

Ranking Queries

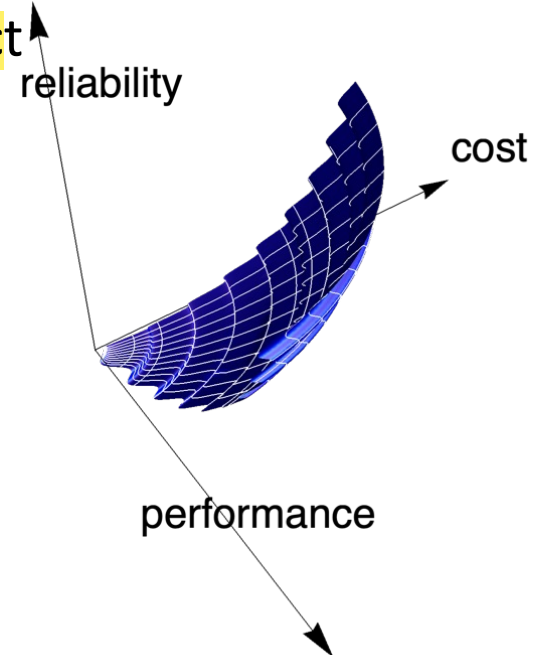
and other means
to find the best objects

Finding the best results

- Data are
 - available in large quantities
 - represented under many different models
 - queried with many different languages
 - processed with many different algorithms
- ...but how to get the best results out of the data?
- What does “best” even mean?

Multi-objective optimization

- Simultaneous optimization of different criteria
 - E.g., different attributes of objects in a dataset
- A general problem formulation:
 - Given N objects described by d attributes
 - and some notion of “goodness” of an object
 - Find the best k objects



Relevance of the problem

- Search engines, e-commerce
- Recommender Systems
- Machine Learning
 - k nearest neighbors
 - Feature selection
 - Classification
- Caching
- ...

Classical applications

- **Multi-criteria queries**
 - Example: ranking hotels by combining criteria about available facilities, driving distance, stars, ...

Search

Destination/property name:

Check-in date

Thursday 19 December 2... ▼

Check-out date

Friday 20 December 2019 ▼

1-night stay

▼

▼ ▼

☐ I'm travelling for work ?

Distance from Central Tokyo


- | | |
|---|-----|
| <input type="checkbox"/> Less than 1 km | 25 |
| <input type="checkbox"/> Less than 3 km | 272 |
| <input type="checkbox"/> Less than 5 km | 932 |

Online payment

- | | |
|---------------------------------|------|
| <input type="checkbox"/> PayPal | 1424 |
|---------------------------------|------|

Fun things to do

- | | |
|---|-----|
| <input type="checkbox"/> Massage | 328 |
| <input type="checkbox"/> Hot tub/jacuzzi | 133 |
| <input type="checkbox"/> Bicycle rental (additional charge) | 114 |
| <input type="checkbox"/> Public Bath | 113 |
| <input type="checkbox"/> Sauna | 86 |



Hotel Keihan Asakusa

Taito, Tokyo · [Show on map](#) · 6 km from centre

Just booked for your dates 11 minutes ago

Black Friday Sale

Double Room with Small Double Bed - Non-Smoking

1 double bed

Risk free: You can cancel later, so lock in this great price today.

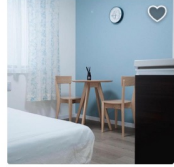
Very good 8.2
2,397 reviews

1 night, 2 adults
€ 58

Additional charges may apply

FREE cancellation
No prepayment needed

[Choose your room >](#)



Apartment Attrait Kita Shinjuku

Shinjuku Ward, Tokyo · [Show on map](#) · 4 km from centre

One-Bedroom Apartment

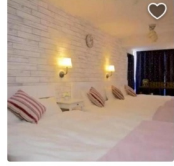
1 bedroom • 1 bathroom
1 single bed • 2 futon beds
20 m²

Only 1 left like this on our site

1 night, 2 adults
€ 120

includes taxes and charges

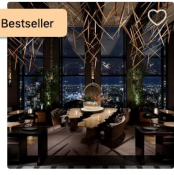
[See availability >](#)



Apartment Minato-Azabujuban 1 BR Apartment GAE52

Minato, Tokyo · [Show on map](#) · Metro access

You missed it!
Your dates are popular – we've run out of rooms at this property! Check out more below.



Shinagawa Prince Hotel

Minato, Tokyo · [Show on map](#) · 6 km from centre

Black Friday Sale

Twin Room

Bestseller

Good 7.8
9,677 reviews

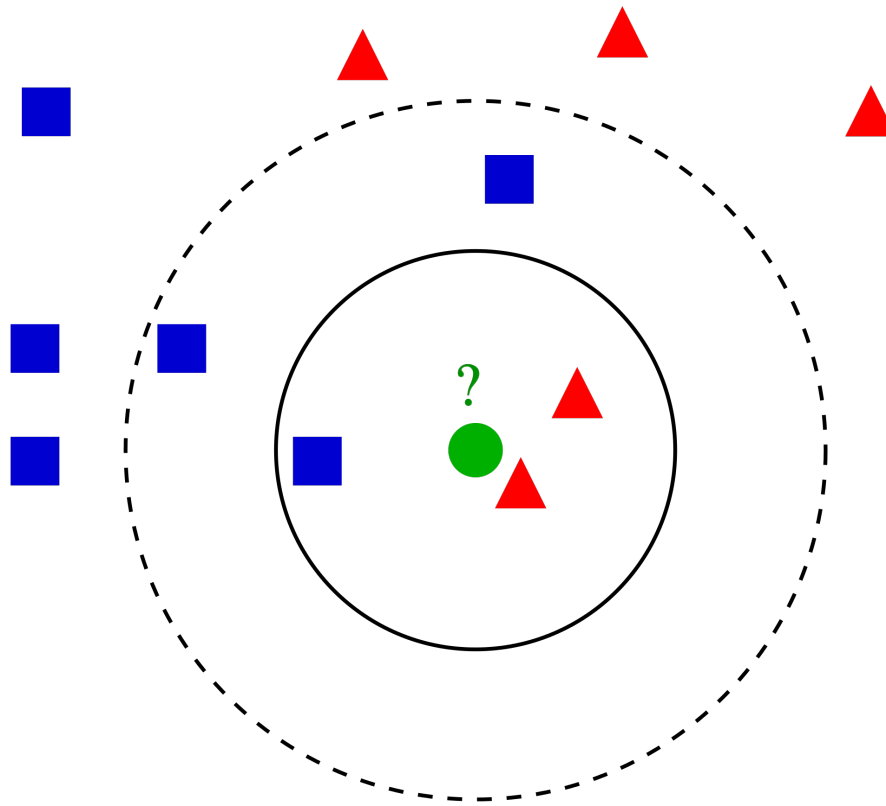
1 night, 2 adults
€ 140.21

includes taxes and charges

[See available room >](#)

Classical applications

- **k-Nearest Neighbors** (e.g., similarity search)
 - Given N points in some metric (d -dimensional) space, and a query point q in the same space, find the k points closest to q
 - Used for classification in Machine Learning



Multi-objective optimization

- Main approaches:
 - Ranking (aka top-k) queries
 - Top k objects according to a given scoring function
 - Skyline queries
 - Set of non-dominated objects

Dominating an object means being better than that object in all dimensions

Historical perspective

Rank aggregation

[Borda, 1770][Marquis de Condorcet, 1785][Llull, 13th century]

- Rank aggregation is the problem of combining several ranked lists of objects in a robust way to produce a single consensus ranking of the objects



Jean-Charles de Borda



Marie Jean Antoine Nicolas de Caritat,
Marquis de Condorcet



Ramon Llull

Rank aggregation

- Old problem (social choice theory) with lots of open challenges
- Given: n candidates, m voters

Candidate	Candidate	Candidate	Candidate	Candidate
A	B	D	E	C
B	D	B	A	E
C	E	E	C	A
D	A	C	D	B
E	C	A	B	D
Voter 1	Voter 2	Voter 3	Voter 4	Voter 5

- What is the overall ranking according to all the Voters?
 - No visible **score** assigned to candidates, only ranking
- Who is the best candidate?

