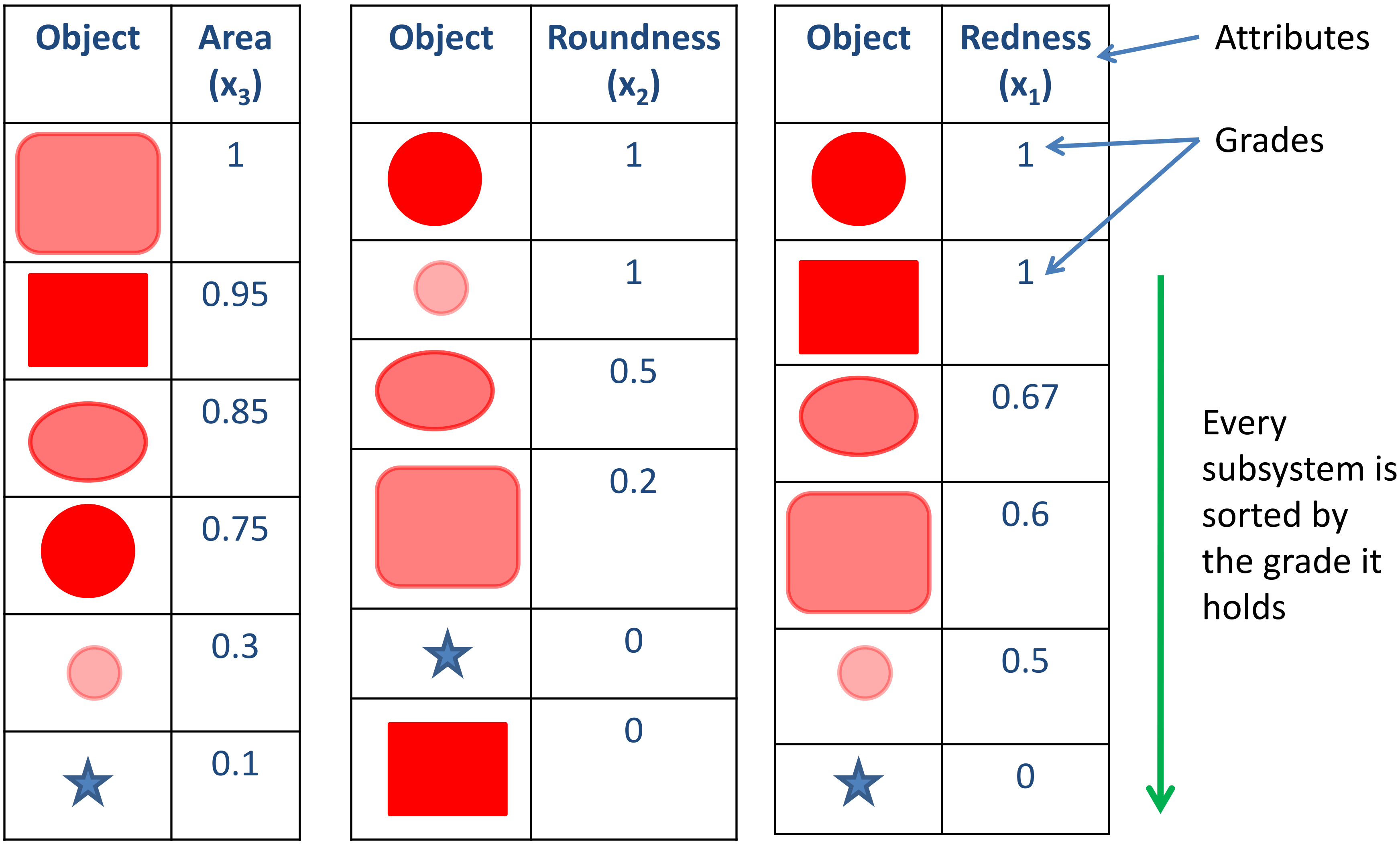
Problem Addressed

- Given a collection of objects, our goal is to find Top-k objects, whose scores are greater than the remaining objects.

A sample set of Databases



Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

Before Moving On….

- Aggregate Function : Aggregate functions perform a calculation on a set of values and return a single value.

  Eg: sum(), min()

- Monotone: In mathematics, a monotonic function is a function between ordered sets that preserves the given order.

  i.e $t(x_1,\ldots,x_m) \leq t(x'_1,\ldots,x'_m)$ if $x_i \leq x'_i$ for every I

  Eg :

