that MILP is NP-hard and all known algorithms run in exponential times in the worst case. However, in practice, an MILP solver can be quite fast, and it was shown that an MILP-based algorithm is efficient for finding a linear scoring function for the reverse top-k query [16], making it an approach worth considering.

For simplicity, assume without loss of generality that each scoring attribute value is in the range of $[0, 1]$. We define a indicator variable $\delta_c \in \{0, 1\}$ for each candidate c satisfying

$$-1 \leq w \cdot p(c) - \lambda - \delta_c \leq 0 \tag{3}$$

for any given weight vector w and a cut-off value $\lambda \in [0, 1]$. δ_c encodes the top-k subset membership of c for a (w, λ) pair defining a separating hyperplane (see [11, Section 5.2]).

To ensure a value of λ will be infeasible unless it yields k candidates with $\delta_c = 1$, the constraint $\sum_c \delta_c = k$ is applied. Moreover, the fairness constraint can be encoded as follows:

$$L_k^{\mathcal{G}_1} \leq \sum_{A(c)=\mathcal{G}_1} \delta_c \leq U_k^{\mathcal{G}_1}. \tag{4}$$

Finally, constraints on the weight vector (i.e., V) can also be easily incorporated. We arrive at a MILP with a weight vector, a cut-off value, and n indicator variables as its unknown variables. Finding a fair weight vector becomes finding a feasible solution to the MILP.

Now, let us revisit the "Non-negativity Trap" (see Section 4.1). In our MILP formulation, all variables (w, λ and δ_c) and most of the coefficients ($p(c)$) are non-negative. The main issue is the negative coefficient of λ, which prevent direct adaptations of the techniques such as those in [3, 43]. While one might hope to circumvent this inconvenience, our NP-hardness result show that the problem may be fundamentally intractable in the worst case. This also suggests that the practical success of the MILP-based algorithm is largely heuristic.

For our implementation, we use Gurobi [23] – a state-of-the-art, mathematical programming tool – to solve the MILP. This solver also exploits parallelism to enhance performance.

Tie-breaking. The tie-breaking consideration has already been embodied in our MILP formulation which naturally addresses it. Notice that if $w \cdot p(c) = \lambda$, δ_c is free to choose its value. If the top-k subset is not unique for a weight vector, all possible top-k subsets will be considered in the MILP. The fairness constraint is applied to further restrict values of δ_c.

6 Experiments

We present experimental results on real-world datasets, including both run times and quality evaluations. We conclude with key observations and discussions of algorithm selection.

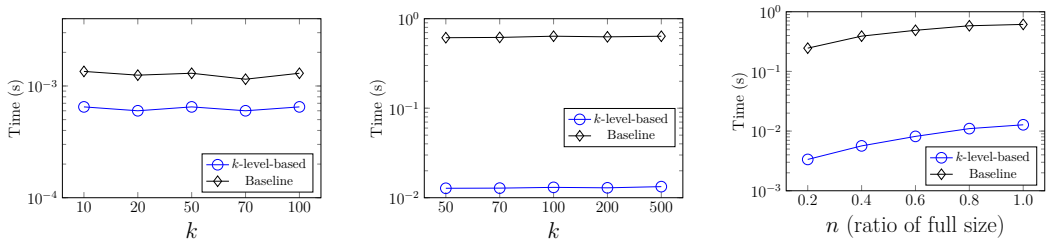
6.1 Experimental setup

In our experiments, a weight vector w^o is provided as an input. If w^o is unfair, the algorithm is run to find a "nearby" fair weight vector w^f. More formally, for a given unfair w^o, one aims to find a fair w^f such that $|w_i^f - w_i^o| \leq \epsilon$ for all $1 \leq i \leq d$, where ϵ is user-specified parameter that limits the allowable deviation. A failure is reported if no such fair w^f exists. We used two real-world datasets, *COMPAS* ($n = 7214$, $2 \leq d \leq 6$) [38] and *IIT-JEE* ($n = 384\,977$, $2 \leq d \leq 3$) [28], with preprocessing as needed. 2DRAYSWEEP and ATC$_+$ from [6] were implemented as baselines, with several enhancements incorporated. By default, $k = 50$, with $\epsilon = 0.1$ for $d = 2$ and $\epsilon = 0.05$ for $d \geq 3$. See [11, Section 6.1] for the full experimental setup and `https://github.com/caiguangya/fair-topk` for the code, data, and container [33].