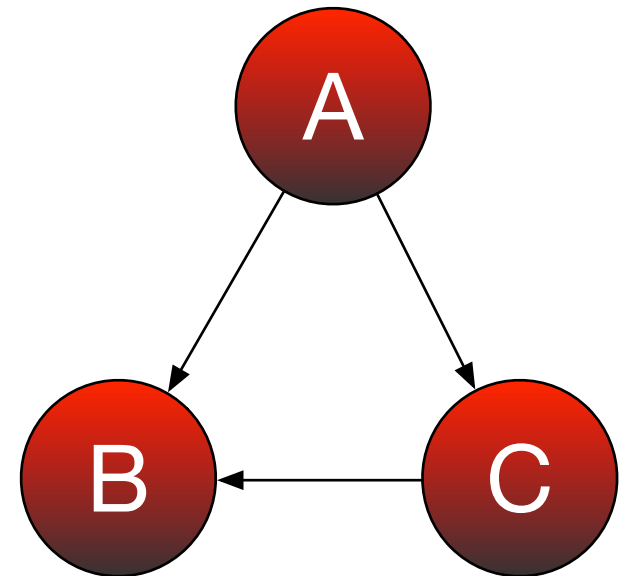
Borda's and Condorcet's proposals

- Borda's proposal
 - Election by order of merit
 - First place \rightarrow 1 point
 - Second place \rightarrow 2 points
 - ...
 - n-th place \rightarrow n points
 - Candidate's penalty: sum of points
- **Borda** winner: candidate with the lowest penalty
- **Condorcet** winner:
 - A candidate who defeats every other candidate in pairwise majority rule election

Borda winner \neq Condorcet winner

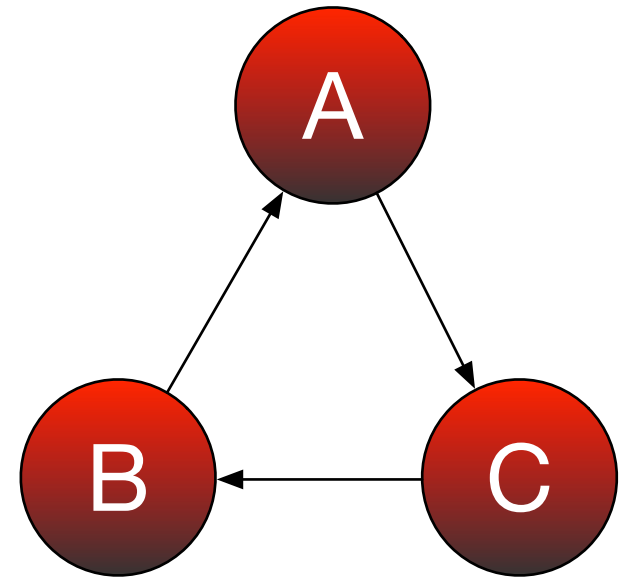
1	2	3	4	5	6	7	8	9	10
A	A	A	A	A	A	C	C	C	C
C	C	C	C	C	C	B	B	B	B
B	B	B	B	B	B	A	A	A	A

- Borda scores:
 - A: $1 \times 6 + 3 \times 4 = 18$
 - B: $3 \times 6 + 2 \times 4 = 26$
 - C: $2 \times 6 + 1 \times 4 = 16 \leftarrow$ Borda winner
- Condorcet's criterion:
 - A beats both B and C in pairwise majority
 - A is Condorcet's winner



Condorcet's paradox

1	2	3
C	B	A
B	A	C
A	C	B



- Condorcet's winner may not exist
 - Cyclic preferences

Approaches to rank aggregation

[Arrow, 1950]

■ Axiomatic approach

- Desiderata of aggregation formulated as “axioms”
- Arrow’s result: a small set of natural requirements cannot be simultaneously achieved by any nontrivial aggregation function
- **Arrow’s paradox**: no rank-order electoral system can be designed that always satisfies these “fairness” criteria:
 - No dictatorship (nobody determines, alone, the group’s preference)
 - If all prefer X to Y, then the group prefers X to Y
 - If, for all voters, the preference between X and Y is unchanged, then the group preference between X and Y is unchanged

Kenneth Arrow



**Perfect
democracy is
unattainable!**

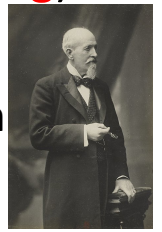
Approaches to rank aggregation

■ Metric approach

- Finding a new ranking R whose **total distance** to the initial rankings R_1, \dots, R_n is **minimized**
- Several ways to define a distance between rankings, e.g.:
 - **Kendall tau** distance $K(R_1, R_2)$, defined as the number of exchanges in a bubble sort to convert R_1 to R_2
 - **Spearman's footrule** distance $F(R_1, R_2)$, which adds up the distance between the ranks of the same item in the two rankings
- Finding an exact solution is
 - computationally hard for Kendall tau (NP-complete)
 - tractable for Spearman's footrule (PTIME)
- These distances are related:
$$K(R_1, R_2) \leq F(R_1, R_2) \leq 2 K(R_1, R_2)$$
 - and $F(R_1, R_2)$ admits efficient approximations (e.g., **median ranking**)



Sir Maurice George Kendall



Charles Edward Spearman

Combining opaque rankings
(i.e., rankings with no visible scores)

Combining opaque rankings

[Fagin, Kuvar, Sivakumar, SIGMOD 2003]

- Techniques using only the **position** of the elements in the ranking (a.k.a. **ranked list**)
 - no other associated score
- We review **MedRank**
 - Based on the notion of **median**, it provides a(n approximation of) Footrule-optimal aggregation

Input: integer k , ranked lists R_1, \dots, R_m of N elements

Output: the top k elements according to median ranking

1. Use **sorted accesses** in each list, one element at a time, until there are k elements that occur in more than $m/2$ lists
2. These are the top k elements

Idea: descend in all ranking from top to bottom until we find k elements that occur in more than $m/2$ lists

MedRank example: hotels in Paris

- Three rankings
 - By price, by rating, and by distance
 - Looking for the top 3 hotels by “median rank”
 - Strategy:
 - Make one sorted access at a time in each ranking
 - Look for hotels that appear in at least 2 rankings
- NB: price, rating and distance are opaque, only the position matters

The three ranks

price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	

Top 3 hotels	Median rank

price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	

Top 3 hotels	Median rank

price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	

Top 3 hotels	Median rank

price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	



Novotel appears in more than half rankings (2 out of 3)

Top 3 hotels	Median rank
Novotel	$\text{median}\{2,3,?\}=3$



From just that information that no other hotels will have a lower median

price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	



Top 3 hotels	Median rank
Novotel	$\text{median}\{2,3,?\}=3$

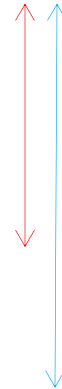
price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	



Top 3 hotels	Median rank
Novotel	$\text{median}\{2,3,5\}=3$
Hilton	$\text{median}\{4,5,?\}=5$
Ibis	$\text{median}\{1,5,?\}=5$

- The (maximum) number of sorted accesses made on each list is also called the **depth** reached by the algorithm --> number of rows we had to access in order to solve the problem
- Here, the depth is 5

price	rating	distance
Ibis	Crillon	Le Roch
Etap	Novotel	Lodge In
Novotel	Sheraton	Ritz
Mercure	Hilton	Lutetia
Hilton	Ibis	Novotel
Sheraton	Ritz	Sheraton
Crillon	Lutetia	Mercure
...	...	



Top 3 hotels	Median rank
Novotel	$\text{median}\{2,3,5\}=3$
Hilton	$\text{median}\{4,5,?\}=5$
Ibis	$\text{median}\{1,5,?\}=5$

When the median ranks are all distinct (unlike here), we have the Footrule-optimal aggregation

Optimality of MedRank

- An algorithm is **optimal** if its execution cost is never worse than any other algorithm on any input (can't be beaten by other algorithms in efficiency)
- MedRank is **not optimal**
- However, it is **instance-optimal**
 - Among the algorithms that access the lists in sorted order, this is the **best possible algorithm** (up to a constant factor) on every input instance
 - In other words, its cost cannot be arbitrarily worse than any other algorithm on any problem instance

Definition of instance optimality

- A form of optimality aimed at when standard optimality is unachievable
 - Let \mathcal{A} be a family of algorithms, \mathcal{I} a set of problem instances
 - Let $cost$ be a cost metric applied to an algorithm-instance pair
 - Algorithm A^* is **instance-optimal** wrt. \mathcal{A} and \mathcal{I} for the cost metric $cost$ if there exist constants k_1 and k_2 such that, for all $A \in \mathcal{A}$ and $I \in \mathcal{I}$,
$$cost(A^*, I) \leq k_1 \cdot cost(A, I) + k_2$$
- If A^* is instance-optimal, then any algorithm can improve wrt A^* by only a constant factor r , which is therefore called the **optimality ratio** of A^*
- Instance optimality is a much stronger notion than optimality in the average or worst case!
 - E.g., binary search is worst-case optimal, but not instance optimal
 - For each instance, a positive answer can be obtained in 1 probe, a negative answer in 2 probes ($\ll \log N$)
- Example: MedRank
 - Optimality ratio = 2, additive constant = m
 - If we want to get rid of dependence on m , we get optimality ratio = 4

Ranking queries
(a.k.a. top-k queries)

Top- k queries

Aim: to retrieve only the k best answers from a potentially (very) large result set

- Best = most important/interesting/relevant/...
- Useful in a variety of scenarios, such as e-commerce, scientific DB's, Web search, multimedia systems, etc.
- Needed: ability to “rank” objects (1st best, 2nd best, ...)