Before Moving on

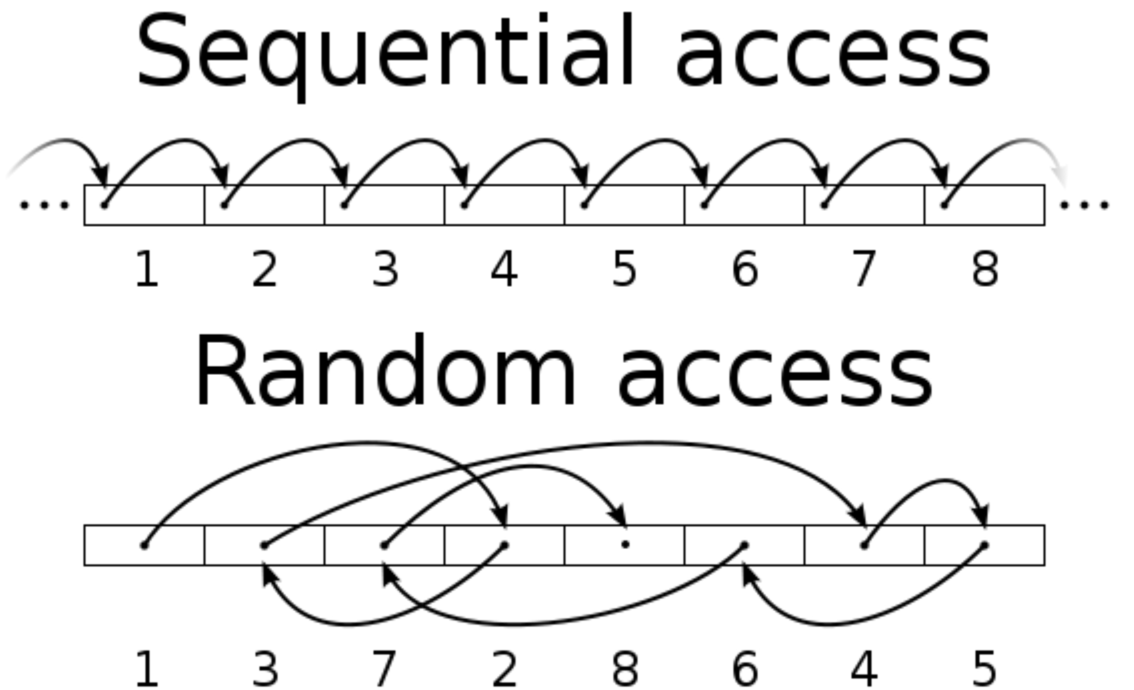
- Strictly Monotone:

  $t(x_1,\ldots,x_m) < t(x'_1,\ldots,x'_m)$ if $x_i < x'_i$ for every i

- Strict Monotone :

  $t(x_1,\ldots,x_m) = 1$ precisely when $x_i = 1$ for every i

Before Moving On



Before Moving On....

A = class of algorithms, $A \in \mathbf{A}$ represents an algorithm

D = legal inputs to algorithms (databases), $D \in \mathbf{D}$ represents a database

middleware cost = cost for processing data subsystems = $sc_S + rc_R$

Cost(A,D) = middleware cost when running algorithm A over database D

Algorithm B is instance optimal over **A** and **D** if :

$B \in \mathbf{A}$ and $\text{Cost}(B,D) = O(\text{Cost}(A,D))\ \forall A \in \mathbf{A},\ \forall D \in \mathbf{D}$

Which means that:

$\text{Cost}(B,D) \leq c \,.\, \text{Cost}(A,D) + c',\quad A \in \mathbf{A},\ \forall D \in \mathbf{D}$

↑

optimality ratio

Top-k Object Problem

- Naïve Algorithm
- Fagin’s Algorithm
- Threshold Algorithm

Naïve Algorithm

- Basic Idea:
  - For for each object, use the aggregation function to get the score
  - According to the scores, get the top *k*.
  - Since the time complexity is linear, it is not efficient for large database.

Questions

- Do we need to count the score for every object in the database?
- Can we SAFELY ignore some objects whose scores are lower than what we already have?

Fagin's Algorithm

- Do Sorted access in parallel at all the lists
- Stop when we have k objects which appear in all the lists
- Calculate score value of all the objects
- Compute Top-k objects

Example: Fagin's Algorithm

1 Objects appear in every list:

{ }

Objects seen so far:

{ , }

Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

k = 3

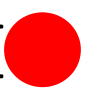
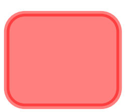
Example: Fagin's Algorithm

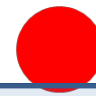
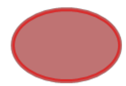
2

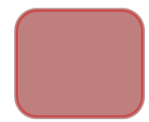
Objects appear in every list:

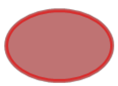
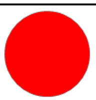
{ }

Objects seen so far:

{, , ,  }

Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

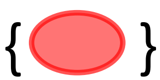
Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

