Fairness in Database Research Ranking & Selection

Fangzhu Shen, Yuxi Liu
03/23/2022

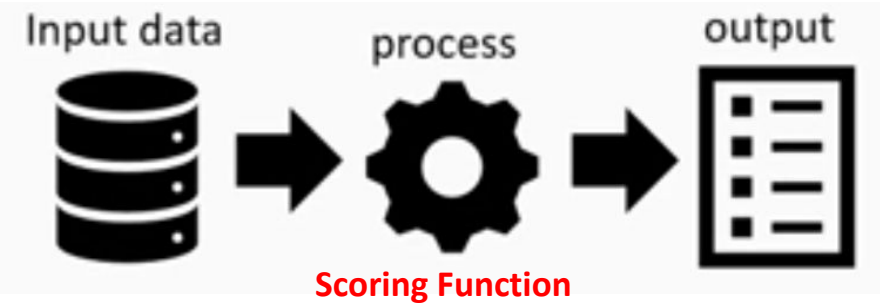
Designing Fair Ranking Schemes

Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, Gautam Das,
SIGMOD 2019

Ranking Problem

- Example: college admission, creditworthiness of individuals, tennis players ranking

- A prominent family of ranking schemes are **score-based rankers**: compute the score as a linear combination of attribute values, with weights



Ranking Example: College Admission

Goal: design a ranking scheme to evaluate a pool of applicants

Two scoring attributes: high school GPA (*g*) & SAT (*s*)

Fairness criterion: the admitted class should comprise at least 40% women

Ranking Example: College Admission

Goal: design a ranking scheme to evaluate a pool of applicants

Two scoring attributes: high school GPA (*g*) & SAT (*s*)

Fairness criterion: the admitted class should comprise at least 40% women

Priori: $f(t) = 0.5g + 0.5s$

Result: only 150 women among the top-500
(= 30% < 40%)

Possible Reason: women's average SAT is lower than men's

Ranking Example: College Admission

Goal: design a ranking scheme to evaluate a pool of applicants

Two scoring attributes: high school GPA (*g*) & SAT (*s*)

Fairness criterion: the admitted class should comprise at least 40% women

Priori: $f(t) = 0.5g + 0.5s$

Result: only 150 women among the top-500 (= 30% < 40%)

Possible Reason: women's average SAT is lower than men's

Goal →

Find an alternative scoring function both "fair" (given fairness criterion) & "high quality" (close to *f*):

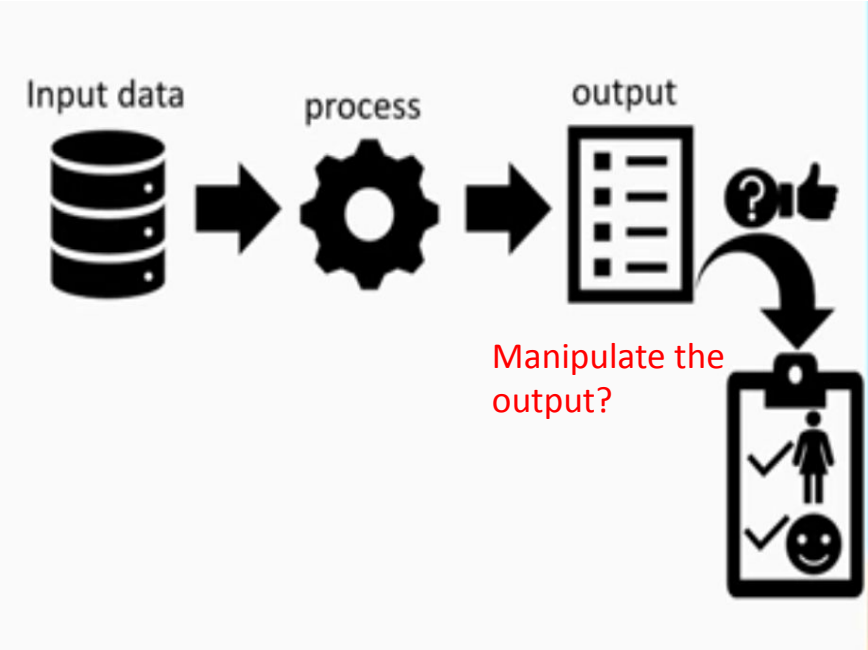
$$f'(t) = 0.45s + 0.55g$$

Fair Ranking Problem

Desire a ranking scheme:

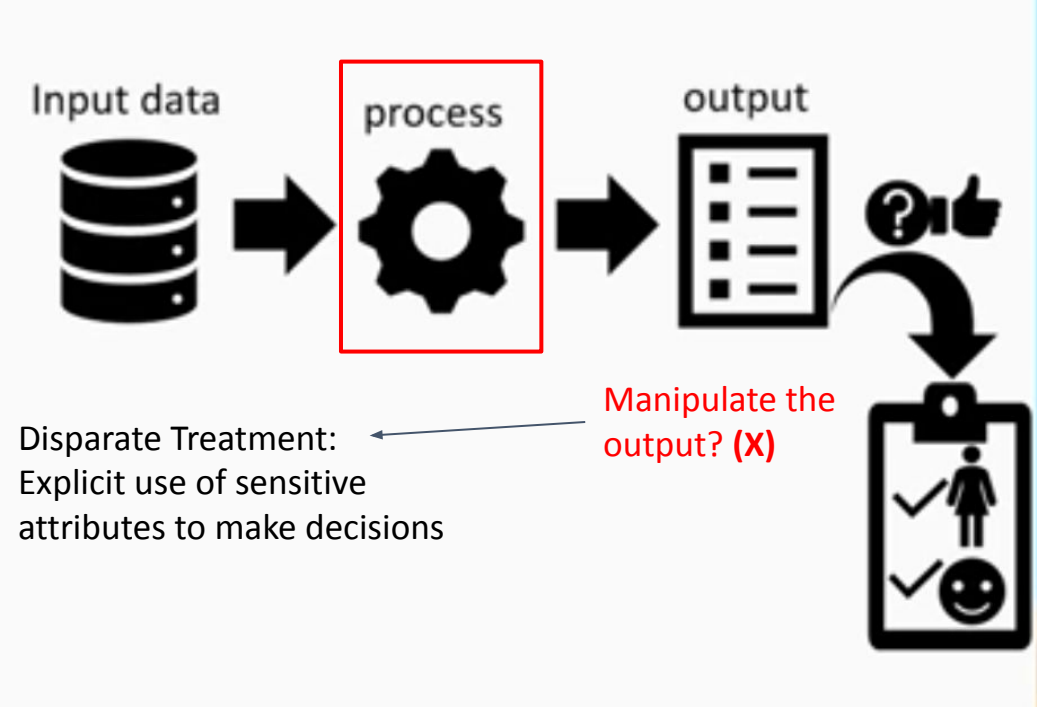
mitigate preexisting bias in **sensitive attributes**, e.g., disability, gender, race

Problem: how to make the output satisfy the fairness criterion?



Fair Ranking Problem

How to make the output satisfy some given fairness criterion?