Ranking = ordering objects based on their “relevance”

Top-k queries: naïve approach (1)

- Assume a *scoring function* S that assigns to each tuple t a numerical score for ranking tuples
 - E.g. $S(t) = t.Points + t.Rebounds$
- Straightforward way to compute the result

Name	Points	Rebounds	...
Shaquille O'Neal	1669	760	...
Tracy McGrady	2003	484	...
Kobe Bryant	1819	392	...
Yao Ming	1465	669	...
Dwyane Wade	1854	397	...
Steve Nash	1165	249	...
...

Input: cardinality k , dataset R , scoring function S

Output: the k highest-scored tuples wrt S

1. for all tuples t in R
 - 1.1. compute $S(t)$
2. sort tuples based on their scores
3. return the first k highest-scored tuples

complexity: $n \cdot n \log(n)$
scan of all tuple* sorting

In the context of databases each time you have a complexity which is more than linear, it is considered bad

Top-k queries: naïve approach (2)

- Naïve approach expensive for large datasets
 - requires **sorting** a large amount of data
- Even worse if more than one relation is involved

$$S(t) = t.Points + t.Rebounds$$

- Need to join all tuples, which is also costly
- Here the join is **1-1**, but in general it can be **M-N** (each tuple can join with an arbitrary number of tuples)

Name	Points	...
Shaquille O'Neal	1669	...
Tracy McGrady	2003	...
Kobe Bryant	1819	...
Yao Ming	1465	...
Dwyane Wade	1854	...
Steve Nash	1165	...
...

Name	Rebounds	...
Shaquille O'Neal	760	...
Tracy McGrady	484	...
Kobe Bryant	392	...
Yao Ming	669	...
Dwyane Wade	397	...
Steve Nash	249	...
...

Top-k queries in SQL

- Two abilities required:
 - 1) **Ordering** the tuples according to their scores
 - 2) **Limiting** the output cardinality to k tuples
- Ordering is taken care of by `ORDER BY`
- Limiting was not native in SQL until 2008
`FETCH FIRST k ROWS ONLY (SQL:2008)`
 - Supported by IBM DB2, PostgreSQL, Oracle, MS SQL Server, ...
- Many non-standard syntaxes available:
`ROWNUM <= k (ORACLE)`
`LIMIT k (PostgreSQL, MySQL)`
`SELECT TOP k FROM (MS SQL Server)`
...

Shortcomings of ORDER BY

- Consider the following queries:

A) `SELECT *
FROM UsedCarTable
WHERE Vehicle = 'Audi/A4'
AND Price <= 21000
ORDER BY 0.8*Price+0.2*Miles`

B) `SELECT *
FROM UsedCarTable
WHERE Vehicle = 'Audi/A4'
ORDER BY 0.8*Price+0.2*Miles`

The values 0.8 and 0.2, also called **weights**, are a way to normalize our preferences on Price and Mileage

- Query A will likely miss some relevant answers (*near-miss*)
—e.g., a car with a price of \$21,500 but very low mileage
- Query B will return all Audi/A4 in the dataset (*information overload*)

Semantics of top-k queries

- Only the first k tuples become part of the result
- If more than one set of k tuples satisfies the ORDER BY directive, any of such sets is a valid answer (non-deterministic semantics)

```
SELECT *  
FROM R  
ORDER BY Price  
FETCH FIRST 3  
ROWS ONLY
```

R

<u>OBJ</u>	Price
t15	50
t24	40
t26	30
t14	30
t21	40

<u>OBJ</u>	Price
t26	30
t14	30
t21	40

<u>OBJ</u>	Price
t26	30
t14	30
t24	40

Both are valid results

Top-k queries: examples

- The best NBA player (by points and rebounds):

```
SELECT *  
FROM NBA  
ORDER BY Points + Rebounds DESC  
FETCH FIRST 1 ROW ONLY
```

- The 2 cheapest Chinese restaurants

```
SELECT *  
FROM RESTAURANTS  
WHERE Cuisine = 'Chinese'  
ORDER BY Price  
FETCH FIRST 2 ROWS ONLY
```

- The top-5 Audi/A4 (by price and mileage)

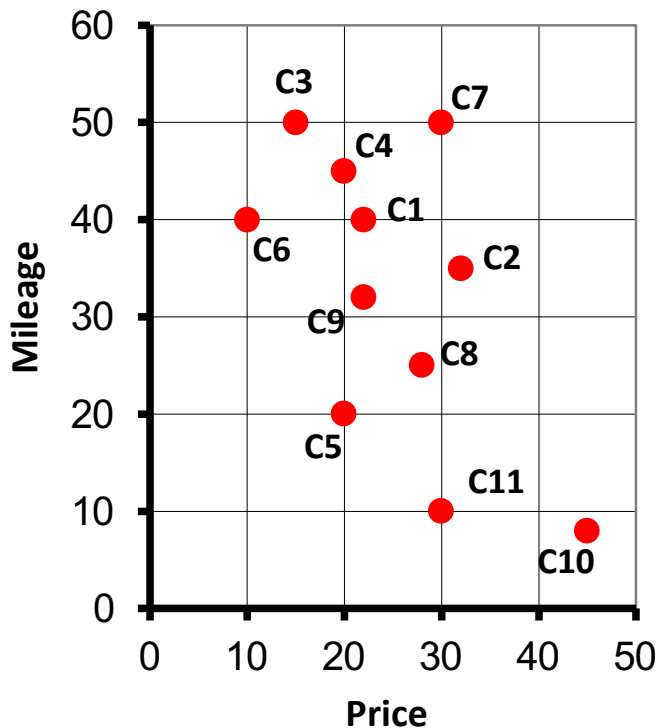
```
SELECT *  
FROM USEDCARS  
WHERE Vehicle = 'Audi/A4'  
ORDER BY 0.8*Price + 0.2*Miles  
FETCH FIRST 5 ROWS ONLY
```

Evaluation of top-k queries

- Two basic aspects to consider:
 - query type: 1 relation, many relations, aggregate results, ...
 - access paths: no index, indexes on all/some ranking attributes
- Simplest case: top-k selection query with only 1 relation:
 - If input **sorted** according to **S**: read only first **k** tuples
 - No need to scan the entire input
 - Tuples **not sorted**: if **k** is not too large (which is the typical case), perform in-memory sort (through a *heap*)
 - The entire input needs to be read (cost $O(N \log k)$)
 - Can be improved to $O(N + k \log N)$ or even $O(N + k \log k)$

A geometric view

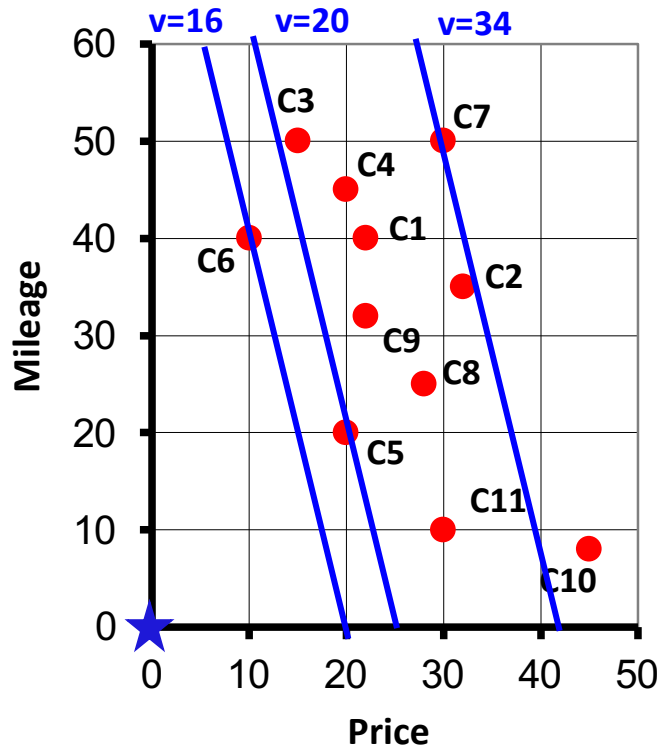
- Consider the 2-dimensional *attribute space* (Price, Mileage)



- Each tuple is represented by a 2-dimensional point (p, m) :
 - p is the Price value
 - m is the Mileage value
- Intuitively, minimizing $0.8 * \text{Price} + 0.2 * \text{Mileage}$ is equivalent to looking for points “close” to $(0, 0)$
- $(0, 0)$ is **our** (ideal) “target value” (i.e., a free car with 0 km’s!)