6.2 Runtime experiments

Since our main focus is execution speed, we start with runtime experiments and highlight speedups from better hardware utilization that also enable more informed comparisons [32].

6.2.1 Runtime experiments for 2-D datasets



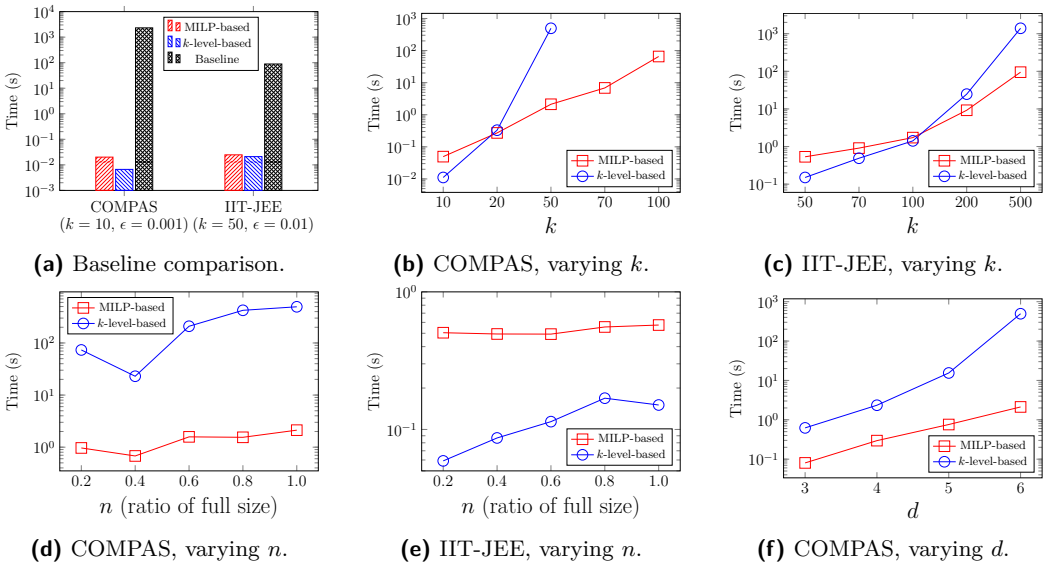
(a) COMPAS, varying k. **(b)** IIT-JEE, varying k. **(c)** IIT-JEE, varying n.

Figure 5 Runtime experimental results for 2-D datasets.

Varying k. As shown in Figures 5a and 5b, our k-level-based algorithm consistently outperformed the baseline, achieving over 40x speedups on the larger dataset (IIT-JEE). Notably, its performance remained stable as k increased. In fact, the line-sweeping step only contributed a small fraction of the total run time in our experiments. The algorithm [14] we adapted is output-sensitive, running efficiently when the actual k-level complexity is low. Moreover, it is known that the best known complexity of the k-level in 2-D, i.e., $O(nk^{1/3})$ [18], might not be a tight one [45], and the "average" complexity can be significantly lower [17].

Varying n. Figure 5c shows run times with n controlled by sampling 20%–100% of the IIT-JEE dataset. For the baseline, the run time increased mildly as n increased, performing better than its worst-case time complexity would suggest. However, our algorithm was still substantially faster. Comparing two algorithms side-by-side, both exchange orders of lines during the sweep, but the baseline considers all pairs, while our k-level-based approach uses kinetic tournament trees to skip many exchanges, directly reducing their number. Since many tree operations incur an $O(\log n)$ overhead, efficient implementation becomes critical. The experiments reveal an area where an efficient implementation should focus, and our optimizations (Section 5.1.1) addresses this need effectively (Section 6.2.3), enhancing the algorithm's performance in scenarios where the worst-case behavior does not manifest.