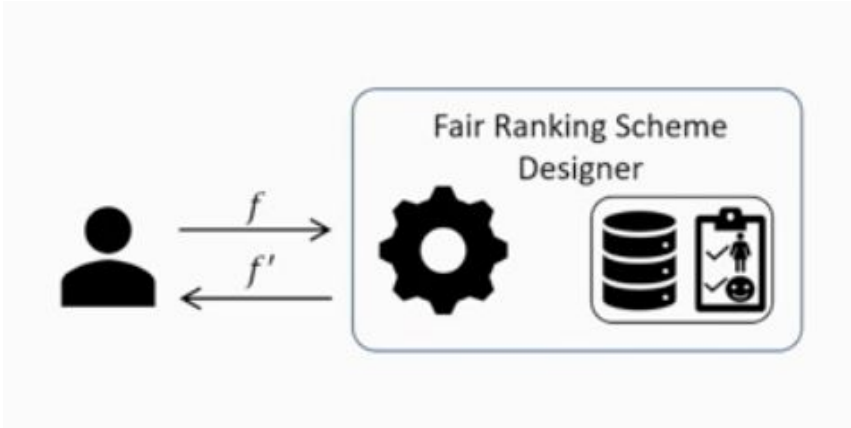


Problem

Goal: build a system that assists in designing fair score-based ranking schemes

Given the input data and fairness criterion, the system helps the designer:

1. check whether *f* satisfies the fairness criterion
2. identify an alternative function *f'* that satisfies the fairness criterion and similar to *f*



Data model

- Dataset $\mathcal{D}$ of n items, d scalar scoring attributes
- scoring attributes of item $t : < t[1], t[2], \dots, t[d] >$
- without loss of generality, assume that each scoring attributes is a non-negative number and that larger values are preferred

Ranking model

Linear Scoring function: $f(t) = \sum_{j=1}^{d} w_j t[j]$

Assumptions without loss of generality:

(1) non-negative weight

(2) higher score, higher rank

Fairness model

The system can adopt a general fairness model

We focus on **proportionality constraints**:

that bound the number of items belonging to a particular categorical attribute at the top-k

Example: sensitive attribute – gender

> “the admitted class should comprise at least 40% women”

Intuitive Geometric Interpretation

Items are represented by **points** in $\mathbb{R}^d$

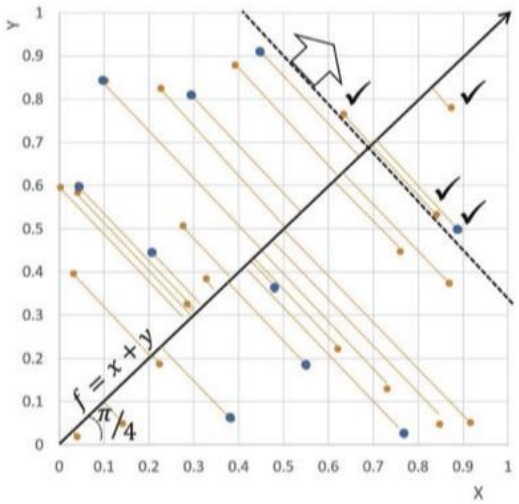
A linear scoring function *f* is represented by a **ray** starting from the origin and passing through the point

$$\overrightarrow{w} =< w_1, w_2, \cdots, w_d >$$

Every ray can be identified by angles.

The score-based ordering of the points induced by *f* corresponds to the ordering of their **projections** onto the ray.

Example with d=2:



(a) function $f = x + y$

Problem Definition

Closest Satisfactory Function: *Given a dataset $\mathcal{D}$ with n items over d scalar scoring attributes, a fairness oracle $O : \nabla_f(\mathcal{D}) \rightarrow \{\top, \bot\}$, and a linear scoring function f with the weight vector $\vec{w} = \langle w_1, w_2, \cdots, w_d \rangle$, find the function f' with the weight vector $\vec{w}'$ such that $O(\nabla_{f'}(\mathcal{D})) = \top$ and the angular distance between $\vec{w}$ and $\vec{w}'$ is minimized.*

Structure of the System

Offline phase:

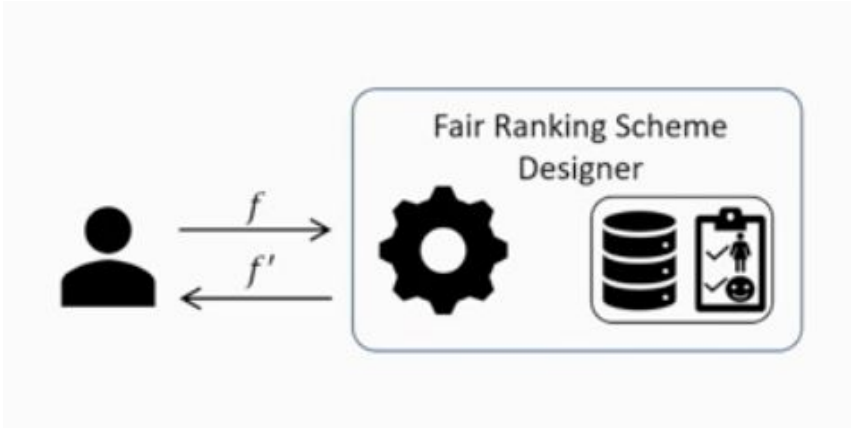
Preprocess the data and identify the set of functions that satisfy the fairness criterion

Online phase:

Answer user queries→Response fast!

Check whether *f* is satisfactory

Find an alternative *f'*



2-Dimensional Case

Important Notion: Ordering Exchange

The ordering of the items is the ordering of their projections on the ray of function f

- Right of green function: *t2>t1*
- Left of green function: *t1>t2*
- Partition the set of scoring functions

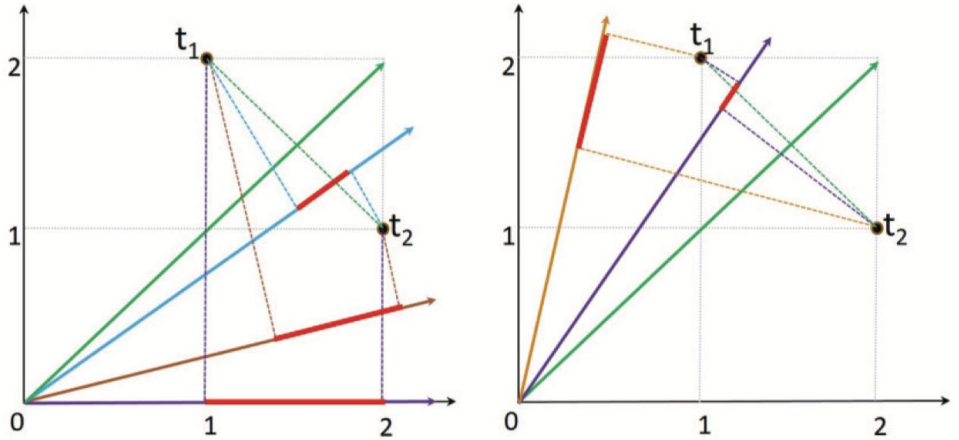


Figure 2: ordering exchange between a pair of points

Important Notion: Ordering Exchange

Idea: we do not need to consider every possible ranking function (every angle), we only need to consider at most as many as there are orderings of the items