The role of weights (preferences)

- Our preferences (e.g., 0.8 and 0.2) are essential to determine the result



- Consider the line of equation

$$0.8 * \text{Price} + 0.2 * \text{Mileage} = v$$

where v is a constant

- This can also be written as

$$\text{Mileage} = -4 * \text{Price} + 5 * v$$

from which we see that all the lines have a slope = -4

- By definition, all the points of the same line are “equally good”

the best car is the first one I encounter if I move orthogonally towards the best

- With preferences (0.8,0.2) the best car is C6, then C5, etc.
- In general, preferences are a way to determine, given points (p_1, m_1) and (p_2, m_2) , which of them is “closer” to the target point (0,0)

Changing the weights

- Changing the weight values will likely lead to a different result

■ With

$$0.8 * \text{Price} + 0.2 * \text{Mileage}$$

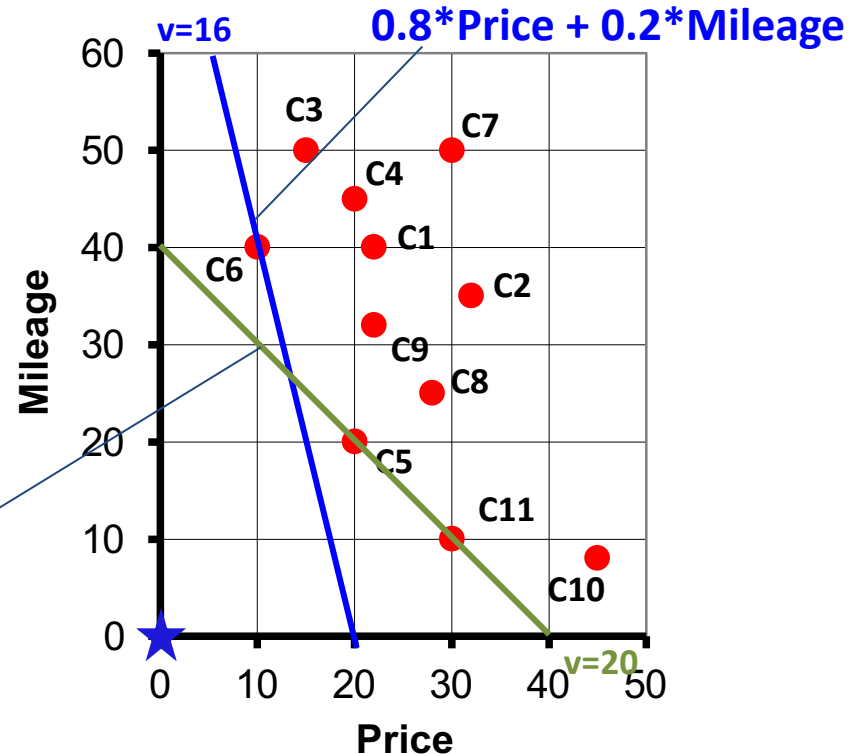
the best car is C6

■ With

$$0.5 * \text{Price} + 0.5 * \text{Mileage}$$

the best cars are C5 and C11

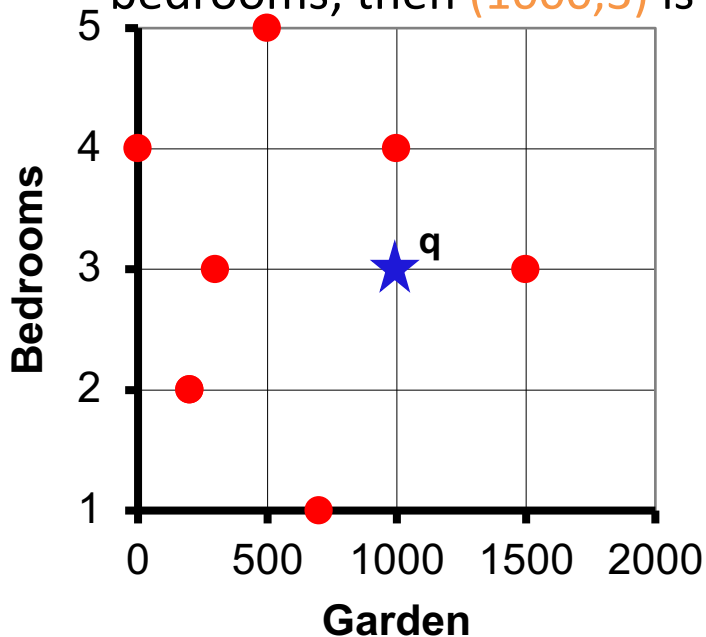
$$0.5 * \text{Price} + 0.5 * \text{Mileage}$$



- On the other hand, if weights do not change too much, the results of two top-k queries will likely have a high degree of overlap

Changing the target

- The target of a query is not necessarily $(0,0)$, rather it can be any point $q=(q_1,q_2)$ (q_i = query value for the i -th attribute)
- Example: assume you are looking for a house with a 1000 m² garden and 3 bedrooms; then $(1000,3)$ is the target for your query



■ In general, in order to determine the “goodness” of a tuple t , we compute its “distance” from the target point q :

The lower the distance from q , the better t is

Note that distance values can always be converted into goodness “scores”, so that a higher score means a better match
Just change the sign and possibly add a constant,...

Top-k tuples = k-nearest neighbors

- It is sometimes useful to consider **distances** rather than scores
 - since most indexes are “distance-based”
- Therefore, the model is now:
 - An m -dimensional ($m \geq 1$) space $\mathbf{A} = (A_1, A_2, \dots, A_m)$ of ranking attributes
 - A relation $R(A_1, A_2, \dots, A_m, B_1, B_2, \dots)$, where B_1, B_2, \dots are other attributes
 - A target (query) point $\mathbf{q} = (q_1, q_2, \dots, q_m)$, $\mathbf{q} \in \mathbf{A}$
 - A function $d: \mathbf{A} \times \mathbf{A} \rightarrow \mathbb{R}^+$, measuring the distance between points in \mathbf{A} (e.g., $d(\mathbf{t}, \mathbf{q})$ is the distance between \mathbf{t} and \mathbf{q})
- Under this model, a top-k query is transformed into a so-called

k-Nearest Neighbors (k-NN) Query

- Given a point \mathbf{q} , a relation R , an integer $k \geq 1$, and a distance function d
- Determine the k tuples in R that are closest to \mathbf{q} according to d

Some common distance functions

- The most commonly used distance functions are L_p -norms (Minkowski distance):

$$L_p(t, q) = \left(\sum_{i=1}^m |t_i - q_i|^p \right)^{1/p}$$

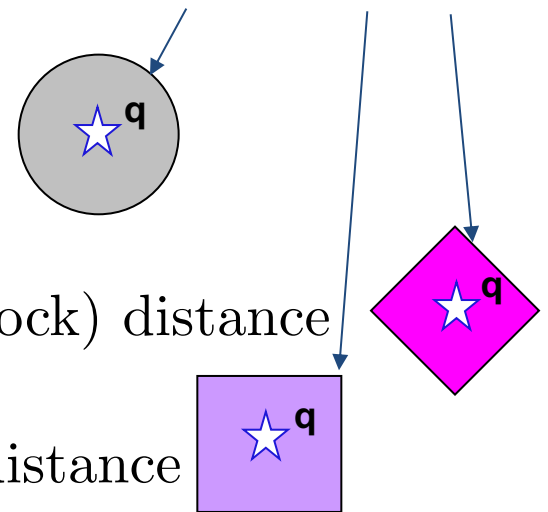
- Relevant cases are:

$$L_2(t, q) = \sqrt{\sum_{i=1}^m |t_i - q_i|^2} \quad \text{Euclidean distance}$$

$$L_1(t, q) = \sum_{i=1}^m |t_i - q_i| \quad \text{Manhattan (city-block) distance}$$

$$L_\infty(t, q) = \max_i \{|t_i - q_i|\} \quad \text{Chebyshev (max) distance}$$

