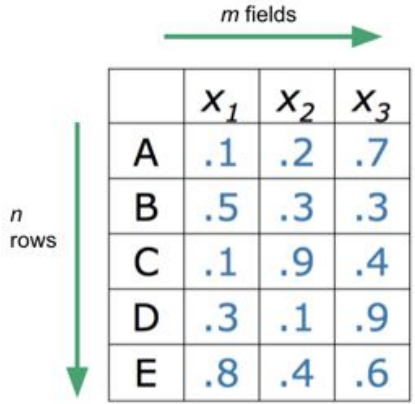
Aggregation Papers!

UW DB Seminar Slides 2015-11-09

Dylan Hutchison, Dominik Mortiz,
Kanit "Ham" Wongsuphasawat

Motivation

- query may be fuzzy (probability rather than boolean value)
    - For example, redness of an object varies.



	x_1	x_2	x_3
A	.1	.2	.7
B	.5	.3	.3
C	.1	.9	.4
D	.3	.1	.9
E	.8	.4	.6

- Each attribute has a scoring function
- **Goal: Query based on** Ranking score that combines graded attributes through **an aggregation function**

redness images from www.cse.iitd.ac.in/~ramanath/CSL868-2011/**slides**/suresh.pptx

Incorporating User Preferences in Multimedia Queries

Ronald Fagin, Edward L. Wimmers. ICDT 1997: 247-261. Paper.

Goal: Formula for weighted score combination for any aggregation function.

Based On & Compatible

- **Based on**
  - if all weights are equal, the resulting score is equivalent to unweighted score.
- **Compatible**
  - if a particular argument has zero weight, then it can be dropped without affecting result.

These two properties are essential to create a weighted collection,

but **not sufficient for** determining a **unique weighted collection**

Local Linearity

- Combining with the other two properties produces a unique weight collection
- **order-equivalence:** having similar order of importance
  - (.3, .5, .2) is order-equivalent to (.2, .7, .1)
- Local linearity: balance between order equivalent weightings

Formally, we say that a weighted collection $\mathcal{F}$ of scoring functions is *locally linear* if whenever Θ and Θ' are order-equivalent and $\alpha \in [0, 1]$, then

$$f_{\alpha\cdot\Theta+(1-\alpha)\cdot\Theta'}(X) = \alpha \cdot f_{\Theta}(X) + (1-\alpha) \cdot f_{\Theta'}(X).$$

weight

if Θ is ordered by σ, $\theta_{\sigma(1)} \geq \theta_{\sigma(2)} \geq \ldots \geq \theta_{\sigma(m)}$.

$\sigma[i] = \{\sigma(1), \ldots, \sigma(i)\}$, for $1 \leq i \leq m$ = set of indices of the i largest θ_j's.

and $m = card(I)$

weight score weight score

$$f_\Theta(X) = m \cdot \theta_{\sigma(m)} \cdot b_{\sigma[m]}(X) + \sum_{i=1}^{m-1} i \cdot (\theta_{\sigma(i)} - \theta_{\sigma(i+1)}) \cdot b_{\sigma[i]}(X). \quad (2)$$

rewrite (2) as a linear combination of the θ_i's:

$$f_\Theta(X) = \theta_{\sigma(1)} \cdot b_{\sigma[1]}(X) + \sum_{i=2}^{m} \theta_{\sigma(i)} \cdot (i \cdot b_{\sigma[i]}(X) - (i-1) \cdot b_{\sigma[i-1]}(X)). \quad (3)$$

f_Θ depend only on $b_{\sigma[1]}, \ldots, b_{\sigma[m]}$

Example: $m = 3$, each b_I is min, and $\theta_1 \geq \theta_2 \geq \theta_3$

f_Θ is a convex combination of the three terms x_1, $\min(x_1, x_2)$, and $\min(x_1, x_2, x_3)$ only