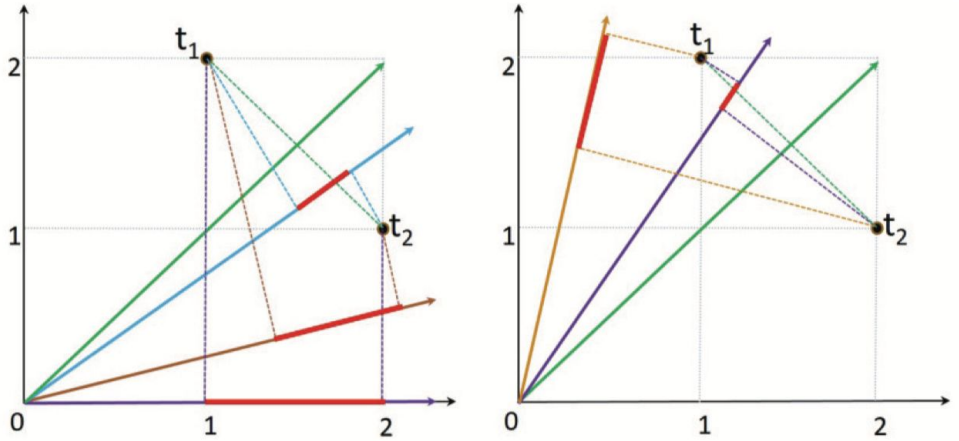


Figure 2: ordering exchange between a pair of points

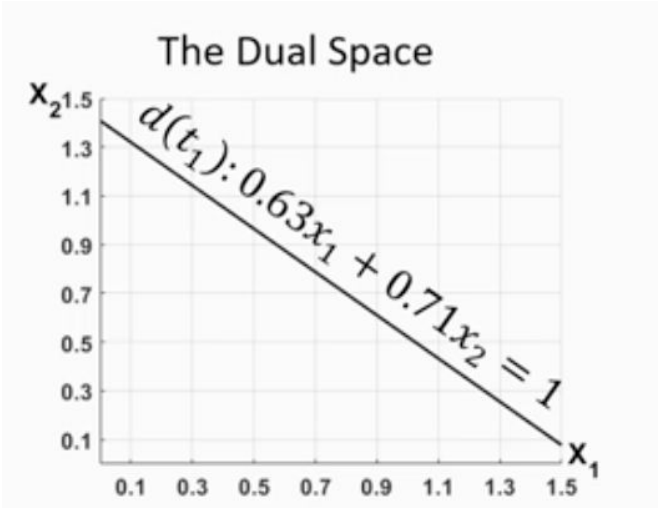
How to Identify Ordering Exchange: Dual Space

Transform items in dual space using the equation:

$$d(t) : \sum t[i]x_i = 1$$

2D: $d(t) : t[1]x + t[2]y = 1$

	$\mathcal{D}$		f
id	x_1	x_2	$x_1 + x_2$
t_1	0.63	0.71	1.34
t_2	0.72	0.65	1.37
t_3	0.58	0.78	1.36
t_4	0.7	0.68	1.38
t_5	0.53	0.82	1.35
t_6	0.61	0.79	1.4



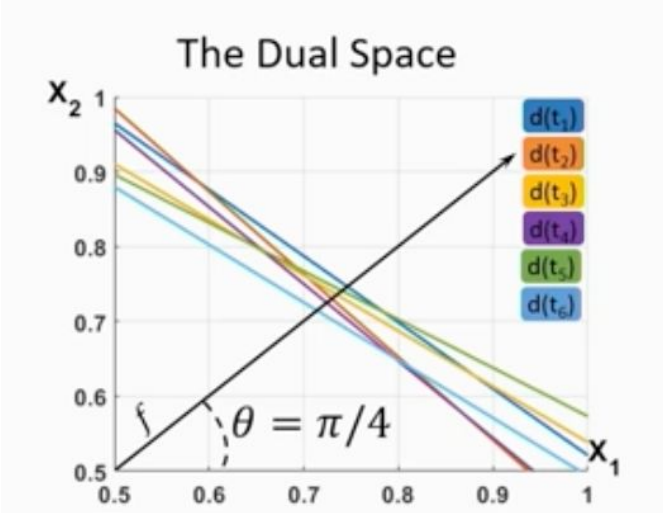
How to Identify Ordering Exchange:Dual Space

Every item *t* is transformed into the **line** *d(t)*

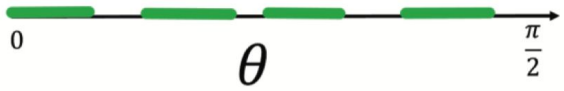
The ordering of items based on a function is identified by the **intersections** of the dual lines with the ray

→ intersection of two dual line can identify an ordering exchange

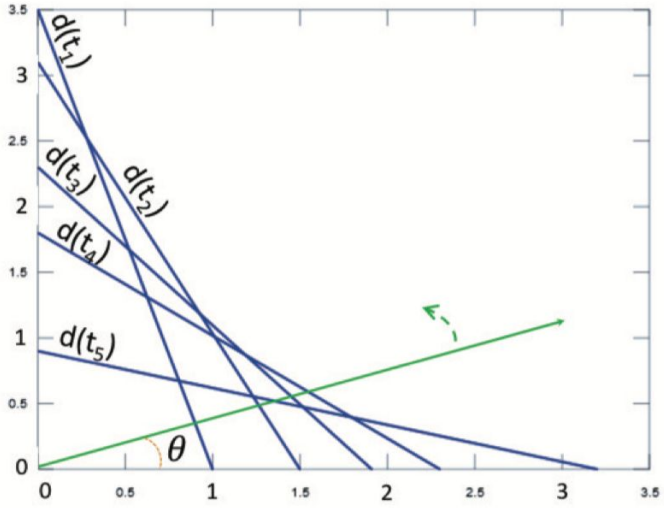
	$\mathcal{D}$		f
id	x_1	x_2	$x_1 + x_2$
t_1	0.63	0.71	1.34
t_2	0.72	0.65	1.37
t_3	0.58	0.78	1.36
t_4	0.7	0.68	1.38
t_5	0.53	0.82	1.35
t_6	0.61	0.79	1.4

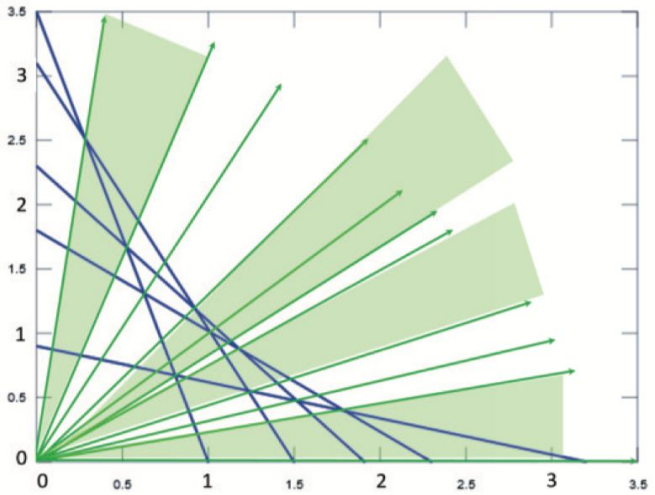


Offline Processing



- Identify all “satisfactory regions” by sweeping from x-axis to y-axis

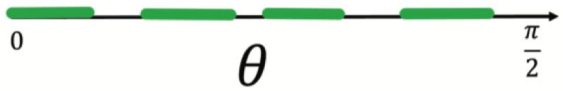




*Green region means all function within that angle satisfies the fairness criterion

Online Processing

- Note: every ray can be identified as an angle
- Check whether *f* is in satisfactory regions
- Apply binary search on the sorted list of satisfactory regions
- Fast!



Multi-Dimensional Case

Multi-Dimensional Case

- 2D Extension:
  - geometry of ordering exchange
- Offline processing:
  - arrangement tree construction
- Online processing:
  - approximation
- *Sampling approaches to improve the performance

MD: Ordering exchange from hyperplanes

D-dimension: $\overrightarrow{w} =< w_1, w_2, \cdots, w_d >$

Angle coordinate system: every function (ray in $\mathbb{R}^d$) can be represented by the point $< \theta_1, \theta_2, \cdots, \theta_{d-1} >$ (d-1 angles), each is in [0, π/2]