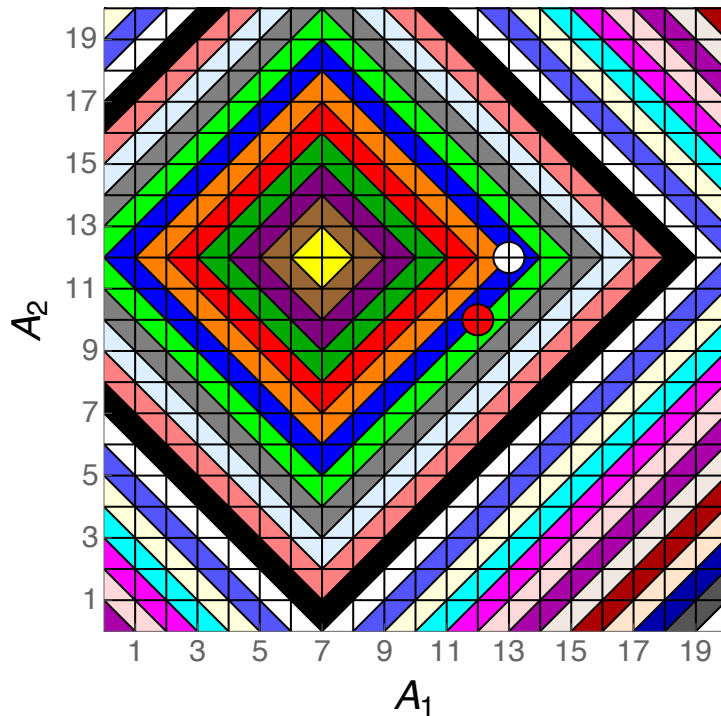
Iso-distance (hyper-)surfaces



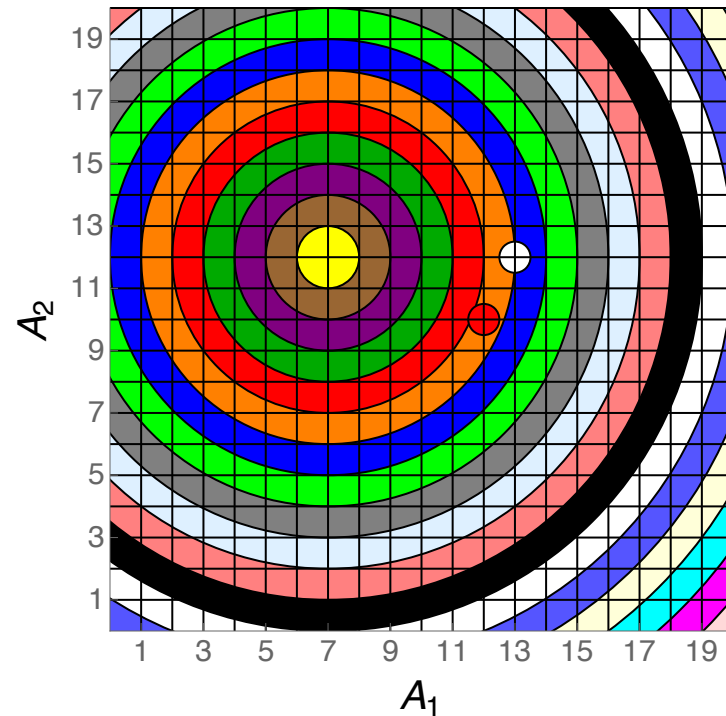
Shaping the attribute space

- Changing the distance changes the shape of the attribute space
 - each “stripe” corresponds to points with distance values between v and $v+1$, v integer

$L_1; q=(7,12)$



$L_2; q=(7,12)$

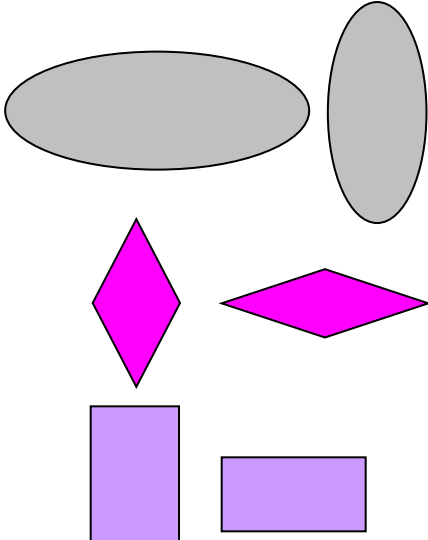


Note that, for 2 tuples t_1 and t_2 , it is possible to have
 $L_1(t_1, q) < L_1(t_2, q)$ and $L_2(t_2, q) < L_2(t_1, q)$

E.g.: $t_1=(13,12)$ ○
 $t_2=(12,10)$ ●

Distance functions with weights

- The use of weights $W=(w_1, \dots, w_m)$ entails “stretching” some of the coordinates:

$$L_2(t, q; W) = \sqrt{\sum_{i=1}^m w_i |t_i - q_i|^2}$$
$$L_1(t, q; W) = \sum_{i=1}^m w_i |t_i - q_i|$$
$$L_\infty(t, q; W) = \max_i \{w_i |t_i - q_i|\}$$


(hyper-)ellipsoids

(hyper-)rhomboids

(hyper-)rectangles

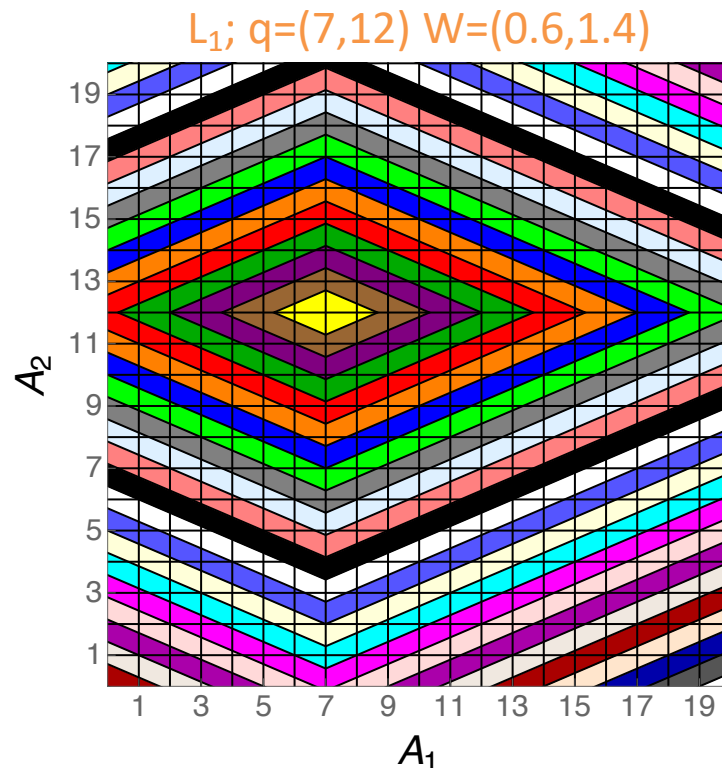
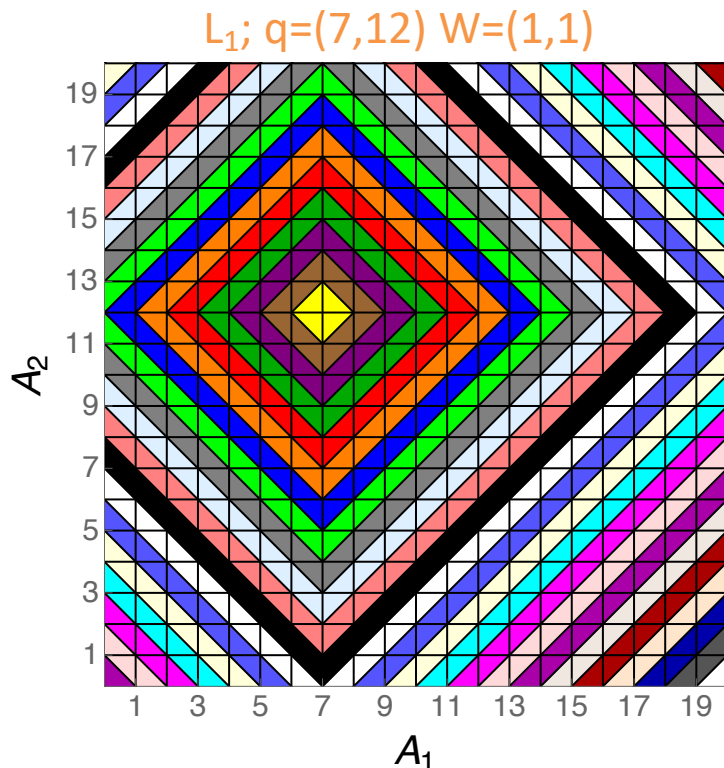
- Thus, the scoring function

$$0.8 * \text{Price} + 0.2 * \text{Mileage}$$

is just a particular case of weighted L_1 distance

Weights the attribute space

- The figures show the effects of using L_1 with different weights



- If $w_2 > w_1$, then the hyper-rhomboids are more elongated along A_1
 - i.e., difference on A_1 values is less important than an equal difference on A_2 values