Inherited Properties

The weighted function *F(B)* inherits properties of combined scoring functions if every scoring function in *B*:

is **continuous**, is **monotonic**, is **translation-preserving**,

satisfies **betweenness**, satisfies **identity**

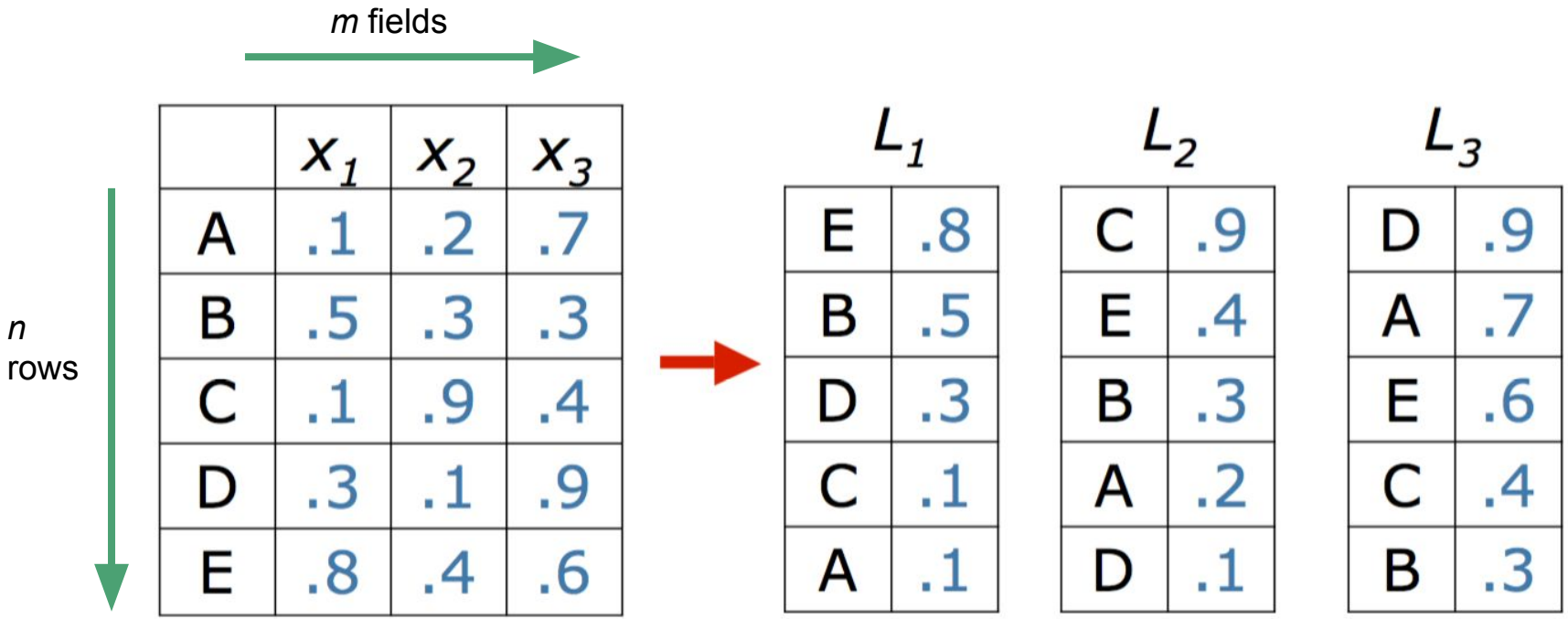
Also, ***If $\mathcal{B}$ is symmetric, then $\mathcal{F}(\mathcal{B})$ is symmetric.***

Optimal aggregation algorithms for middleware

Ronald Fagin, Amnon Lotem, Moni Naor. PODS 2001.
Paper. Journal version.