6.2.2 Runtime experiments for multi-dimensional datasets



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

(d) COMPAS, varying n. **(e)** IIT-JEE, varying n. **(f)** COMPAS, varying d.

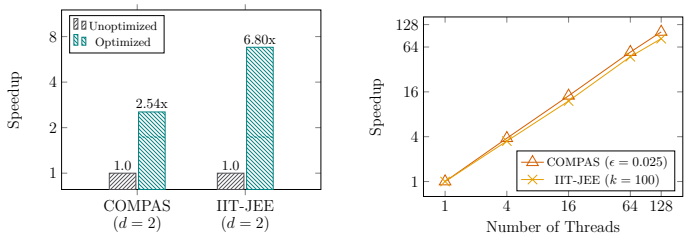
Figure 6 Runtime experimental results for multi-dimensional datasets ($3 \leq d \leq 6$).

Baseline comparison. In higher-dimensional experiments, the baseline regularly failed to finish within the time limit, so a smaller value of ϵ was used. As shown in Figure 6a, for the IIT-JEE dataset ($d = 3$), our MILP-based and k-level-based algorithms achieved 3598x and 4222x speedups, respectively. For the COMPAS dataset ($d = 6$), we had to further lower the value of ϵ, and our algorithms were five orders of magnitude faster. Due to the inefficiency of the baseline, subsequent experiments report only the run times of our two algorithms.

Varying k. As shown in Figures 6b and 6c, the MILP-based algorithm outperformed the k-level-based one for larger k, especially on the higher-dimensional dataset (COMPAS), while the k-level-based algorithm was faster for smaller k and lower dimensions (IIT-JEE). These results align with known bounds on the k-level complexity for $d \geq 3$. Unlike in 2-D, the multi-dimensional k-level algorithm's run time is more dependent on k, as each top-k subset will generate $O(nk)$ linear programs. Notably, the MILP-based algorithm also performed well on the lower-dimensional dataset (IIT-JEE) despite a larger number of (indicator) variables.

Varying n. As shown in Figures 6d and 6e, the run time of the k-level-based algorithm generally increased as n increased. However, some anomalies appeared since a larger n does not always yield more k-level cells, especially when only part of the k-level is explored. The MILP-based algorithm showed more stable run times despite increasing variable count.

Varying d. Figure 6f shows run times with d controlled by selecting subsets of COMPAS scoring attributes after preprocessing. The run times of both algorithms increased significantly as d increased. For the k-level-based algorithm, this aligns with the known results of k-level structural complexity. Besides, the $O(d!n)$ cost of the linear programming algorithm [39] we used also contributed to the run time increase. The MILP-based algorithm also showed significant slowdown, despite the number of variables only increasing by 1 as d increased by 1.



(a) Memory access. **(b)** Parallelism.

Figure 7 Performance gains from better hardware utilization, measured by total run time.

Notably, for $d = 3$, the k-level-based algorithm was slower than the MILP-based algorithm – unlike what we observed on the IIT-JEE dataset with the default $k = 50$ (Figure 6c). We note that this 3-D dataset created by selecting three attributes contains many candidates with identical scoring attribute values, which exposed the tie-breaking inefficiency in our implementation. However, since this lower-dimensional dataset is derived from a higher-dimensional one, it may contain fewer scoring attributes than intended to distinguish each candidate. Therefore, the slowdown is more likely an artifact rather than a real-world issue.

6.2.3 Hardware utilization

Memory access. Figure 7a shows the performance gains from optimizing the tournament tree of the 2-D k-level-based algorithm. We compared optimized and unoptimized versions, with the latter approximating a fully dynamic tournament tree while avoiding self-balancing by inserting new nodes into vacancies left by deletions. Our experimental results show notable speedups of 2.54x and 6.80x respectively, highlighting the benefits of our optimizations.