Top-k join queries

- In a **top-k join query** we have $n > 1$ input relations and a scoring function S defined on the result of the join, i.e.:

```
SELECT      <some attributes>
FROM         $R_1, R_2, \dots, R_n$ 
WHERE       <join and local conditions>
ORDER BY     $S(p_1, p_2, \dots, p_m)$  [DESC]
FETCH FIRST  $k$  ROWS ONLY
```

where p_1, p_2, \dots, p_m are scoring criteria
– the “preferences”

Top-k join queries: examples

- The highest paid employee (compared to her dept budget):

```
SELECT E.*  
FROM EMP E, DEPT D  
WHERE E.DNO = D.DNO  
ORDER BY E.Salary / D.Budget DESC  
FETCH FIRST 1 ROW ONLY
```

- The 2 cheapest restaurant-hotel combinations in the same Italian city

```
SELECT *  
FROM RESTAURANTS R, HOTELS H  
WHERE R.City = H.City  
AND R.Nation = 'Italy' AND H.Nation = 'Italy'  
ORDER BY R.Price + H.Price  
FETCH FIRST 2 ROWS ONLY
```

Top-k 1-1 join queries

- All the joins are on a common key attribute(s)
 - Simplest case to deal with
 - Basis for the more general cases
 - Practically relevant
- Two main scenarios:
 - There is an index for retrieving tuples according to each preference
 - The relation is spread over several sites, each providing information only on part of the objects (the “middleware” scenario)

Top-k 1-1 join queries: assumptions

- Each input list supports **sorted access**:
 - Each access returns the id of the next best object, its partial score p_j (and possibly other attributes)
- Each input list supports **random access**:
 - Each access returns the partial score of an object, given its id (requires index on primary key)
- The id of an object is the same across all inputs (perhaps after reconciliation)
- Each input consists of the same set of objects

Top-k 1-1 join queries: example

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

- Aggregating restaurant reviews

```
SELECT *  
FROM EatWell EW, BreadAndWine BW  
WHERE EW.Name = BW.Name  
ORDER BY EW.Score + BW.Score DESC  
FETCH FIRST 1 ROW ONLY
```

Note: the winner is not the best locally!

Name	Global Score
Cheers!	16.8
The old mill	16.7
Da Gino	16.5
The canteen	16.0
Chez Paul	13.0
Let's eat!	12.4
Los pollos hermanos	11.5

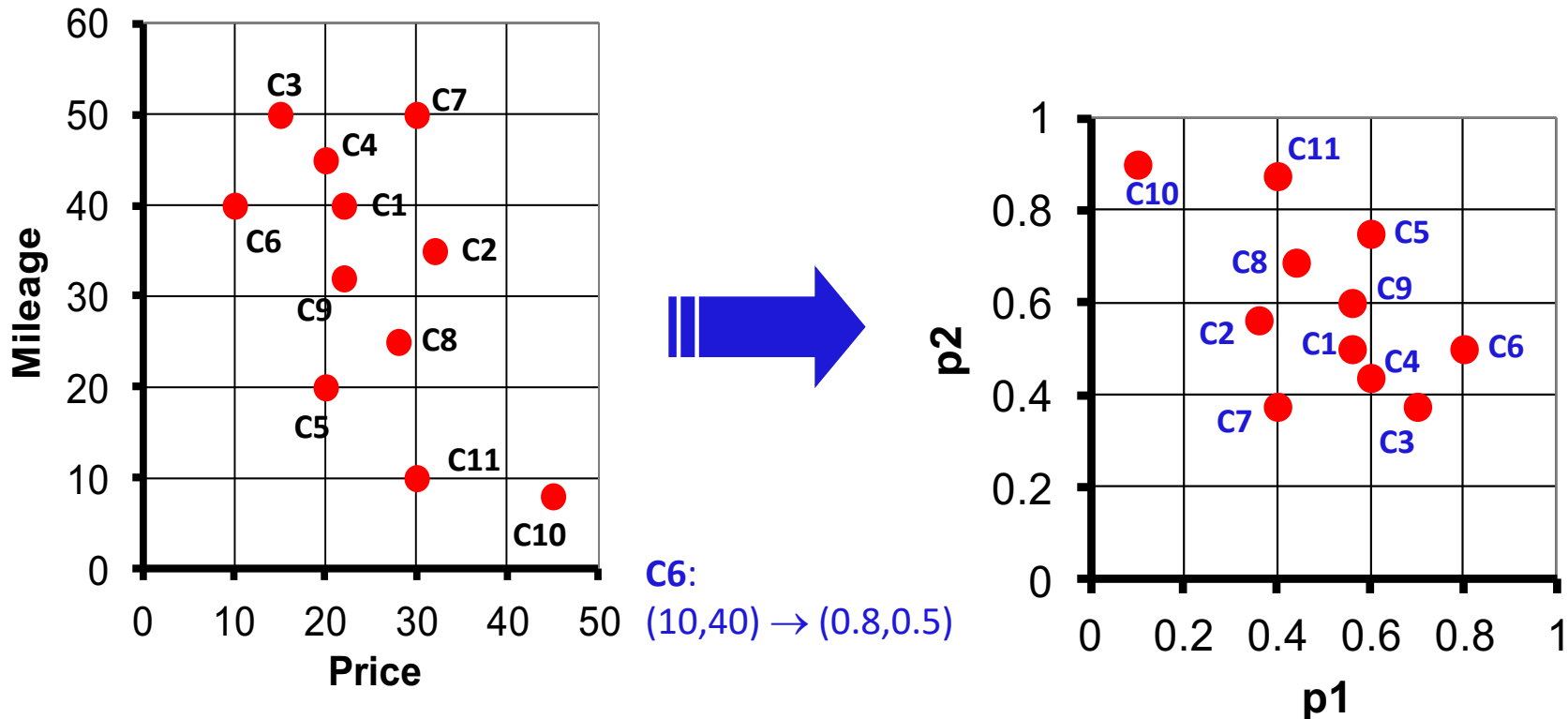
A model for scoring functions

- Each object o returned by the input L_j has an associated local/partial score $p_j(o) \in [0,1]$
 - For convenience, scores are **normalized**
 - higher is better (this is a convention that can be changed!)
 - only the best and the worst possible value of p_j matter
 - The hypercube $[0,1]^m$ is called the **score space**
- The point $p(o) = (p_1(o), p_2(o), \dots, p_m(o)) \in [0,1]^m$ is the map of o into the score space
- The **global score** $S(o)$ of o is computed by means of a **scoring function** S that combines in some way the local scores of o :

$$S : [0,1]^m \rightarrow \mathbb{R}^+ \quad S(o) \equiv S(p(o)) = S(p_1(o), p_2(o), \dots, p_m(o))$$

The score space

- Consider the attribute space $\mathbf{A} = (\text{Price}, \text{Mileage})$
- Let: $p_1(o) = 1 - o.\text{Price}/\text{MaxP}$, $p_2(o) = 1 - o.\text{Mileage}/\text{MaxM}$
- Let's take $\text{MaxP} = 50,000$ and $\text{MaxM} = 80,000$
- Objects in \mathbf{A} are mapped into the score space as shown
 - The relative order on each coordinate (local ranking) remains unchanged



Common scoring functions

SUM (AVG): used to weigh preferences equally

$$\text{SUM}(o) \equiv \text{SUM}(p(o)) = p_1(o) + p_2(o) + \dots + p_m(o)$$

WSUM (Weighted sum): to weigh the ranking attributes differently

$$\text{WSUM}(o) \equiv \text{WSUM}(p(o)) = w_1 * p_1(o) + w_2 * p_2(o) + \dots + w_m * p_m(o)$$