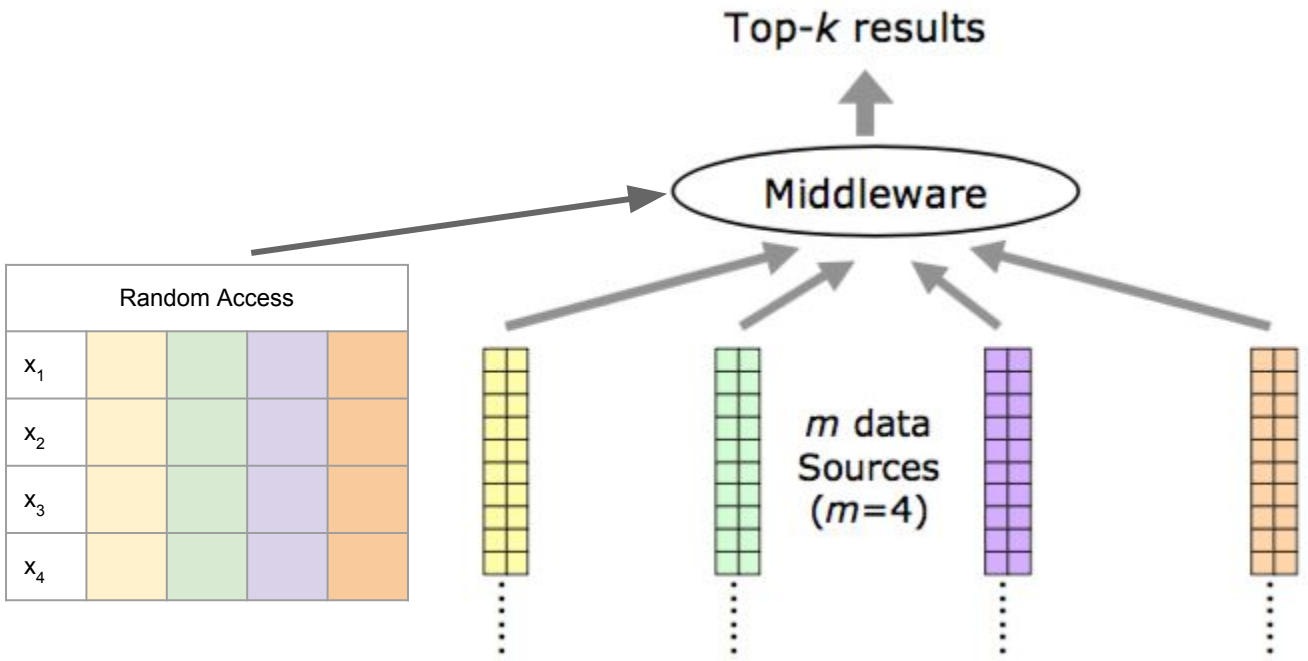
Goal: Optimizing Top-K query

Table



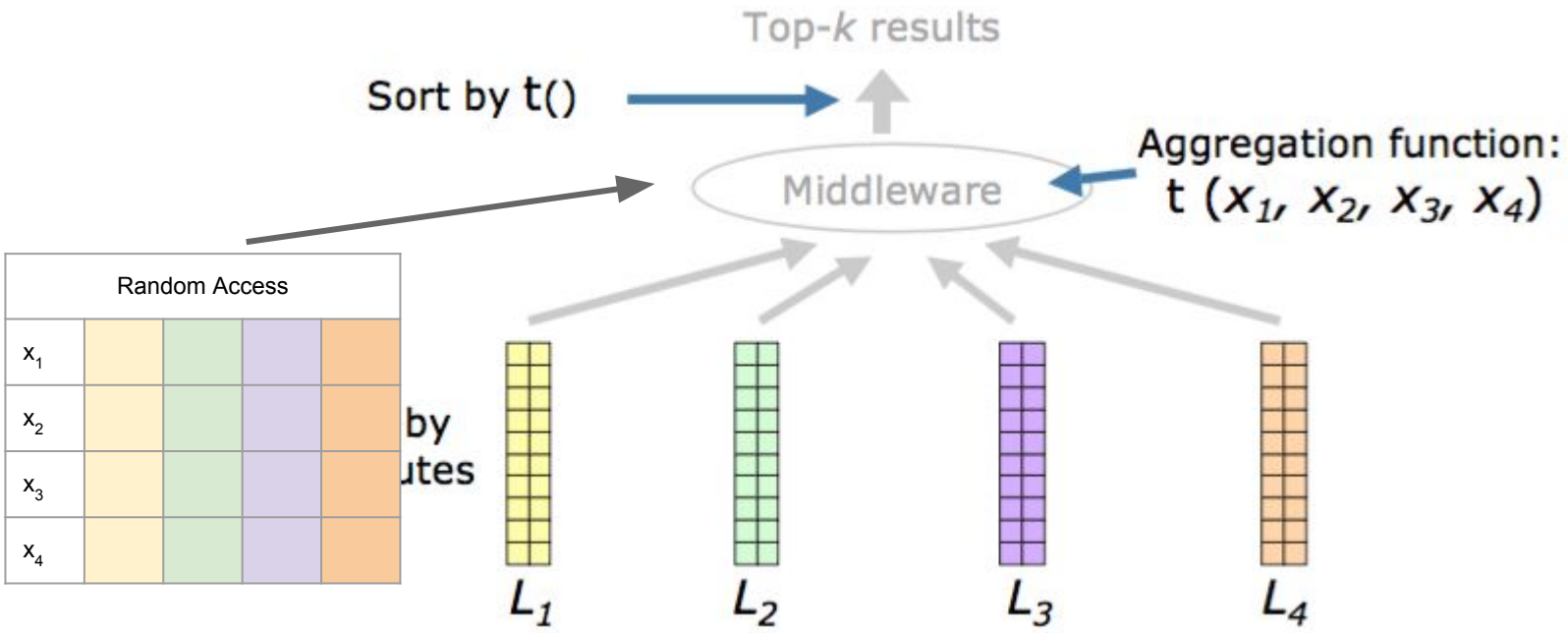
Images from
https://cs.uwaterloo.ca/~tozsu/courses/cs856/F05/Presentations/Week11/optagg.pdf

Middleware (Data Integration Systems)



Images from
https://cs.uwaterloo.ca/~tozsu/courses/cs856/F05/Presentations/Week11/optagg.pdf

Middleware (Data Integration Systems)

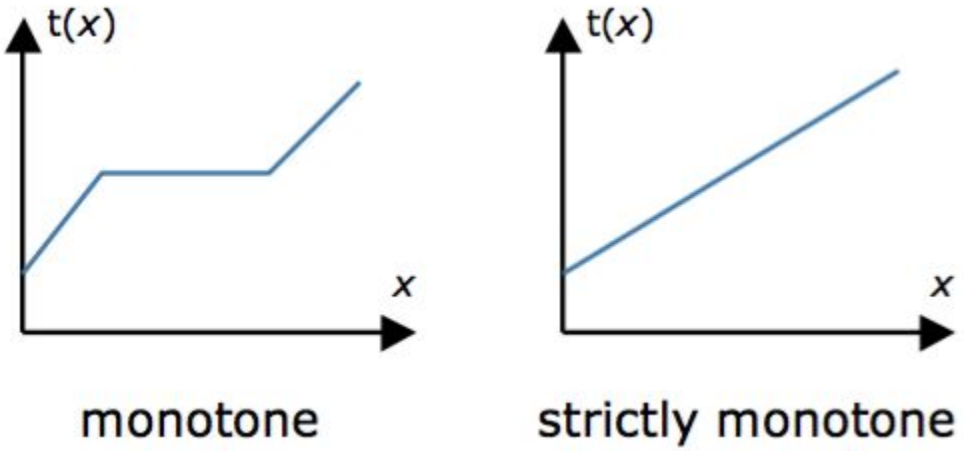


Images from
https://cs.uwaterloo.ca/~tozsu/courses/cs856/F05/Presentations/Week11/optagg.pdf

Monotone Aggregation Functions

min(), max(), avg(), ...

$$\forall\ x_i \leq y_i \rightarrow t(x_1, x_2, ..., x_m) \leq t(y_1, y_2, ..., y_m)$$



Images from
https://cs.uwaterloo.ca/~tozsu/courses/cs856/F05/Presentations/Week11/optagg.pdf

Base Problem

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Idea for both Fagin and Threshold

Do sorted access (and the corresponding random access) until you know you have seen the top k answers.