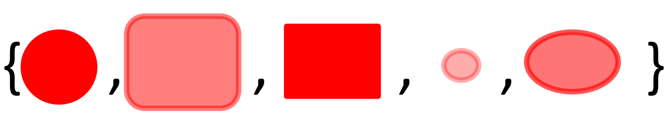
$k = 3$

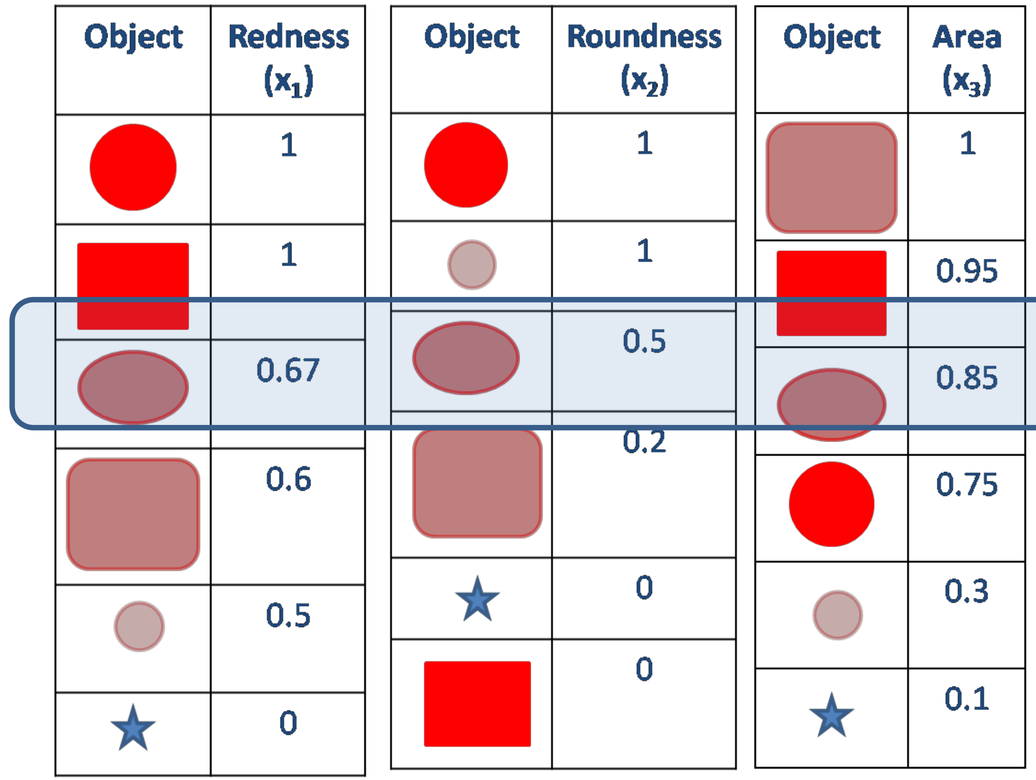
Example: Fagin's Algorithm

3 Objects appear in every list:



Objects seen so far:



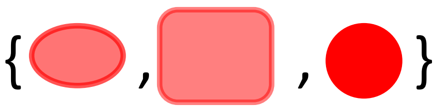


k = 3

Example: Fagin’s Algorithm

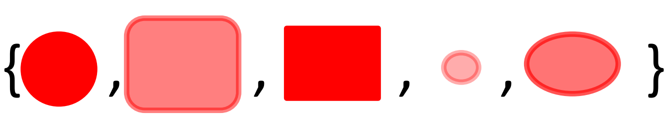
4

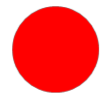
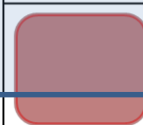
Objects appear in every list:

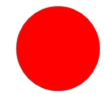
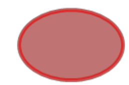


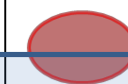
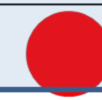
We got enough objects

Objects seen so far:



Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

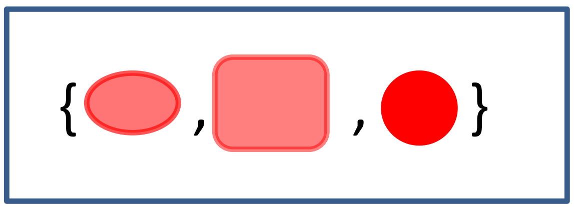
Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

k = 3

Example: Fagin's Algorithm

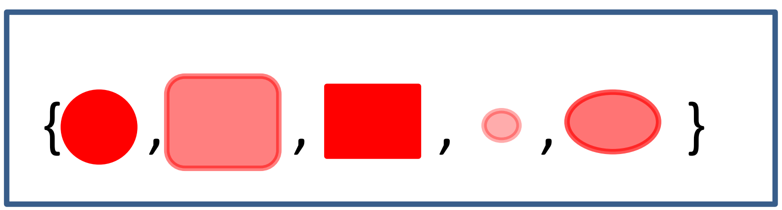
4

Objects appear in every list:

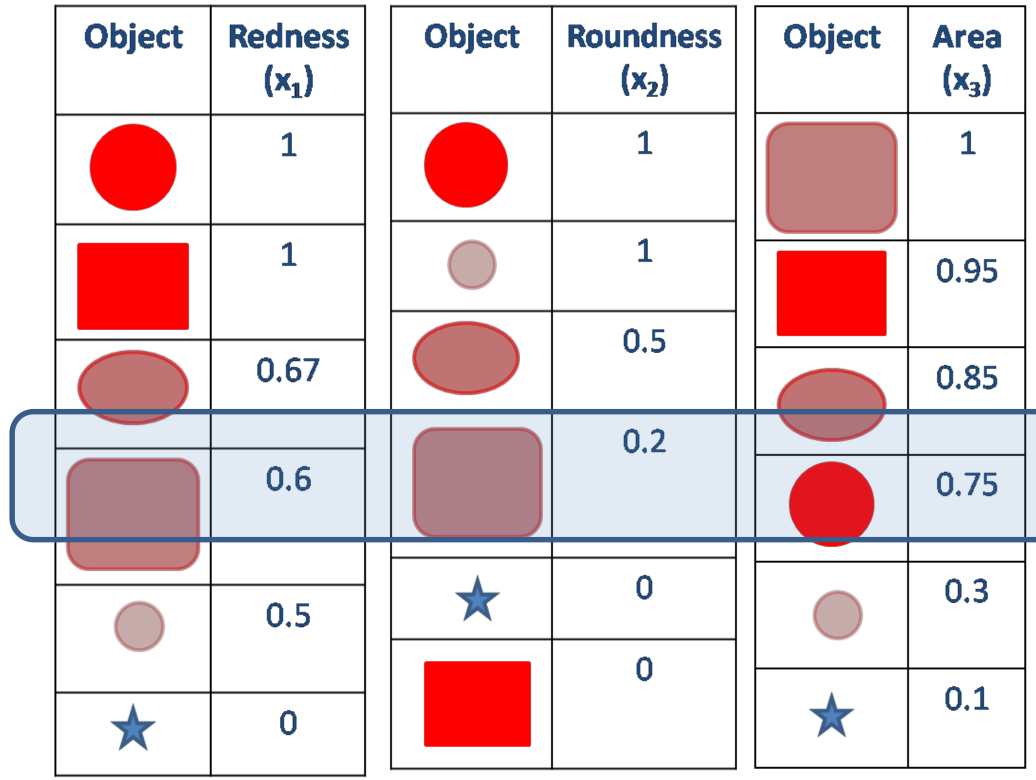


We got enough objects

Objects seen so far:



For all these, calculate the score and get the Top-k



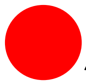
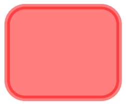
k = 3

The Threshold Algorithm

- Do Sorted access in parallel at all the lists until $\tau < g$
  - For each object *R* that has been seen at least once in any of the list
    - Do random accesses to get the attribute values of *R* from the lists where the object has not been seen yet.
    - Compute *t(R)* and update the list of top *k* objects (Y) if necessary.
  - Compute $\tau = t(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_m)$ where x_i is the grade of the last seen object from list L_i under sorted access.
  - If τ is less than the lowest aggregated grade (g) of the top *k* set (Y) then halt.

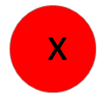
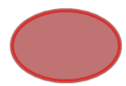
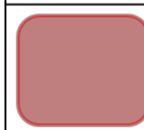
Example : Threshold Algorithm

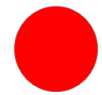
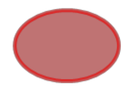
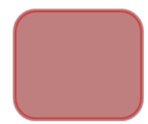
iterations

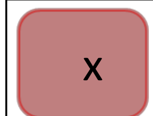
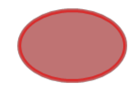
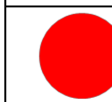
1 $\tau = 3$, Y = {  ,  }

$g = 1.8$

t=sum and k=3

Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

x-marked objects are the first to be seen of their kind and when seen they have been accessed in the other databases randomly to compute their aggregate function.

Example : Threshold Algorithm

iterations

1 $\tau = 3$, Y = { , }
g = 1.8

2 $\tau = 2.95$, Y = { , , }
g = 1.8

t=sum and *k=3*

Object	Redness (x_1)
X	1
X	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
X	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
X	1
	0.95
	0.85
	0.75
	0.3
	0.1

x-marked objects are the first to be seen of their kind and when seen they have been accessed in the other databases randomly to compute their aggregate function.