MIN (Minimum): just considers the worst partial score

$$\text{MIN}(o) \equiv \text{MIN}(p(o)) = \min\{p_1(o), p_2(o), \dots, p_m(o)\}$$

es: the object with the highest minimum value

MAX (Maximum): just considers the best partial score

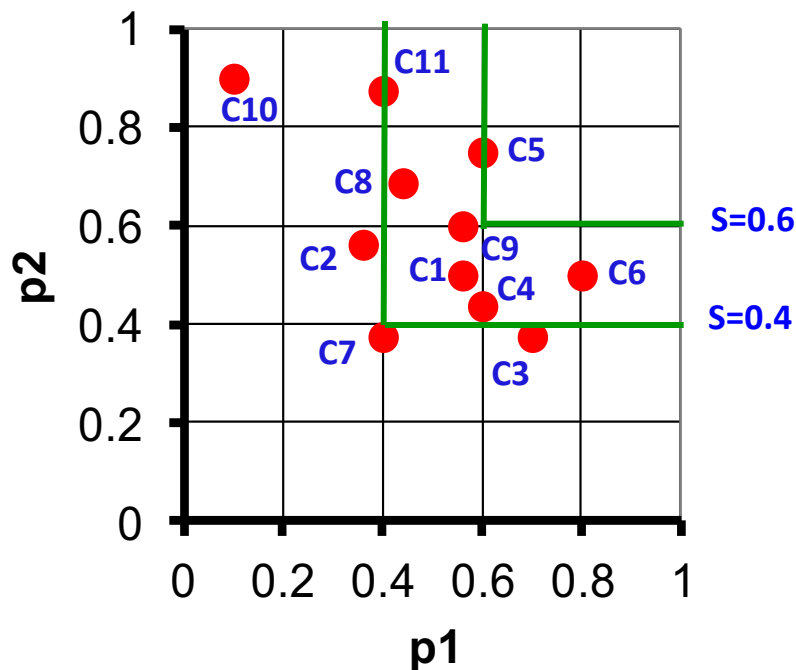
$$\text{MAX}(o) \equiv \text{MAX}(p(o)) = \max\{p_1(o), p_2(o), \dots, p_m(o)\}$$

(even with MIN) we always want to retrieve the k objects with the highest global scores

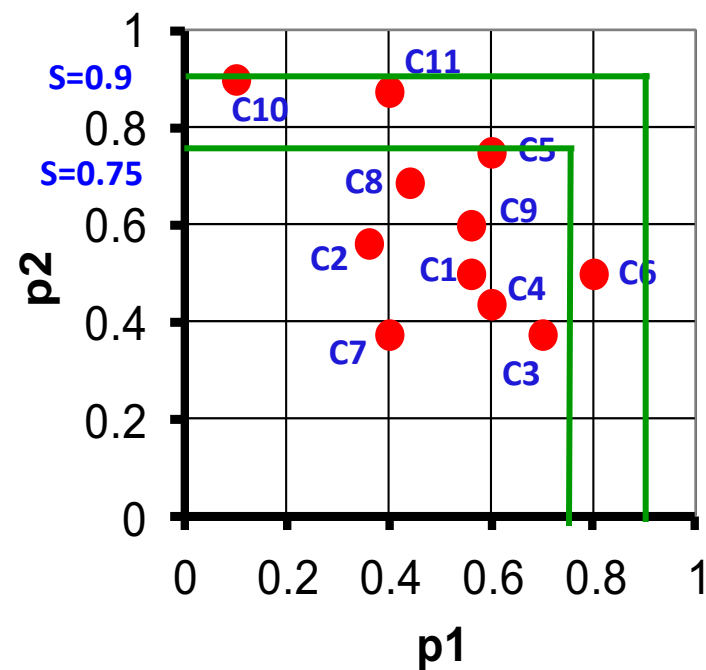
Equally scored objects

- We can define iso-score curves in the score space
 - Sets of points with the same global score
 - Similar to iso-distance curves in an attribute space

$$S(o) = \text{MIN}(p(o))$$



$$S(o) = \text{MAX}(p(o))$$



The simplest case: MAX

[Fagin, PODS 1996]

- We are now ready to ask “the big question”:

How can we efficiently compute the result of a top-k 1-1 join query using a scoring function S ?

- When $S \equiv \text{MAX}$, it is really simple:

You can use my algorithm B_0 ,
which just retrieves the best k objects from
each source, that's all!

Beware! B_0 only works for MAX .
Other scoring functions require
smarter, and more costly, algorithms



Ronald Fagin

The B_0 algorithm

Assumes rankings are sorted based on their local function

Input: integer $k \geq 1$, ranked lists R_1, \dots, R_m

Output: the top- k objects according to the **MAX** scoring function

1. Make k sorted accesses on each list and store objects and partial scores in a buffer **B**
2. For each object in **B**, compute the **MAX** of its (available) partial scores
3. Return the k objects with maximum score

- No need to obtain missing partial scores
- No random accesses are executed
- B_0 is **instance-optimal**

B_0 : examples

$k = 2$

OID	p1
o7	0.7
o3	0.65
o4	0.6
o2	0.5

OID	p2
o2	0.9
o3	0.6
o7	0.4
o4	0.2

OID	p3
o7	1.0
o2	0.8
o4	0.75
o3	0.7

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

B_0 : examples

$k = 2$

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

k = 3

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

k = 3

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

k = 3

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

k = 3

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

k = 3

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

Name	S
The old mill	9.2
Da Gino	9.0
The canteen	9.0
Cheers!	8.5

B₀: examples

k = 2

OID	p1		OID	p2		OID	p3
o7	0.7		o2	0.9		o7	1.0
o3	0.65		o3	0.6		o2	0.8
o4	0.6		o7	0.4		o4	0.75
o2	0.5		o4	0.2		o3	0.7

OID	S
o7	1.0
o2	0.9
o3	0.65

k = 3

EatWell

Name	Score
The old mill	9.2
The canteen	9.0
Cheers!	8.3
Da Gino	7.5
Let's eat!	6.4
Chez Paul	5.5
Los pollos hermanos	5.0

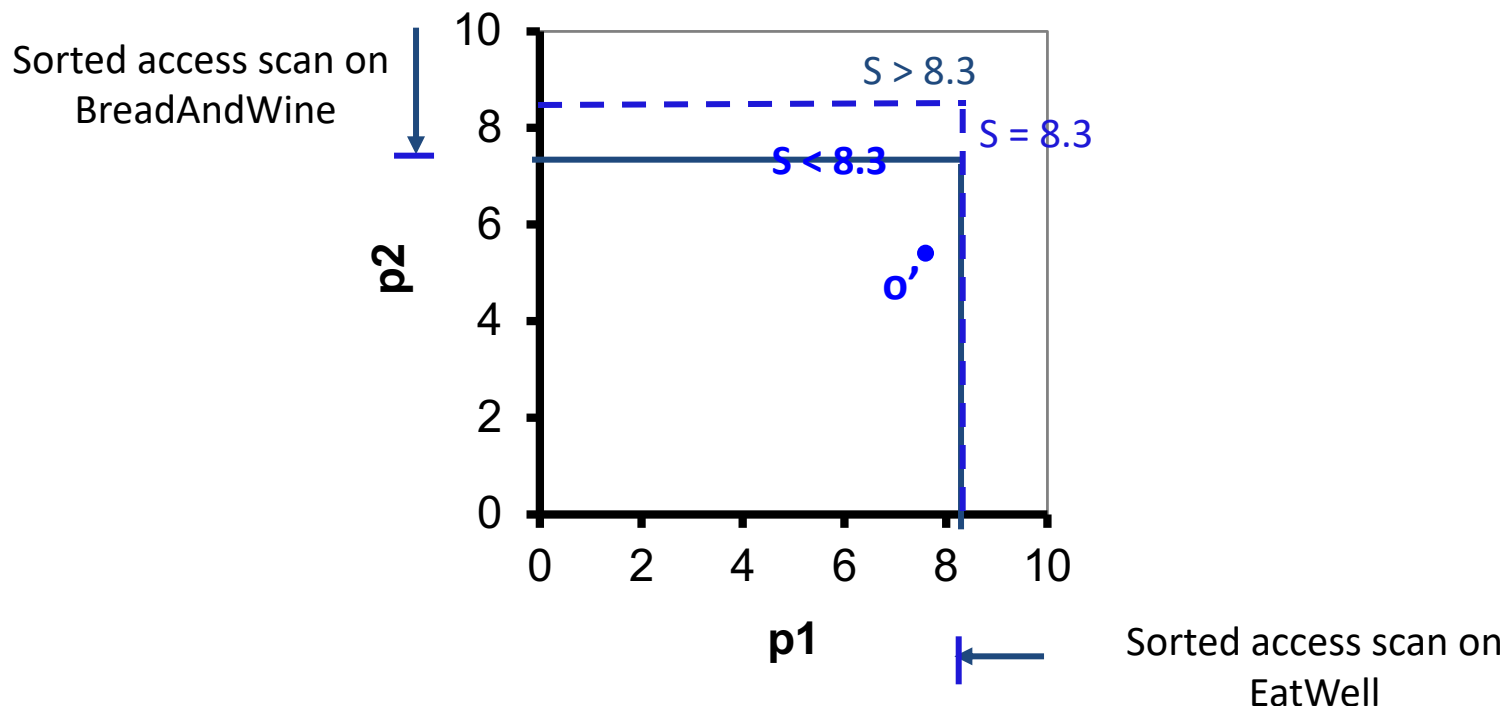
BreadAndWine

Name	Score
Da Gino	9.0
Cheers!	8.5
The old mill	7.5
Chez Paul	7.5
The canteen	7.0
Los pollos hermanos	6.5
Let's eat!	6.0