Consider dual space: item t is transformed into (d-1)-dimensional hyperplane

$$\mathrm{d}(t) : \sum_{k=1}^{d} t[k].x_k = 1$$

Intersection of two hyperplanes: (d-2) structure

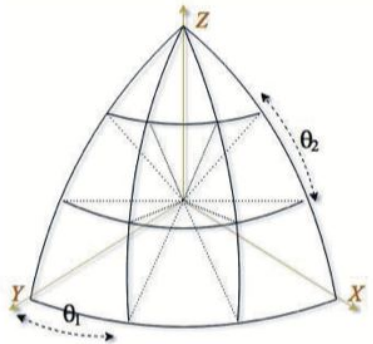
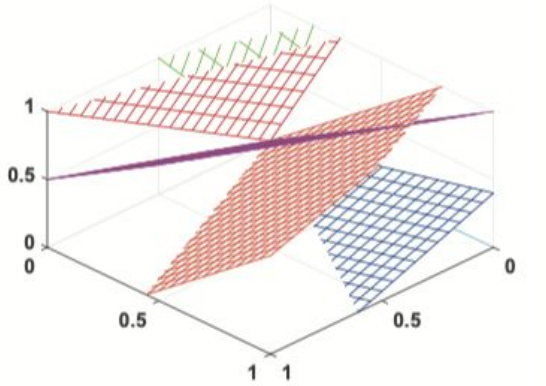


Figure 9: Angles in $\mathbb{R}^3$

Ordering exchanges from hyperplane

3-dimensional case:

id	A_1	A_2	A_3
t_1	1	2	3
t_2	2	4	1
t_3	5.3	1	6
t_4	3	7.2	2



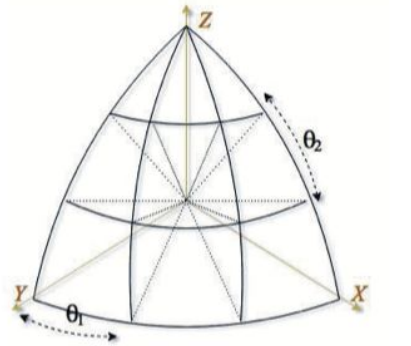


Figure 9: Angles in $\mathbb{R}^3$

How to identify satisfactory regions in MD?

*Satisfactory Region: functions in the region satisfy the given fairness constraint

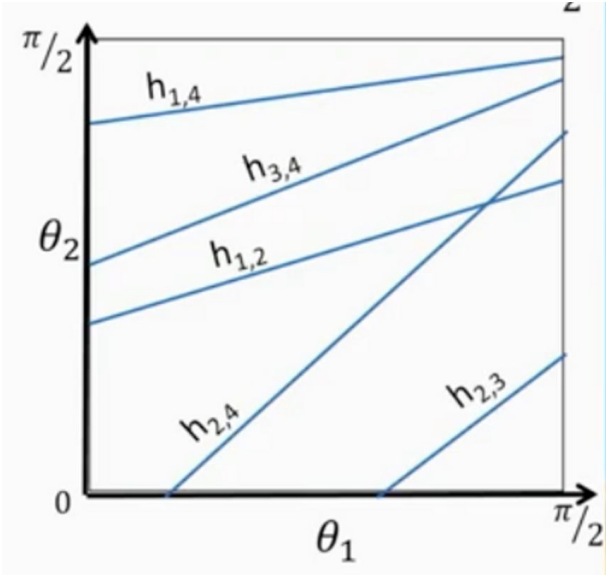
Satisfactory regions in MD

Check the intersection of every two item hyperplanes?

Ordering exchange hyperplane h(i,j): items i and j switch order on the two sides

Every hyperplane h(i,j) divides the space into two half spaces: h+,h-

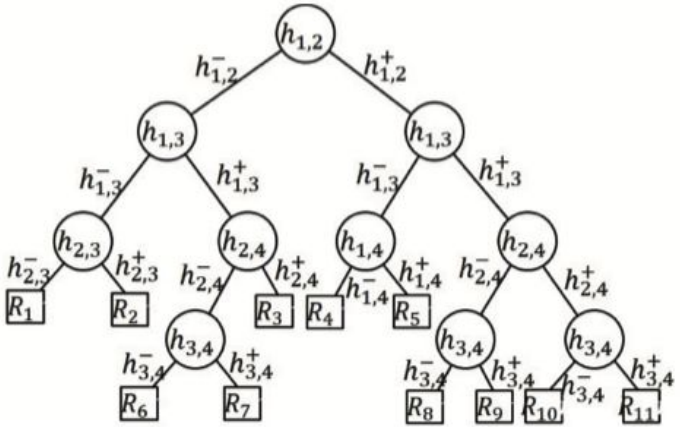
Inside each convex region, the order does not change

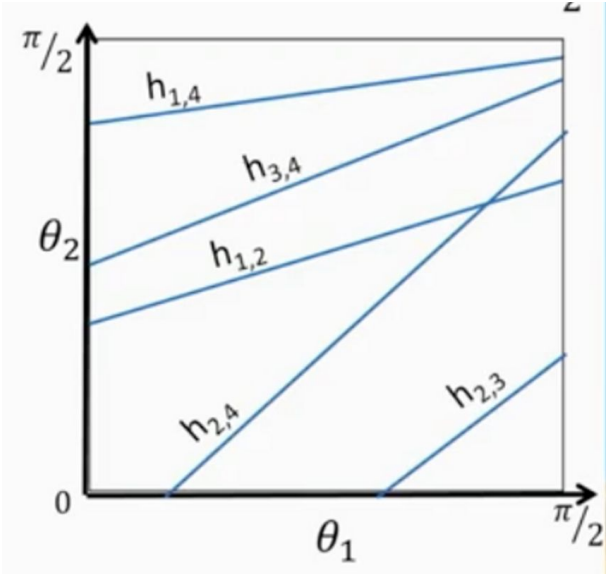


Satisfactory regions in MD

An incremental algorithm to identify all satisfactory convex regions fast: **Arrangement tree**

- add the hyperplanes one after the other, at each iterations, it will consider the set of regions which the new hyperplane intersects





Online Processing

How to find the closest satisfactory function f' to f?

Baseline: solve the problem for each satisfactory region, and return the function with minimum angle distance

Online Processing

How to find the closest satisfactory function f’ to f?

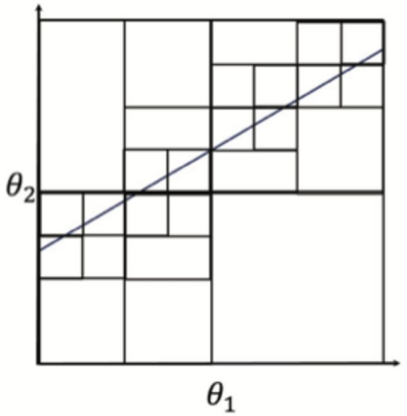
Baseline: solve the problem for each satisfactory region, and return the function with minimum angle distance

Efficient algorithm to obtain **approximate** answer quickly: partition into cells

MD Online Processing: Approximation

Idea: assign a fair function to each cell

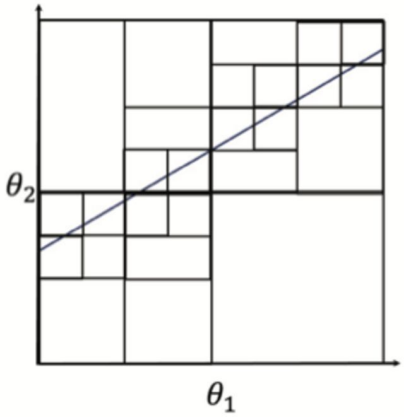
- Partition the function space into equi-volume cells
- Identify the set of all cells that intersect with at least one satisfactory region
- For each cell, identify ***HC***: the set of hyperplanes passing through the cell



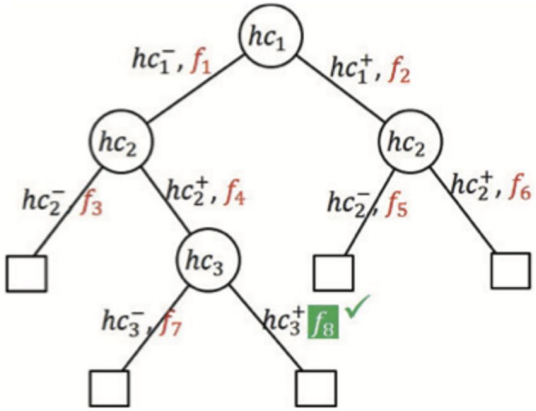
MD Online Processing: Approximation

Idea: assign a fair function to each cell

- For each cell, we try to find a satisfactory function inside it:
  - use the arrangement tree structure and apply a stop early strategy



- Consider other cells: assign the closest discovered satisfactory function



Experimental Evaluation

Experimental Setup

Dataset: COMPAS

A dataset about racial bias in criminal risk assessment

Fairness models

FM1: proportional representation on a **single** type attribute (default)

- "at most 60% of the top-ranked 30% are African American."