Example : Threshold Algorithm

iterations

1 $\tau = 3$, Y = { , }
$g = 1.8$

2 $\tau = 2.95$, Y = { , , }
$g = 1.8$

3 $\tau = 2.02$, Y = { , , }
$g = 1.95$

t=sum and *k=3*

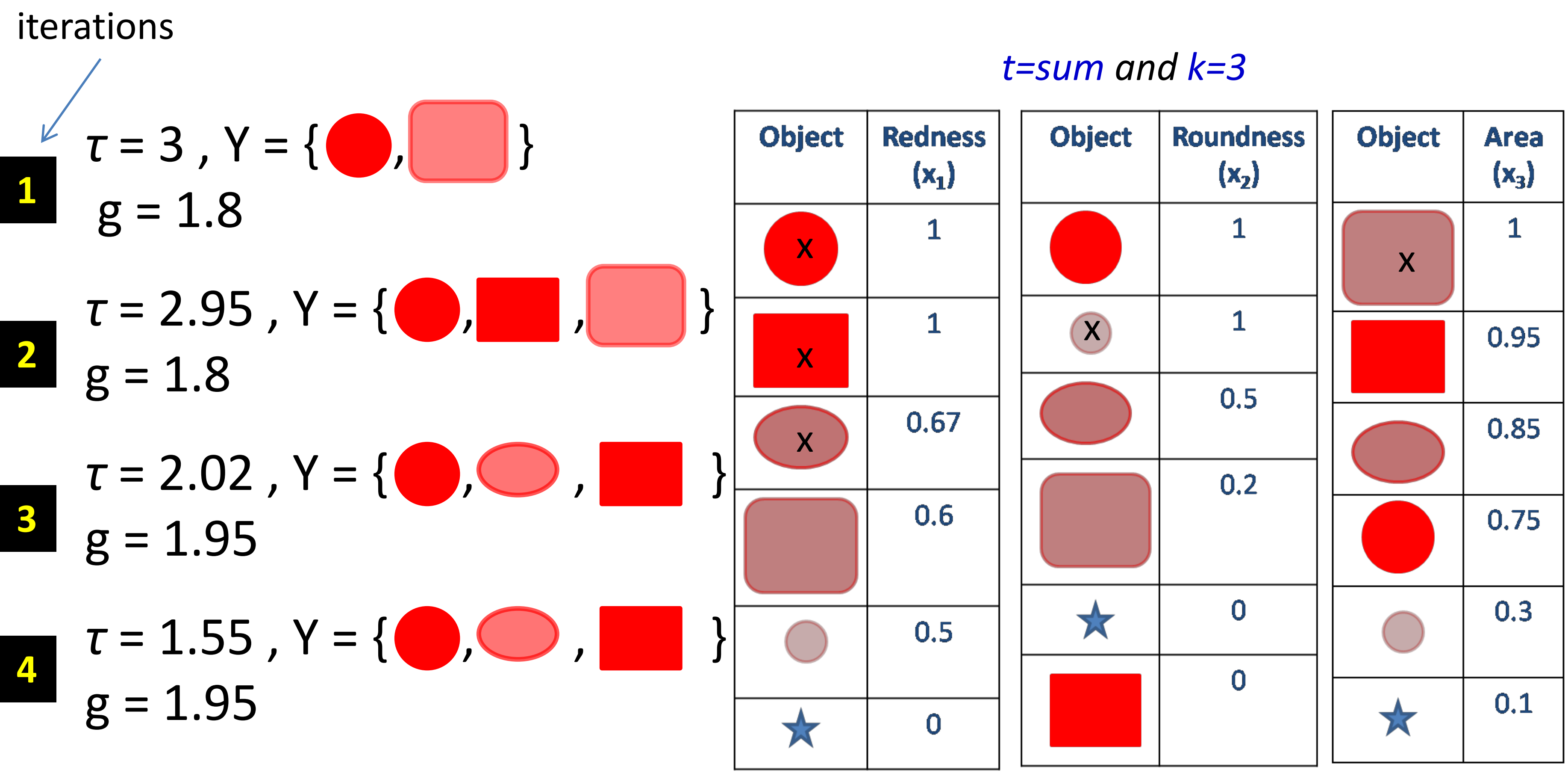
Object	Redness (x_1)
X	1
X	1
X	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
X	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
X	1
	0.95
	0.85
	0.75
	0.3
	0.1

x-marked objects are the first to be seen of their kind and when seen they have been accessed in the other databases randomly to compute their aggregate function.

Example : Threshold Algorithm



x-marked objects are the first to be seen of their kind and when seen they have been accessed in the other databases randomly to compute their aggregate function.

When Sorted Access is Restricted

- ϑ-approximation to the top k answers for the aggregation function t is a collection of k objects (each along with its grade) such that for each y among these k objects and each z not among these k objects, ϑ t(y)>=t(z)
- T_{ϑ} : As soon as at least k objects have been seen whose grade is at least equal to threshold/ ϑ then halt.

Comparison of Fagin's and Threshold Algorithm

- TA sees less objects than FA
    - TA stops at least as early as FA
        - When we have seen *k* objects in common in FA, their grades are higher or equal than the threshold in TA.

- TA may perform more random accesses than FA
    - In TA, (*m*-1) random accesses for each object
    - In FA, Random accesses are done at the end, <u>only for missing grades</u>

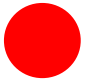
- TA requires only bounded buffer space (*k*)
    - At the expense of more random seeks
    - FA makes use of unbounded buffers

Restricting Sorted Access

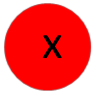
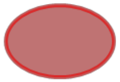
- A subset Z' of the databases are not accessible under sorted access.
- TA is modified to handle such scenario.
- $\tau = t(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_m)$ where x_i is 1 for all inaccessible database L_i.
- All databases in Z' are accessed only under random access mode.