Parallelism. Figure 7b shows the efficiency of our parallel implementation of the multi-dimensional k-level-based algorithm. k and ϵ were default unless specified. Our implementation showed strong parallel scalability up to 128 threads, achieving speedups of 102x and 83x over the (stand-alone) sequential implementation on COMPAS and IIT-JEE, respectively.

6.3 Quality evaluation experiments

The quality was evaluated using the following metrics: (1) **Found/Unfair ratio**: the ratio between the number of input unfair weight vectors and found fair ones. (2) $\boldsymbol{w}$ **difference**: the average L_1 distance between w^o and w^f. (3) $\boldsymbol{\mathcal{G}_1}$ **proportion**: the average $\mathcal{G}_1$ candidate proportion in the top-k subset derived from w^f. (4) **Utility loss**: the average relative utility difference between the top-k subset derived from w^o and w^f. (See [11, Section 6.3].)

Table 1 shows the experimental results. We note that the k-level-based algorithm achieved better results for the w difference and utility loss. In fact, these two metrics are related, as a closer w^f to w^o typically yields a top-k subset with high-utility candidates. Our k-level-based algorithm naturally favors "nearby" weight vectors due to its breadth-first search approach with w^o as the starting point, while the MILP-based algorithm does not directly leverage w^o.

6.4 Observation summary and algorithm selection

We summarize key experimental observations and discuss algorithm selection based on them.

Table 1 Outcome quality evaluation for COMPAS dataset ($k = 50$).

ϵ	Fairness constraint	Found/Unfair ratio	w difference		$\mathcal{G}_1$ proportion		Utility loss	
			k-level	MILP	k-level	MILP	k-level	MILP
0.05	[34%, 66%]	19/39	**0.149**	0.211	66.0%	**65.7%**	**0.476%**	1.210%
	[40%, 60%]	8/44	**0.144**	0.213	59.8%	**59.5%**	**0.449%**	1.328%
	[44%, 56%]	6/47	**0.155**	0.202	**55.3%**	55.6%	**0.718%**	1.319%
0.025	[40%, 60%]	3/44	**0.088**	0.090	60.0%	**59.3%**	**0.121%**	0.142%
0.05		8/44	**0.144**	0.213	59.8%	**59.5%**	**0.449%**	1.328%
0.075		18/44	**0.247**	0.308	59.9%	**59.7%**	**2.416%**	3.187%

2-D. The k-level-based algorithm is the preferred method, with following key observations:

- Run times remained stable across k for both datasets, suggesting that k-level structural complexity may have limited impact on run time in real-world 2-D scenarios.
- Varying n experiments indicate that the k-level-based algorithm's advantage over the baseline may stem from the use of the kinetic priority queue that directly reduces the number of pairwise line exchanges, rather than the worst-case run time improvement.

In particular, building on the second observation, our implementation of the k-level-based algorithm was purposely engineered to further improve the performance (Figure 7a).

Multi-dimensions. By our algorithm design principle (Section 4), determining a threshold value of k is crucial. However, some previous discussions may not apply here, as our concern shifts to real-world performance. For real-world datasets, we have following key observations:

- The k-level-based algorithm outperformed the MILP-based algorithm for a small k, but its run time grew more rapidly with k, unlike its stable performance in 2-D.
- Run times of both algorithms increased dramatically with dimensionality d, with the k-level-algorithm increasing more rapidly, partly due to the LP algorithm used.
- The k-level-based algorithm performed better on the IIT-JEE dataset but worse on the 3-D COMPAS dataset with $k = 50$ (Figures 6c and 6f), likely due to an artifact exposing an inefficiency in our implementation for tie-breaking.

For algorithm selection, the first two observations suggest that as d increases, the threshold value of k at which the MILP-based algorithm becomes superior is likely to decrease. However, the last one suggests that performance comparisons between the two can be influenced by the underlying data distribution. Precisely defining a threshold value of k remains non-trivial, as the performance of the MILP-based algorithm is inherently heuristic. Nonetheless, our experimental results still indicate that this threshold is inversely correlated with d.