FM2: proportional representation on **multiple** type attributes

- "specify the maximum proportion of members of each particular demographic group (sex, race, age) among the top-ranked 30%"

Validation

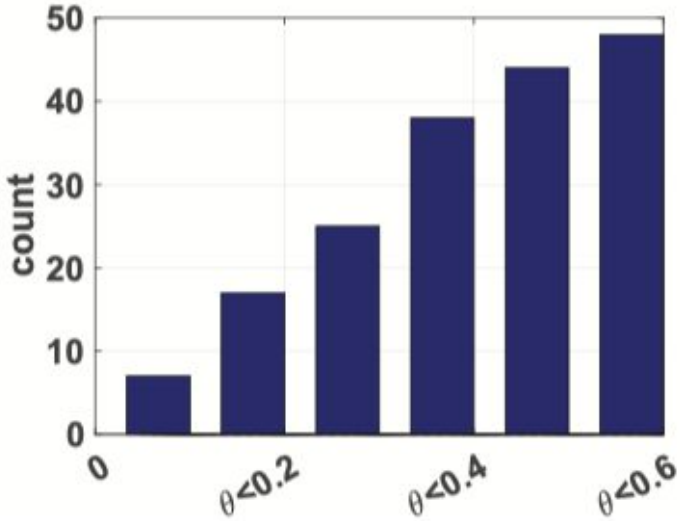
COMPAS with d = 3

Fairness model: FM1

Out of 100 random queries, 52 were satisfactory.

For the remaining 48 functions, the model can find a satisfactory function f' close to the input function f

Show the effectiveness



Angle between input/output function

Performance of Query Answering

The goal of the system: fast for online phase – answering user queries

2D

- Do binary search
- Query time *O(logn)*: 30 usec
- Time for ordering results based on the input function *O(nlogn)*: 25 msec

MD

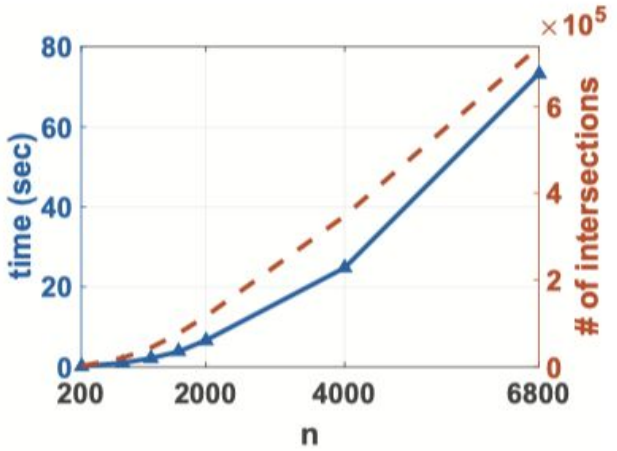
- Find the cell that *f* belongs in *O(logN)*, N is the number of cells
- Query time: 200 usec
- Time for ordering: 25 msec

Performance for Preprocessing

2D

- Evaluate the efficiency of sweep algorithm

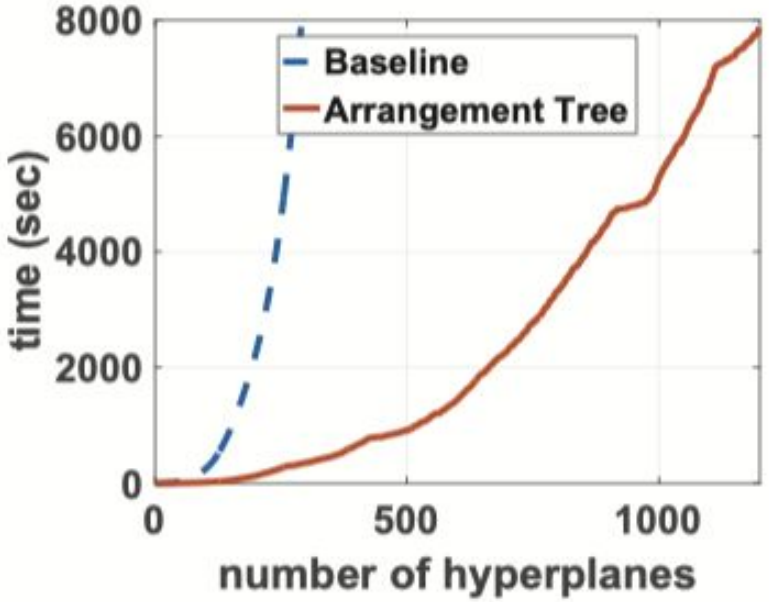
- The number of ordering exchanges is much smaller than the theoretical O(n^2) upper-bound
- Even for the large setting, the preprocessing is within the reasonable time



Performance for preprocessing

MD

Proposed the arrangement tree data structure, in order to skip comparing a new hyperplane with all current regions



Advantage of arrangement Tree

Conclusion

- Use the geometry knowledge, to design a system that given an input function f, checks whether it satisfies fairness criterion which user defines. If it does not satisfy, the system will identify an alternative function f'

- Focus on the linear scoring function
- For a general fairness model

Fairness-Aware Range Queries for Selecting Unbiased Data

Suraj Shetiya et al., ICDE 2022

Background

- In the era of big data and advanced computation models, we are all constantly being judged by the analysis, algorithmic outcomes, and AI models generated using data about us.

- “An algorithm is only as good as the data it works with”

- The use of data in those applications have been highly criticised for being discriminatory, racist, sexist and unfair -> probably because real-life social data is almost always biased..

Prelim & Notions

color(sensitive attribute): race, gender…

id	A_0	A_1	color	id	A_0	A_1	color
t_0	3.1	1.5	red	t_7	13	5.4	red
t_1	0.7	2.3	red	t_8	11.3	2.6	blue
t_2	8	0.65	blue	t_9	2.3	8.4	blue
t_3	10.9	1.5	red	t_{10}	5.6	4.7	red
t_4	4.4	8.7	blue	t_{11}	12.7	2.8	red
t_5	1.2	4.1	red	t_{12}	7	0.3	blue
t_6	6.2	6.3	blue	t_{13}	9.1	9.4	red

TABLE I: A toy example database $\mathbb{D}$ with two attributes A_0 and A_1 and the sensitive attribute *color*.