Access sorted lists “in parallel”.

On seeing a new (object x, grade) in a sorted list:

Find the other grades via random access to x and compute x’s overall grade

Check stopping condition!

Fagin: Stop when k objects seen in all sorted lists.

Threshold: Stop when k seen objects have overall grade ≥ threshold (threshold is the overall grade of worst grades seen)

Fagin’s Algorithm

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8

Aggregation is sum
Computing Top 1

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8

Aggregation is sum
Computing Top 1

No need to re-random-lookup w; w is in cache

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8

Aggregation is sum
Computing Top 1

No need to re-random-lookup w; w is in cache

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3

Aggregation is sum
Computing Top 1

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3
z	1.4

Aggregation is sum
Computing Top 1

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3
z	1.4

Aggregation is sum
Computing Top 1

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3
z	1.4

Aggregation is sum
Computing Top 1

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3
z	1.4

Aggregation is sum
Computing Top 1

FA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	**2.8**
x	2.3
z	1.4

Aggregation is sum
Computing Top 1

Stop! Object x seen in all lists.
Answer is w with overall grade 2.8

“Proof”: Any unseen object (such as y) has grades ≤ grades of x.
Therefore any unseen object must have overall grade ≤ overall grade of x.

Threshold Algorithm

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	

Threshold
4

Aggregation is sum
k=1 (top 1 object)

- Bounded buffers: only need store k (object, overall_grade) in cache
- Stop when overall_grade ≥ threshold

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8

Threshold
3.9

Aggregation is sum

k=1 (top 1 object)

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8

Threshold
3.8

Aggregation is sum
k=1 (top 1 object)

w stored in cache; no need to lookup again

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8

Threshold
3.7

Aggregation is sum
k=1 (top 1 object)

w stored in cache; no need to lookup again

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.2

Threshold
3.6

Aggregation is sum

k=1 (top 1 object)

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.2
z	1.4

Threshold
3.1

Aggregation is sum
k=1 (top 1 object)

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.2

Threshold
2.7

Aggregation is sum

k=1 (top 1 object)

Need to lookup x again if not kept in cache

Stop! Overall grade of object w ≥ threshold

Answer is object w with overall grade 2.8

"Proof": Any unseen object (such as y) has grades ≤ worst grades seen so far in sorted lists.

Object w has overall grade ≥ overall grade of worst grades seen so far (which is the threshold)

Therefore any unseen object's overall grade

≤ overall grade of worst objects seen so far (the threshold)

≤ w's overall grade

Proof: TA finds the top k answers

Let Y be set containing the k objects that have been seen in TA.

Need to show that for $y \in Y$ and $z \notin Y$, $t(z) \leq t(y)$.

$z = (x_1, x_2, ..., x_m)$, since z has not been seen:

$x_i < \underline{x}_i \rightarrow t(z) = t(x_1, x_2, ..., x_m) \leq t(\underline{x}_1, \underline{x}_2, ..., \underline{x}_m) = \tau$ (from monotonicity of t)

By definition of Y, we have $t(y) \geq \tau$

So, $t(y) \geq \tau \geq t(z)$

Instance Optimality

We say that an algorithm $\mathscr{B}$ is *instance optimal over* $\mathbf{A}$ *and* $\mathbf{D}$ if $\mathscr{B} \in \mathbf{A}$ and if for every $\mathscr{A} \in \mathbf{A}$ and every $\mathscr{D} \in \mathbf{D}$ we have

$$cost(\mathscr{B}, \mathscr{D}) = O(cost(\mathscr{A}, \mathscr{D})). \tag{2}$$

Eq. (2) means that there are constants c and c' such that $cost(\mathscr{B}, \mathscr{D}) \leqslant c \cdot cost(\mathscr{A}, \mathscr{D}) + c'$ for every choice of $\mathscr{A} \in \mathbf{A}$ and $\mathscr{D} \in \mathbf{D}$. We refer to c as the *optimality ratio*.