7 Conclusion

We presented an integrated study on finding a fair linear scoring function for top-k selection. Our hardness analysis established the theoretical foundation and guided the algorithm design, leading to our two-pronged solution: A k-level-based algorithm for small k and a MILP-based algorithm for large k. Practical engineering then addressed implementation challenges to ensure the solution works in practice. Finally, empirical evaluation explored real-world scenarios, identifying areas critical to practical performance. (See Figure 1 for the roadmap.) Our result is a practically efficient solution that is significantly faster than the state of the art, accompanied by many insights which we believe can inform future studies.

References

1. Pankaj K. Agarwal and Jiří Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995. doi:10.1007/BF01293483.
2. Yuval Aharoni, Dan Halperin, Iddo Hanniel, Sariel Har-Peled, and Chaim Linhart. On-line zone construction in arrangements of lines in the plane. In *Proceedings of the 3rd International Workshop on Algorithm Engineering (WAE)*, pages 139–153, 1999. doi:10.1007/3-540-48318-7_13.
3. Shabbir Ahmed and Weijun Xie. Relaxations and approximations of chance constraints under finite distributions. *Mathematical Programming*, 170:43–65, 2018. doi:10.1007/S10107-018-1295-Z.
4. 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)*, pages 1–12, 1999. doi:10.1007/3-540-48447-7_1.
5. Abolfazl Asudeh, Gautam Das, H. V. Jagadish, Shangqi Lu, Azade Nazi, Yufei Tao, Nan Zhang, and Jianwen Zhao. On finding rank regret representatives. *ACM Transactions on Database Systems*, 47(3):1–37, 2022. doi:10.1145/3531054.
6. 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)*, pages 1259–1276, 2019. doi:10.1145/3299869.3300079.
7. Luis Barba and Stefan Langerman. Optimal detection of intersections between convex polyhedra. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1641–1654, 2014.
8. 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)*, pages 302–319, 1996. doi:10.1007/3-540-61680-2_64.
9. Guy Blelloch, William Dally, Margaret Martonosi, Uzi Vishkin, and Katherine Yelick. SPAA'21 panel paper: Architecture-friendly algorithms versus algorithm-friendly architectures. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–7, 2021. doi:10.1145/3409964.3461780.
10. Trevor Alexander Brown. Reclaiming memory for lock-free data structures: There has to be a better way. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 261–270, 2015. doi:10.1145/2767386.2767436.
11. Guangya Cai. Finding a fair scoring function for top-k selection: From hardness to practice, 2026. arXiv:2503.11575.
12. Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong, and Wei Zhan. k-regret minimizing set: Efficient algorithms and hardness. In *20th International Conference on Database Theory (ICDT)*, pages 11:1–11:19, 2017. doi:10.4230/LIPIcs.ICDT.2017.11.
13. L Elisa Celis, Damian Straszak, and Nisheeth K Vishnoi. Ranking with fairness constraints. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 28:1–28:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.28.
14. 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.
15. Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete & Computational Geometry*, 64(4):1235–1252, 2020. doi:10.1007/S00454-020-00229-5.
16. 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. doi:10.14778/3598581.3598606.
17. Man-Kwun Chiu, Stefan Felsner, Manfred Scheucher, Patrick Schnider, Raphael Steiner, and Pavel Valtr. On the average complexity of the k-level. *Journal of Computational Geometry*, 11(1):493–506, 2020. doi:10.20382/JOCG.V11I1A19.

18 Tamal K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19:373–382, 1998. doi:10.1007/PL00009354.

19 Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990. doi:10.1145/77635.77639.

20 Jeff Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal on Computing*, 28(4):1198–1214, 1999. doi:10.1137/S0097539797315410.