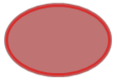
Restricting Sorted Access

1

$\tau = 3$, Y = {}

g = 2.75

Object	Redness (x_1)
 X	1
	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

t=sum and *k=3*

Inaccessible und
sorted access

x-marked objects are the first to be seen of their kind

Restricting Sorted Access

1 $\tau = 3$, Y = { }
$g = 2.75$

2 $\tau = 3$, Y = { , , }
$g = 1.8$

Object	Redness (x_1)
X	1
X	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
X	1
	0.5
	0.2
	0
	0

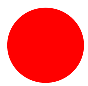
Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

t=sum and *k=3*

x-marked objects are the first to be seen of their kind

Inaccessible und sorted access

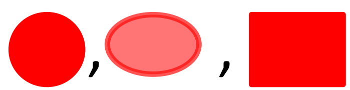
Restricting Sorted Access

1 $\tau = 3$, $Y = \{$  $\}$

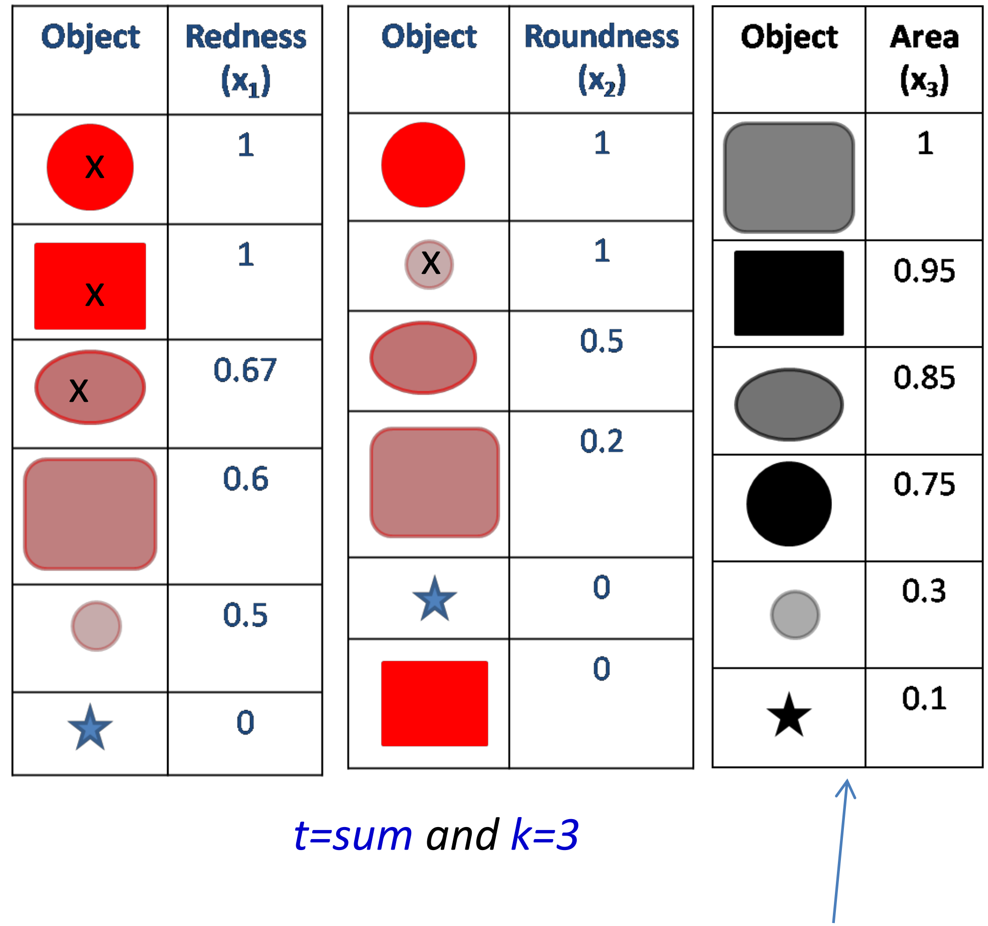
$g = 2.75$

2 $\tau = 3$, $Y = \{$  $\}$

$g = 1.8$

3 $\tau = 2.17$, $Y = \{$  $\}$

$g = 1.95$



t=sum and k=3

x-marked objects are the first to be seen of their kind

Inaccessible und
sorted access

Restricting Sorted Access

1 $\tau = 3$, Y = { }
g = 2.75

2 $\tau = 3$, Y = { , , }
g = 1.8

3 $\tau = 2.17$, Y = { , , }
g = 1.95

4 $\tau = 1.8$, Y = { , , }
g = 1.95

Object	Redness (x_1)
X	1
X	1
X	0.67
X	0.6
	0.5
	0

Object	Roundness (x_2)
	1
X	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

t=sum and *k=3*

Inaccessible und sorted access

x-marked objects are the first to be seen of their kind

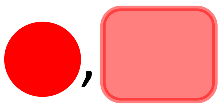
Restricting Random Access

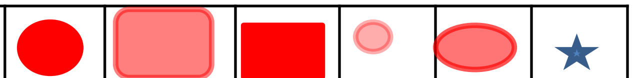
- If t is a monotone , $W(R)$ is a lower bound on $t(R)$ computed by replacing unknown attribute values with 0 in t.
- $B(R)$ is an upper bound on $t(R)$ computed by replacing unknown attribute values with the least value seen in the database.
- Here Y is the top k list that contains k objects with the largest W values seen so far. Ties broken by B values and then arbitrarily.