Range Query

id	A_0	A_1	color	id	A_0	A_1	color
t_0	3.1	1.5	red	t_7	13	5.4	red
t_1	0.7	2.3	red	t_8	11.3	2.6	blue
t_2	8	0.65	blue	t_9	2.3	8.4	blue
t_3	10.9	1.5	red	t_{10}	5.6	4.7	red
t_4	4.4	8.7	blue	t_{11}	12.7	2.8	red
t_5	1.2	4.1	red	t_{12}	7	0.3	blue
t_6	6.2	6.3	blue	t_{13}	9.1	9.4	red

QUERY 1: `select id from` $\mathbb{D}$ `where` $4 \leq A_0 \leq 7$

t4, t6, t10, t12

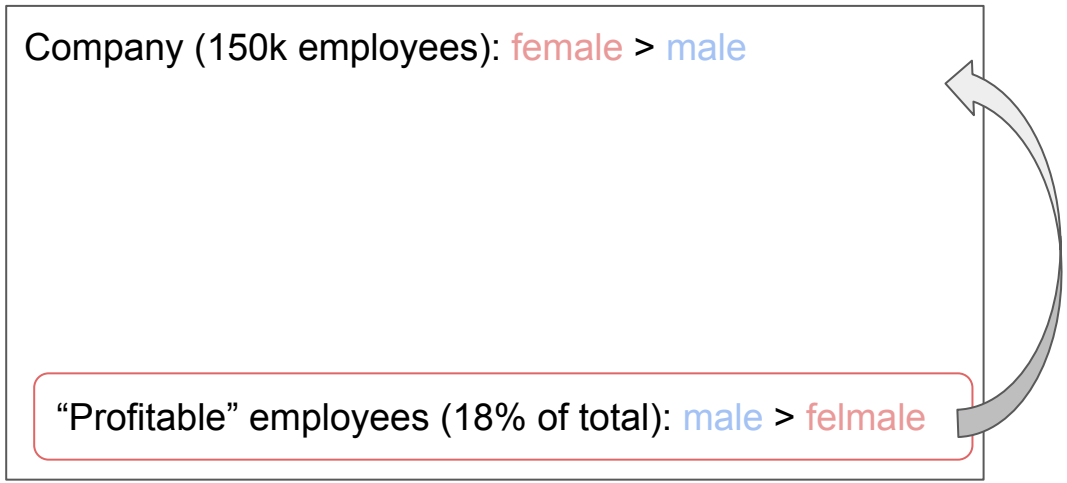
We call Query 1 as **single-predicate** range query.
Multi-predicate range queries can be something like 4 <= A0 <= 7 AND A1 >= 6

A running example

```
SELECT *
FROM EMP
WHERE salary ≥ $65K
```

A company (around 150k employees) would like to make a policy decision

Target at its “profitable” employees, using salary as an indicator of how profitable an employee is



Company will favor the preferences of the male employee, which is unfair to female employees and will, in a feedback loop, result in losing more “profitable” female candidates

Motivation

- **Selection bias** can amplify unfairness issues!

```
SELECT *

FROM EMP

WHERE salary ≥ $65K
```

What can the company do to get a fair outcome?

(achieve fairnesson on the number of male employees and female employees)

SELECT * FROM EMP WHERE salary ≥ $65K

What can the company do to get a fair outcome?

- post-query processing:
  - remove male employees
  - add female employees

Technically easy, but illegal in many jurisdictions…

SELECT * FROM EMP WHERE salary ≥ $65K

Instead of practicing disparate treatment, we adjust the SQL(range) query!

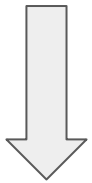
SELECT * FROM EMP WHERE $60.5K <= salary <= $152K

Its outcome is **75%** similar to the initial range query!

The number of male employees returned is **at most 1000 (around 5%) more** than females

Fairnes constraint (binary sensitive attribute only)

1. red = blue
2. |red - blue| <= k
3. (red / all red) = (blue / all blue)
4. …



To cover all the cases

$$|W_r C_r - W_b C_b| \leq \varepsilon$$

Cr, Cb: number of red objects, blue objects in the output

Wr, Wb: weight of red, weight of blue

Wr = Wb: unweighted fairness
Wr != Wb: weighted fairness

Jaccard similarity:

Jaccard similarity between two range queries can be computed by the ratio of intersection between the output of the two range queries to the union of the output to the two range queries.

$$\text{SIM}(Q_1, Q_2) = \frac{out(\mathbb{D}, Q_1) \cap out(\mathbb{D}, Q_2)}{out(\mathbb{D}, Q_1) \cup out(\mathbb{D}, Q_2)}$$

example

Jaccard similarity: Example

id	A_0	A_1	color	id	A_0	A_1	color
t_0	3.1	1.5	red	t_7	13	5.4	red
t_1	0.7	2.3	red	t_8	11.3	2.6	blue
t_2	8	0.65	blue	t_9	2.3	8.4	blue
t_3	10.9	1.5	red	t_{10}	5.6	4.7	red
t_4	4.4	8.7	blue	t_{11}	12.7	2.8	red
t_5	1.2	4.1	red	t_{12}	7	0.3	blue
t_6	6.2	6.3	blue	t_{13}	9.1	9.4	red

QUERY 1: select id from $\mathbb{D}$ where $4 \leq A_0 \leq 7$

t4, t6, t10, t12

QUERY 2: select id from $\mathbb{D}$ where $3 \leq A_0 \leq 6.2$

t0, t4, t6, t10

Jaccard_similarity(Q1, Q2) = |{t4, t6, t10}| / |{t0, t4, t6, t10, t12}| = 0.6

Problem Definition

FAIR RANGE QUERY PROBLEM: *Given a database $\mathbb{D}$, a range query Q and a disparity value ε, find a fair range that is most similar to Q with a disparity value at most ε.*

Fig. 1: Problem Formulation

DECLARATIVE FAIRNESS-AWARE QUERY:

```
SELECT ... FROM DATABASE
WHERE
    RANGE-PREDICATES
SUBJECT TO
    |W_r C_r - W_b C_b| <= eps  and SIM >= tau
```

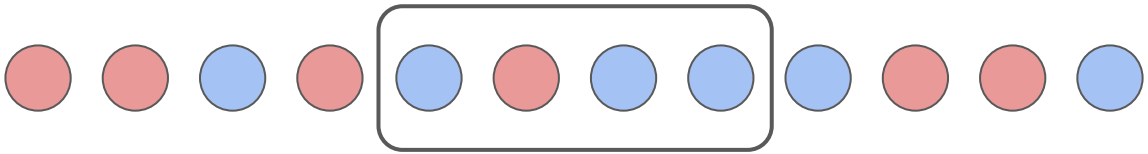
Fig. 2: Declarative Query Model

Overview

1. **Single-**predicate Range Queries
    a. Jump pointers
    b. SPQA for unweighted fairness
    c. Generalization of weighted fairness
2. **Multi-**predicate Range Queries
    a. Best-first search (BFS)
    b. IBFSMP (inspired by A*)
3. Experiments

Single-predicate Range Queries

Any brute force ideas? (fairness: red = blue)



Single-predicate Range Queries

Any brute force ideas?

O(n^2) to check all the possible single-predicate ranges (fairness and similarity)

Or we can change the two end points of the input range..

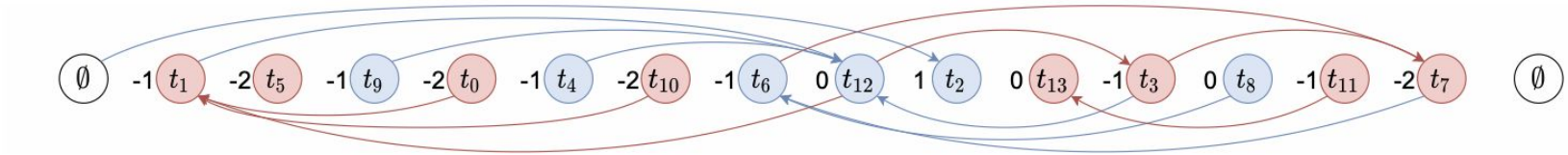
Jump Pointers:

Definition 1. *(Jump pointers): Consider a database* $\mathbb{D}$ *and the attribute A for the single-predicate range query model. A* right *(resp.* left*)* blue *jump pointer from location* o_i *points to the nearest/closest location* b_r*(resp.* b_l*) such that the number of blues in the range* $[o_i + 1, b_r]$ *(resp.* $[b_l, o_i - 1]$*) is equal to the number of reds plus one. Red jump pointers are also defined in the same manner.*

Jump Pointers:

Red left: go left, the first position that we get one more red
Red right: go right, the first position that we get one more red
Blue left: go left, the first position that we get one more blue
Blue right: go right, the first position that we get one more blue

t1 → t12: blue right JP, we get {t5, t9, t0, t4, t10, t6, t12}, so we will get one more blue
t1 ← t0: red left JP, we get {t9, t5, t1}, so we will get one more red



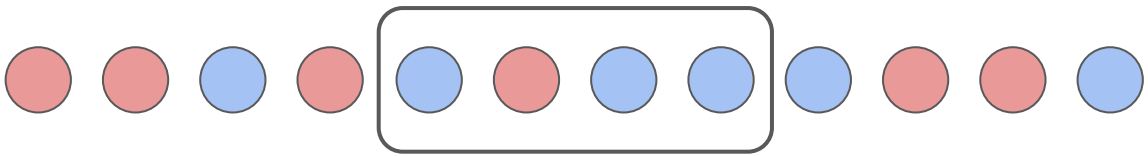
(a) Right and left jump pointers for attribute A_0 of the sample database of Table 1.