21 Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. The (im)possibility of fairness: Different value systems require different mechanisms for fair decision making. *Communications of the ACM*, 64(4):136–143, 2021. doi:10.1145/3433949.

22 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

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

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

25 Sariel Har-Peled. Taking a walk in a planar arrangement. *SIAM Journal on Computing*, 30(4):1341–1367, 2000. doi:10.1137/S0097539799362627.

26 Thomas A. Henzinger, Christoph M. Kirsch, Hannes Payer, Ali Sezgin, and Ana Sokolova. Quantitative relaxation of concurrent data structures. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL)*, pages 317–328, 2013. doi:10.1145/2429069.2429109.

27 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

28 Indian Institute Of Technology. IIT-JEE, 2009. Retrieved from https://indiankanoon.org/doc/1955304.

29 Paul-Virak Khuong and Pat Morin. Array layouts for comparison-based searching. *ACM Journal of Experimental Algorithmics*, 22:1–39, 2017. doi:10.1145/3053370.

30 Christoph M. Kirsch, Michael Lippautz, and Hannes Payer. Fast and scalable, lock-free k-fifo queues. In *Proceedings of the 12th International Conference on Parallel Computing Technologies (PaCT)*, pages 208–223, 2013. doi:10.1007/978-3-642-39958-9_18.

31 Jon Kleinberg and Manish Raghavan. Selection problems in the presence of implicit bias. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 33:1–33:17, 2018. doi:10.4230/LIPIcs.ITCS.2018.33.

32 Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowledge and Information Systems*, 52(2):341–378, 2017. doi:10.1007/S10115-016-1004-2.

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

34 Charles E. Leiserson and Tao B. Schardl. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). In *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 303–314, 2010. doi:10.1145/1810479.1810534.

35 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)*, pages 2338–2350, 2024. doi:10.1109/ICDE60146.2024.00185.

36 László Lovász. On the shannon capacity of a graph. *IEEE Transactions on Information theory*, 25(1):1–7, 1979. doi:10.1109/TIT.1979.1055985.

37 Ketan Mulmuley. On levels in arrangements and voronoi diagrams. *Discrete & Computational Geometry*, 6:307–338, 1991. doi:10.1007/BF02574692.

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

39 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6:423–434, 1991. `doi:10.1007/BF02574699`.

40 Ori Shalev and Nir Shavit. Split-ordered lists: Lock-free extensible hash tables. *Journal of the ACM (JACM)*, 53(3):379–405, 2006. `doi:10.1145/1147954.1147958`.

41 Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, pages 135–146, 2013. `doi:10.1145/2442516.2442530`.

42 Supreme Court of the United States. 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.

43 Yotaro Takazawa, Shinji Mizuno, and Tomonari Kitahara. Approximation algorithms for the covering-type k-violation linear program. *Optimization Letters*, 13:1515–1521, 2019. `doi:10.1007/S11590-019-01425-W`.

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

45 Géza Tóth. Point sets with many k-sets. *Discrete & Computational Geometry*, 26(2):187–194, 2001. `doi:10.1007/S004540010022`.

46 Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Reverse top-k queries. In *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pages 365–376, 2010. `doi:10.1109/ICDE.2010.5447890`.

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

48 Xingxing Xiao and Jianzhong Li. rkHit: Representative query with uncertain preference. *Proceedings of the ACM on Management of Data (SIGMOD)*, 1(2):1–26, 2023. `doi:10.1145/3589271`.

49 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)*, pages 6035–6042, 2019. `doi:10.24963/IJCAI.2019/836`.

50 Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *Proceedings of the 29th international conference on scientific and statistical database management (SSDBM)*, pages 1–6, 2017. `doi:10.1145/3085504.3085526`.

51 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)*, pages 1569–1578, 2017. `doi:10.1145/3132847.3132938`.

52 Meike Zehlike, Ke Yang, and Julia Stoyanovich. Fairness in ranking, part I: Score-based ranking. *ACM Computing Surveys*, 55(6):1–36, 2022. `doi:10.1145/3533379`.