Example: Restricting Random Access

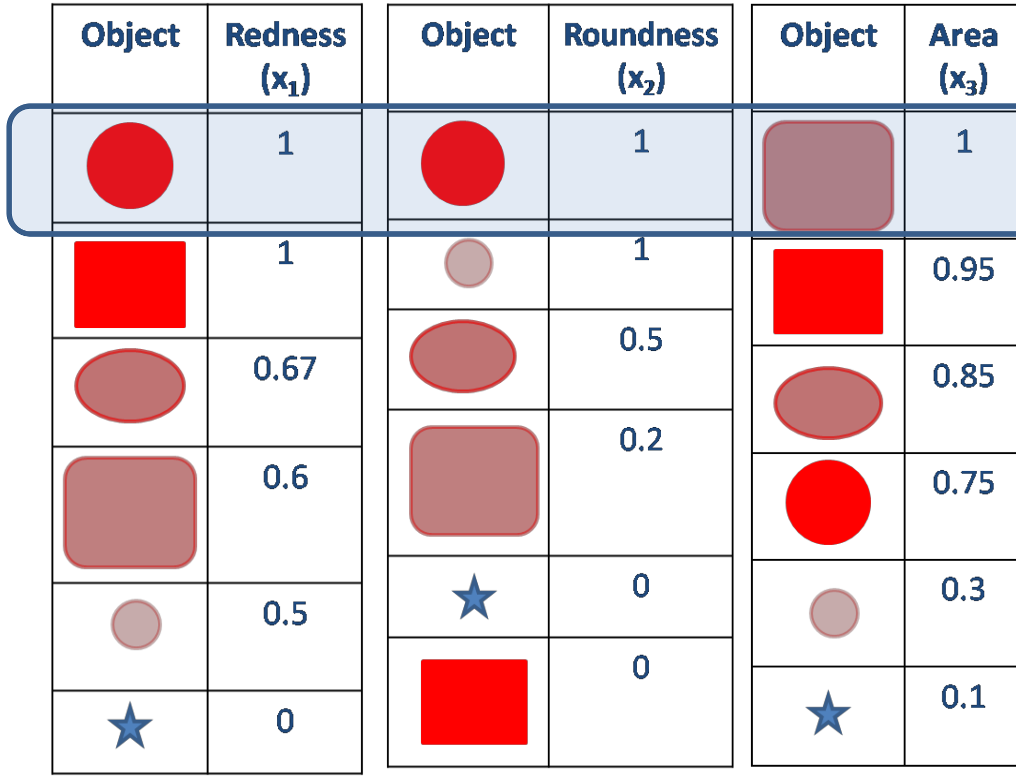
Y is the sorted top-*k* list

1

Y = {  }



x_1	1	-	-	-	-	-
x_2	1	-	-	-	-	-
x_3	-	1	-	-	-	-
W	2	1	0	0	0	0
B	3	3	3	3	3	3



Example: Restricting Random Access

2

Y = { , , }

x_1	1	-	1	-	-	-
x_2	1	-	-	1	-	-
x_3	-	1	0.95	-	-	-
W	2	1	1.95	1	0	0
B	2.95	3	2.95	2.95	2.95	2.95

Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

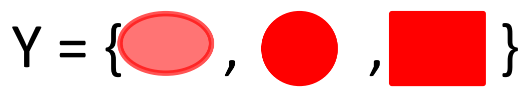
Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

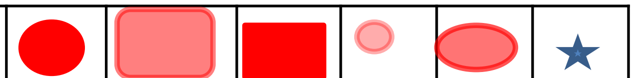
Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