Jump Pointers:

Why?

like an index that can direct us to get closer to the fairness

(fairness: red = blue)



If the disparity (#red - #blue) = -2 < 0, we are targeting at get one more red or remove one more blue, and repeat repeat until we get the fairness range

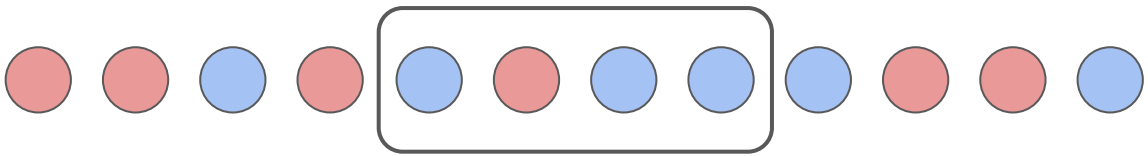
Jump Pointers:

How to get 4 jump pointers for each object? – Binary Search Tree (BST) in O(nlogn)

Why?

like an index that can direct us to get closer to the fairness

(fairness: red = blue)



If the disparity (#red - #blue) = -2 < 0, we are targeting at get one more red or remove one more blue, and repeat repeat until we get the fairness range

Algorithm: SPQA

Algorithm 1 *SPQA* Algorithm

Input : Database $\mathbb{D}$, Input query $Q(A_j : [start, end])$, acceptable disparity ε
Output : Most similar fair range query $fair$

$t_s \leftarrow binary\_search(\mathbb{D}, A_j, start)$
$t_e \leftarrow binary\_search(\mathbb{D}, A_j, end)$
$disparity \leftarrow c[j, t_s] - c[j, t_e]$
dColor $\leftarrow disparity > 0$ ▷ deficient: true-red, false-blue
if $disparity \leq \varepsilon$ **then** ▷ Input range is already fair
    return $[start, end]$
$LEP \leftarrow t_s$ ▷ LEP stands for Left End Point
while $disparity > \varepsilon$ **do**
    Push $JP(\mathbb{D}, A_j, LEP, "left", deficient)$ to LEP, update disparity for $[t_{LEP}, t_e]$
$fair \leftarrow \{\}$; $sim \leftarrow 0$
$WindowSweep(t_{LEP}, t_e,$ "shrink") ▷ Shift window by shrinking t_e
$WindowSweep(t_{LEP}, t_e,$"expand")
$WindowSweep(t_s, t_e,$"shrink") ▷ Shift win. by shrinking t_e
$WindowSweep(t_s, t_e,$ "expand")
return $fair$

Algorithm 2 $WindowSweep$ algorithm

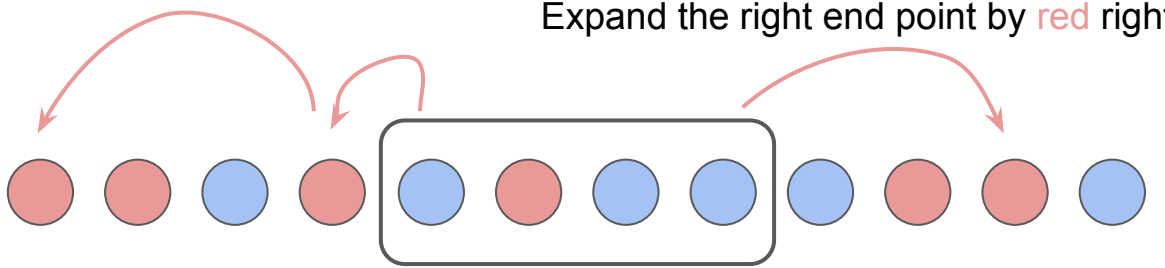
Input : Database $\mathbb{D}$, Attribute: A_j, start end-point: t_s, end end-point: t_e, operation, input query $Q(A_j : [start, end])$, reference to $fair$, sim
Output : Update $fair$ based on most fair range found

$disparity \leftarrow c[j, t_e] - c[j, t_s - 1]$
$dColor \leftarrow disparity > 0$
if $t_s < start$ **then** $t_s \leftarrow$ Pop(LEP)
else $t_s \leftarrow JP(\mathbb{D}, A_j, t_s, "right", !dColor)$ ▷ Shrink t_s
while $disparity > \varepsilon$ **do** ▷ Adjust t_e pointer
    $disparity \leftarrow c[j, t_e] - c[j, t_s - 1]$
    $dColor \leftarrow disparity > 0$
    if $operation == "expand"$ **then**
        $t_e \leftarrow JP(\mathbb{D}\ A_j, t_e, "right", dColor)$ ▷ Expand
    else $t_e \leftarrow JP(\mathbb{D}\ A_j, t_e, "left", !dColor)$ ▷ Shrink
if $Jaccard([t_s, t_e], Q) > sim$ **then**
    $sim \leftarrow Jaccard([t_s, t_e], Q)$; $fair \leftarrow [t_s, t_e]$
$WindowSweep(t_s, t_e, operation)$

Algorithm: SPQA

(fairness: red = blue)

Expand the right end point by red right pointer get one more red



Expand the left end point by red left pointer get one more red

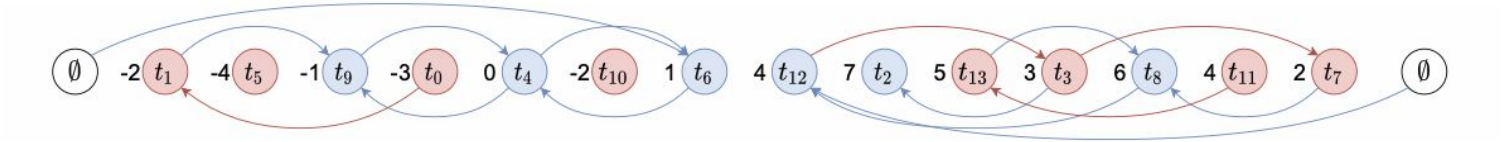
You may ask:
Whether we can use the jump pointers to shrink the end points instead of expanding? Sure! Although it is a little bit more complicated…

Time complexity: O(log(n) + disparity(input)) -> sublinear

Weighted Single-predicated Range Queries?