Intuition: TA is instance optimal

If **A** is an algorithm that stops sooner than TA on some database, before **A** finds k objects whose grade is at least equal to the threshold value τ; then **A** must make a mistake on some database, since the next object in each list might have grade x_i in each list i; and hence have grade $t(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_m) = \tau$. This new object, which **A** has not even seen, has a higher grade than some object in the top k list that was output by **A**; and so **A** erred by stopping too soon.

True for algorithms that do not make “wild guesses”
(random access to object not yet seen in a sorted list),
or for databases that satisfy the distinctness property
(no two grades in a sorted list are the same),
and strictly monotone aggregation functions

Approximation algorithm

- Find top k such that θ t(y) ≥ t(z)
- Simply adjust threshold to τ/θ

No Random Access Algorithm (NRA)

- Can’t calculate exact score / order of the top-k
- But still can calculate lower and upper bound of each objects
- Sequential access until there are k objects whose lower bound ≥ the upper bound of all other objects

No Random Access Algorithm (NRA)

Access sequentially all lists in parallel until there are k objects for which the lower bound (unknown to 0) is higher than the upper bound (unknown to x) of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

	LB	UB
X_1	1	2.6
X_2	.8	2.6
X_4	.8	2.6

1+0+0 **1**+0.8+0.8

0+**0.8**+0 1+**0.8**+0.8

0+0+**0.8** 1+0.8+**0.8**

Aggregation: sum

Images from
http://alumni.cs.ucr.edu/~skulhari/Top-k-Query.pdf

No Random Access Algorithm (NRA)

Access sequentially all lists in parallel until there are k objects for which the lower bound (unknown to 0) is higher than the upper bound (unknown to x) of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

	LB	UB
X_2	1.6	2.2
X_3	1.3	2.1
X_1	1	2.3
X_4	0.8	2.3

0.8+0.8+0 **0.8+0.8+0.6**

0+0.7+0.6 **0.8+0.7+0.6**

1+0+0 **1+0.7+0.6**

0+0+0.8 **0.8+0.7+0.8**

Images from
http://alumni.cs.ucr.edu/~skulhari/Top-k-Query.pdf

No Random Access Algorithm (NRA)

Access sequentially all lists in parallel until there are k objects for which the lower bound (unknown to 0) is higher than the upper bound (unknown to x) of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

	LB	UB
X_3	1.8	1.8
X_2	1.6	1.8
X_1	1.5	1.5
X_4	0.8	1.6

Images from
http://alumni.cs.ucr.edu/~skulhari/Top-k-Query.pdf

No Random Access Algorithm (NRA)

Access sequentially all lists in parallel until there are k objects for which the lower bound (unknown to 0) is higher than the upper bound (unknown to x) of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

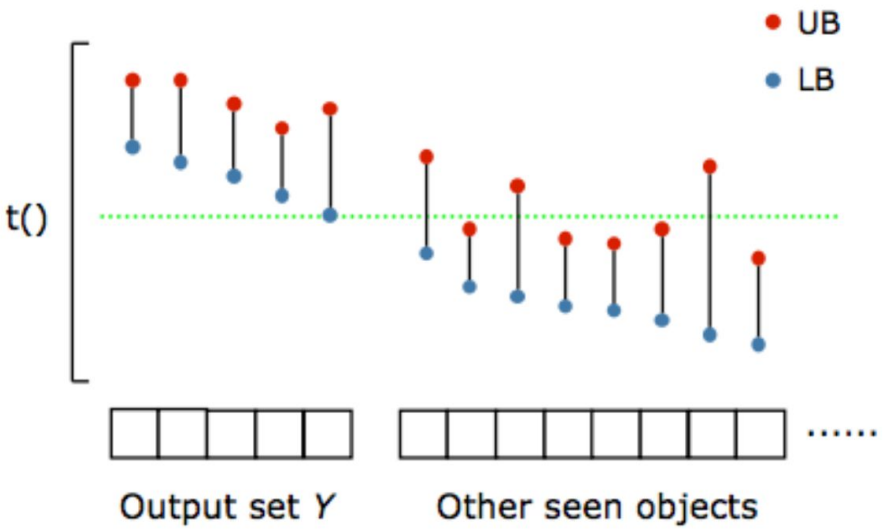
	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