W(= 1+0+0.95 = 1.95

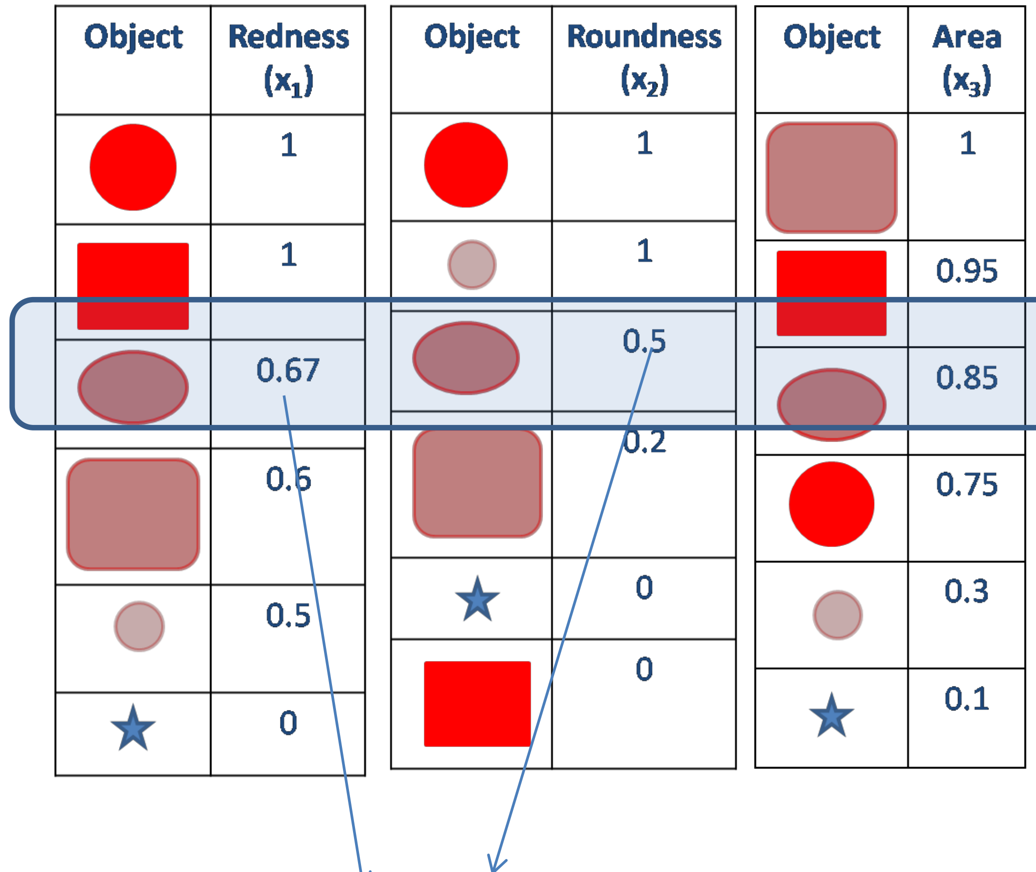
Example: Restricting Random Access

3



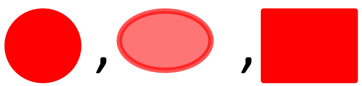


x_1	1	-	1	-	0.67	-
x_2	1	-	-	1	0.5	-
x_3	-	1	0.95	-	0.85	-
W	2	1	1.95	1	2.02	0
B	2.85	2.17	2.45	2.52	2.02	2.02



Example: Restricting Random Access

4

Y = {  }

x_1	1	0.6	1	-	0.67	-
x_2	1	0.2	-	1	0.5	-
x_3	0.75	1	0.95	-	0.85	-
W	2.75	1.8	1.95	1	2.02	0
B	2.75	1.8	2.05	2.35	2.02	1.55

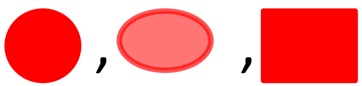
Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

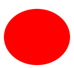
Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

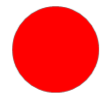
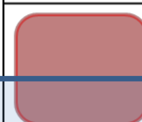
Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

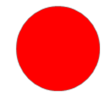
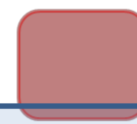
Example: Restricting Random Access

5

Y = {  }

						
x_1	1	0.6	1	0.5	0.67	-
x_2	1	0.2	-	1	0.5	0
x_3	0.75	1	0.95	0.3	0.85	-
W	2.75	1.8	1.95	1.8	2.02	0
B	2.75	1.8	1.95	1.8	2.02	0.8

Object	Redness (x_1)
	1
	1
	0.67
	0.6
	0.5
	0

Object	Roundness (x_2)
	1
	1
	0.5
	0.2
	0
	0

Object	Area (x_3)
	1
	0.95
	0.85
	0.75
	0.3
	0.1

At this point the algorithm halts because all the objects not in *Y* have smaller *B* values than the smallest *W* value in the *Y* which is 1.95 here.

Instance Optimality: Fagin's Algorithm

- Database with N objects, each with m attributes.
- Orderings of lists are independent
- FA finds top-k with middleware cost $O(N^{(m1)/m}k^{1/m})$
- FA = optimal with high probability in the worst case for strict monotone aggregation functions

Instance Optimal : Threshold Algorithm

- TA = instance optimal (always optimal) for **every monotone** aggregation function, over every database **(excluding wild guesses)**
  = optimal in much stronger sense than Fagin's Algorithm

- If strict monotone aggregation function:
  Optimality ratio = m + m (m-1)c_R/c_s = best possible (m = # attributes)

  - If random acces not possible (c_r = 0) → optimality ratio = m
  - If sorted access not possible (c_s = 0) → optimality ratio = infinite

  → TA not instance optimal

- TA = instance optimal (always optimal) for every strictly monotone aggregation function, over every database (including wild guesses) that satisfies the distinctness property

- Optimality ratio = cm^2 with

$c = \max \{c_R/c_S, c_S/c_R\}$

Algorithm Comparision

(from Zhang2002 talk)

Algorithm	Assumption	Access Model	Termination Worst Case	Termination Expected	Buffer Space
FA	Monotone	Sorted Random	$n(m-1)/m + k/m$	$N^{m-1/m} k^{1/m}$	N
TA	Monotone	Sorted Random	Bounded by FA	Depends on distribution	k
NRA	Monotone	Sorted	N	Depends on distribution	N