Combining the weights into the generation of jump pointers

- it can also solve co-located objects



(b) Right and left jump pointers for attribute A_0 of the sample database of Table 1 *with weights*

Experiments: proof of concept

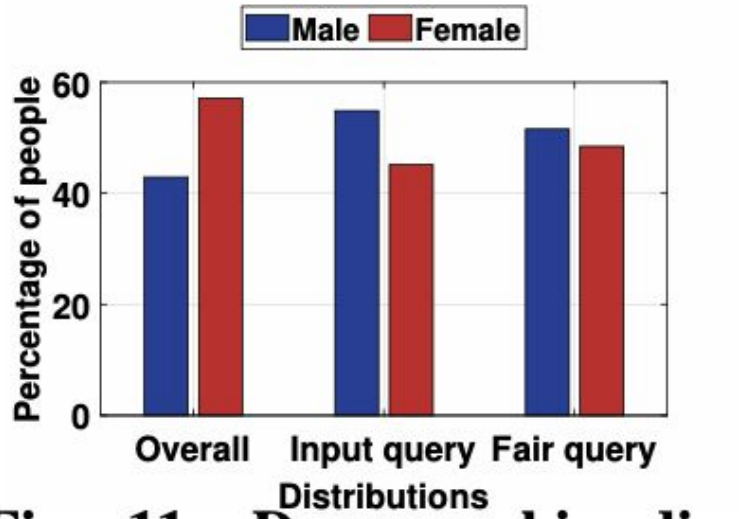


Fig. 11: Demographic distributions in dataset, input query, and similar fair query.

Input query: salary > $65000
SPQA: $60562 <= salary <= $152000 (76.23% Jaccard similarity), at most 1000 (around 5%) males more than females

Experiments: Systems Integration

SPQA vs. O(n^2) naive algorithm

Average time taken:

0.0054 seconds vs. 6.938 seconds

1200x faster

Experiments: IBFS (for multi-predicate)

vs. single predicate: 3 orders of magnitude **slower** than the jump pointer algorithm (SPQA)

vs. baseline: 3.6 seconds against 697.4 seconds

Multi-predicate Range Queries

complicates the problem significantly

- the idea of jump pointer does not carry over
    - there are many different directions which a single jump can occur

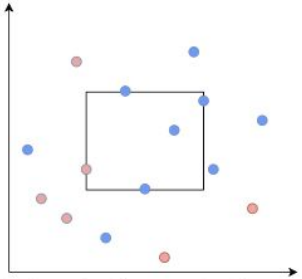


Fig. 6: Sample input range

Observation: user might be not interested in fair ranges that are too far away from the input query. (should be highly similar)

Multi-predicate Range Queries

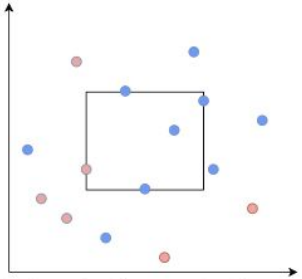


Fig. 6: Sample input range

complicates the problem significantly

- the idea of jump pointer does not carry over
  - there are many different directions which a single jump can occur

Observation: user might be not interested in fair ranges that are too far away from the input query. (should be highly similar)

Searching!

Neighbor:

At a high level, the BFS algorithm can be viewed as a "smart" traversal over a graph where every range is modeled as a node and there is an edge between two nodes if the outputs of their corresponding queries vary only in one tuple. That is, a node Q_2 is a *neighbor* of Q_1 if the output of query Q_1 differs from the output of query Q_2 by exactly 1 element. Mathematically, sets $out(\mathbb{D}, Q_1)$ and $out(\mathbb{D}, Q_1)$ have a symmetric difference of size 1.

BFSMP: Best-first Search algorithm for Multi-predicate

Algorithm 3 Best-First Search algorithm for MP : BFSMP

Input : Database $\mathbb{D}$, attribute list A, input query Q
Output : *most similar fair range*

```
1: Heap ← Q
2: while |Heap| ≠ 0 do
3:     top ← Heap.pop()
4:     if fair(top) then return top
5:     for neighbor ∈ neighbors(top) do
6:         Heap.push(neighbor)
7: return ∅
```

1. Starting from the input query
2. Calculate all the “neighbors”
3. Use a max-heap (priority queue) with the key as similarity to maintain all the ranges
4. Check the one with highest similarity, if it satisfies the fairness constraint, done.

How to calculate neighbors?

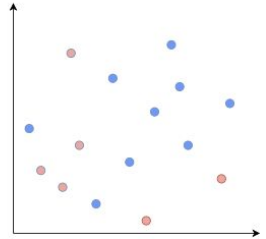


Fig. 5: Sample set of points

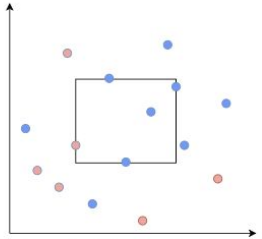


Fig. 6: Sample input range

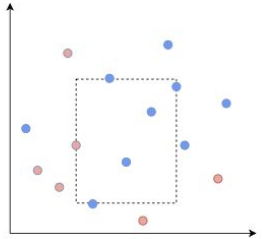


Fig. 7: Expanding the rectangle downwards

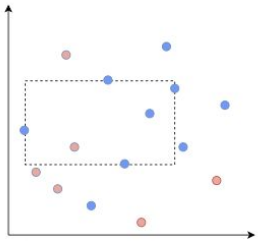


Fig. 8: Expanding the rectangle towards left

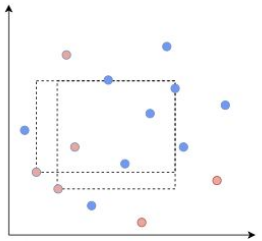


Fig. 9: Neighboring ranges in diagonal

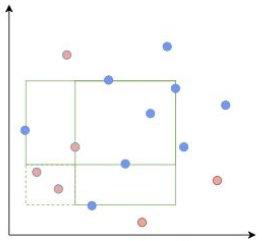


Fig. 10: Skyline computation over a range query

Expanding sides: O(log^d_n + k) using a range tree

Expanding corners: O((k + 1)log^d_n) using Range-Skyline-Query Algorithm, where k is the size of skyline

Optimization of Searching: heuristic

(Inspired by A*)
IBFSMP: Informed best first search
Use a heuristic to prune the searching tree

U-threshold: the upper-bound threshold on the maximum similarity for a fair range that one can hope to achieve by branching out from the current node

Theorem 4. *The U-threshold of a node Q is:*

$$J_U(Q) = \begin{cases} \max\limits_{C'_r \le \lceil \frac{\delta-\varepsilon}{W_r} \rceil} \frac{I - \lceil \frac{max(\delta-\epsilon-W_r \cdot C'_r, 0)}{W_b} \rceil}{U + C'_r} & W_r > W_b;\ \delta > \varepsilon \\ \max\limits_{C'_b \le \lceil \frac{\delta-\varepsilon}{W_b} \rceil} \frac{I - C'_b}{U + \lceil \frac{max(\delta-\epsilon-W_b \cdot C'_b, 0)}{W_r} \rceil} & W_b > W_r;\ \delta > \varepsilon \\ \max\limits_{C'_r \le \lceil \frac{\delta-\varepsilon}{W_r} \rceil} \frac{I - C'_r}{U + \lceil \frac{max(-\delta-\epsilon-W_r \cdot C'_r, 0)}{W_b} \rceil} & W_r > W_b; \delta < -\varepsilon \\ \max\limits_{C'_b \le \lceil \frac{\delta-\varepsilon}{W_b} \rceil} \frac{I - \lceil \frac{max(-\delta-\epsilon-W_b \cdot C'_b, 0)}{W_r} \rceil}{U + C'_b} & W_b > W_r; \delta < -\varepsilon \end{cases} \quad (3)$$

where $\delta = W_b \cdot C_b - W_r \cdot C_r$.

Instead of selecting the most similar node(range query) to be explored next, IBFSMP selects the node with maximum U-threshold to be explored next

Take-away Points & Possible Future directions

1. Integrate fairness into DBMS
2. Design selection query (rubric) instead of post-processing/disparate treatment
3. Jump pointers & Neighbors: building an index helps
4. Transform (sub-)problems/concepts into computational geometry
5. Binary fairness constraint -> arbitrary fairness constraint
6. Range queries -> other SQL queries