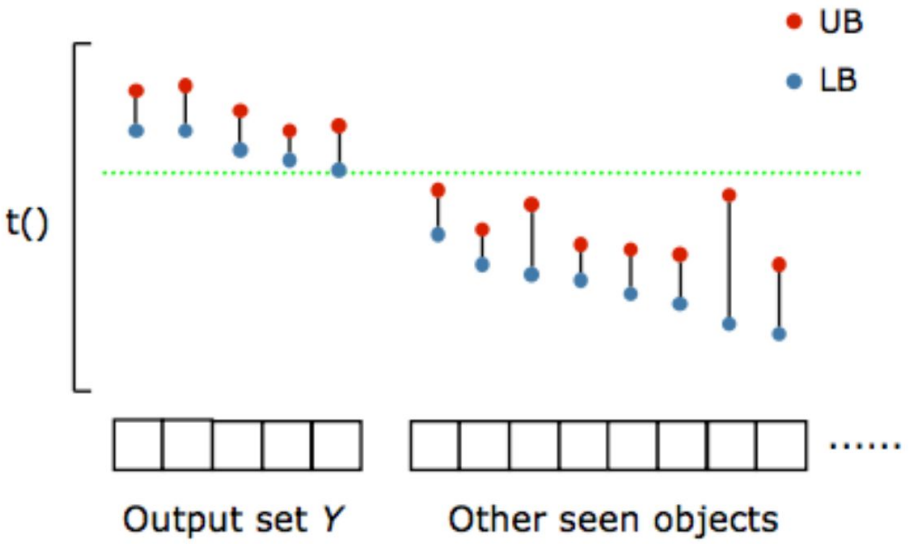
	LB	UB
X_3	1.8	1.8
X_2	1.6	1.8
X_1	1.5	1.5
X_4	0.8	1.6

Return top 2 objects

Images from
http://alumni.cs.ucr.edu/~skulhari/Top-k-Query.pdf

No Random Access Algorithm - Visual





Combined Algorithm (CA)

- Combining TA and NRA

- Run NRA but also run random access step for every *h* steps

- NRA and CA are instance optimal when the aggregation function is monotone

Discussion / Questions

- What if t is not monotone? For example with negative values in DB.
- How does the algorithm change for bottom k?
- Why the “no wild guesses” / strict monotone and distinctness clause?
- Can we restrict sorted access?
- Do we have to get next value from all lists in TA or can we do it step by step? Is there an early stopping condition? Can we rule out one list?
- Can we avoid random requests in some cases (for instance for sum, avg, ...)?

Can we restrict sorted access? Yes, we can restrict sorted access by letting the threshold calculation assume 1 for attributes with no sorted access

TA - Non-lockstep sorted list access

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	

Threshold
4

- Aggregation is sum
- k=1 (top 1 object)

- Bounded buffers: only need store k (object, overall_grade) in cache
- Stop when overall_grade ≥ threshold

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
x	2.3

Threshold
3.9

Aggregation is sum
Computing Top 1

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3

Threshold
3.8

Aggregation is sum
Computing Top 1

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	2.8
x	2.3

Threshold
3.7

Aggregation is sum
Computing Top 1

TA

Random Access Table				
w	.9	.9	.9	.1
x	.4	.5	.4	.9
y	.2	.3	.4	.2
z	.4	.5	.2	.3

A	B	C	D
w, .9	w, .9	w, .9	x, .9
z, .4	x, .5	x, .4	z, .3
x, .4	z, .5	y, .4	y, .2
y, .2	y, .3	z, .2	w, .1

Cache	
w	**2.8**
x	2.3
z	1.4

Threshold
2.1

Aggregation is sum
Computing Top 1

Stop! Value of w > threshold