Name	S
The old mill	9.2
Da Gino	9.0
The canteen	9.0
Cheers!	8.5

Why B_0 works (restaurants example)

- After 3 sorted access rounds it is guaranteed that there are at least 3 restaurants o with $S(o) \geq 8.3$
- A restaurant like o' , that has not been retrieved by any sorted access scan cannot have a global score higher than 8.3



B_0 doesn't work when $S \neq \text{MAX}$

- Let $S = \text{MIN}$ and $k = 1$

OID	p1
o7	0.9
o3	0.65
o2	0.6
o1	0.5
o4	0.4

OID	p2
o2	0.95
o3	0.7
o4	0.6
o1	0.5
o7	0.5

OID	p3
o7	1.0
o2	0.8
o4	0.75
o3	0.7
o1	0.6

B_0 doesn't work when $S \neq \text{MAX}$

- Let $S = \text{MIN}$ and $k = 1$

OID	p1		OID	p2		OID	p3
o7	0.9		o2	0.95		o7	1.0
o3	0.65		o3	0.7		o2	0.8
o2	0.6		o4	0.6		o4	0.75
o1	0.5		o1	0.5		o3	0.7
o4	0.4		o7	0.5		o1	0.6

B_0 doesn't work when $S \neq \text{MAX}$

- Let $S = \text{MIN}$ and $k = 1$

OID	p1		OID	p2		OID	p3
o7	0.9		o2	0.95		o7	1.0
o3	0.65		o3	0.7		o2	0.8
o2	0.6		o4	0.6		o4	0.75
o1	0.5		o1	0.5		o3	0.7
o4	0.4		o7	0.5		o1	0.6

OID	S
o2	0.95
o7	0.9

WRONG!!

B_0 doesn't work when $S \neq \text{MAX}$

- Let $S = \text{MIN}$ and $k = 1$

OID	p1		OID	p2		OID	p3
o7	0.9		o2	0.95		o7	1.0
o3	0.65		o3	0.7		o2	0.8
o2	0.6		o4	0.6		o4	0.75
o1	0.5		o1	0.5		o3	0.7
o4	0.4		o7	0.5		o1	0.6

OID	S
o2	0.95
o7	0.9

WRONG!!

- What if we consider **all** the partial scores of the seen objects ($\{o2, o7\}$ in the figure)?
- After performing the necessary **random accesses** we get:

B_0 doesn't work when $S \neq \text{MAX}$

- Let $S = \text{MIN}$ and $k = 1$

OID	p1	OID	p2	OID	p3
o7	0.9	o2	0.95	o7	1.0
o3	0.65	o3	0.7	o2	0.8
o2	0.6	o4	0.6	o4	0.75
o1	0.5	o1	0.5	o3	0.7
o4	0.4	o7	0.5	o1	0.6

OID	S
o2	0.95
o7	0.9

WRONG!!

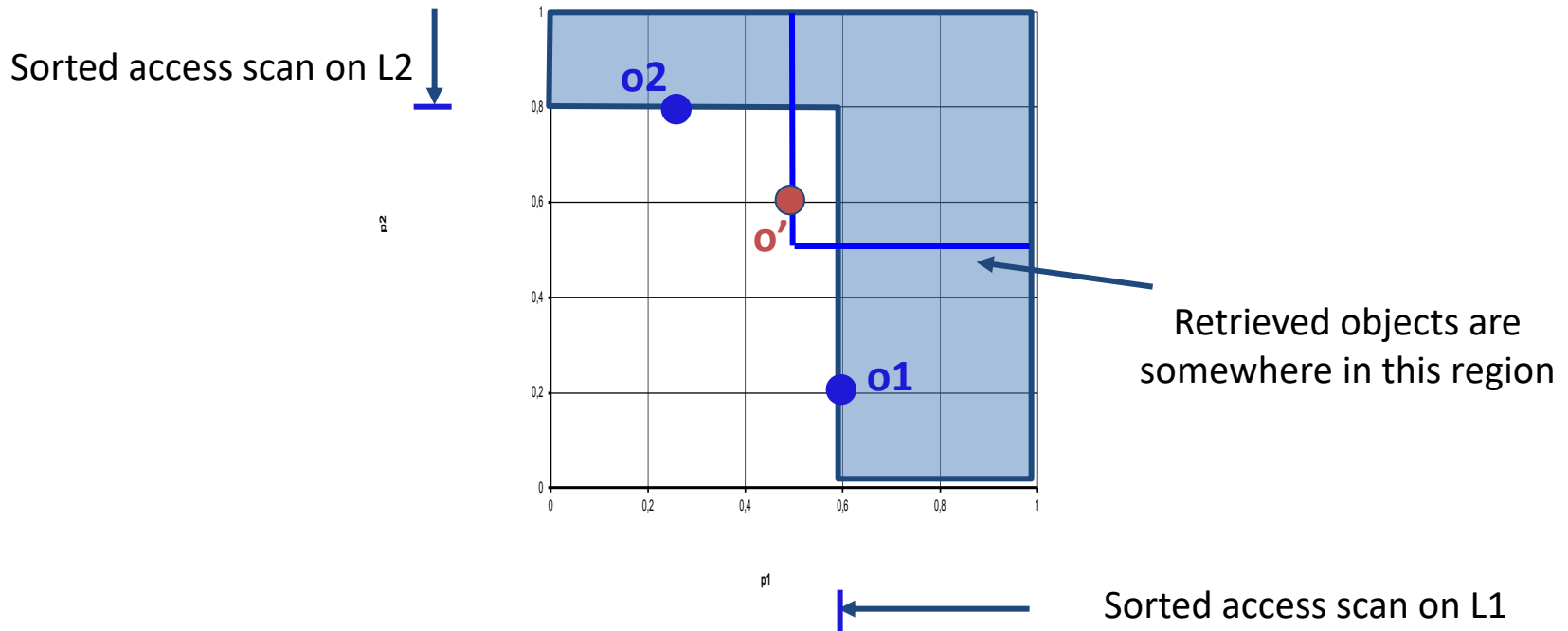
- What if we consider **all** the partial scores of the seen objects ($\{o2, o7\}$ in the figure)?
- After performing the necessary **random accesses** we get:

OID	S
o2	0.6
o7	0.5

STILL WRONG!!? ☹

Why B_0 doesn't work: graphical intuition

- Let $S \equiv \text{MIN}$ and $k = 1$
- When the sorted accesses terminate, we have no lower bound on the global scores of the retrieved objects (i.e., it might also be $S(o) = 0$)
- Any non-retrieved object, like o' , can now be the winner!
 - Note that, in this case, o' would be the best match even for $S \equiv \text{SUM}$



Fagin's Algorithm (FA)

[Fagin, PODS 1998]

Input: integer k , a **monotone function** S combining ranked lists R_1, \dots, R_m

Output: the top k <object, score> pairs

1. Extract the same number of objects by **sorted accesses** in each list until there are at least k objects in common
2. For each extracted object, compute its overall score by making **random accesses** wherever needed
3. Among these, output the k objects with the best overall score

- Complexity is **sub-linear** in the number N of objects
 - Proportional to the square root of N when combining two lists (complexity is $O(N^{(m-1)/m} k^{1/m})$) \rightarrow square root of m
 - The stopping criterion is **independent** of the scoring function
 - **Not instance-optimal**

Example: hotels in Paris with FA

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Assumptions: the rankings are sorted by their attributed in each table

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

Nothing in common, continue...

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

Nothing in common, continue...

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

1 hotel in common, but we need k=2, continue...

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

1 hotel in common, but we need k=2 objects in common, continue...

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

3 hotels in common, we can stop since we have our 2 elements

We descended 5 rows in each ranking

We notice that for some hotels we just have one information, their information must be searched in the rest of the table by performing random access.

for ex: for the "Mercure" we only have cheapness, we need to find its rating. We do that by random access.

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Look for hotels that appear in both lists

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

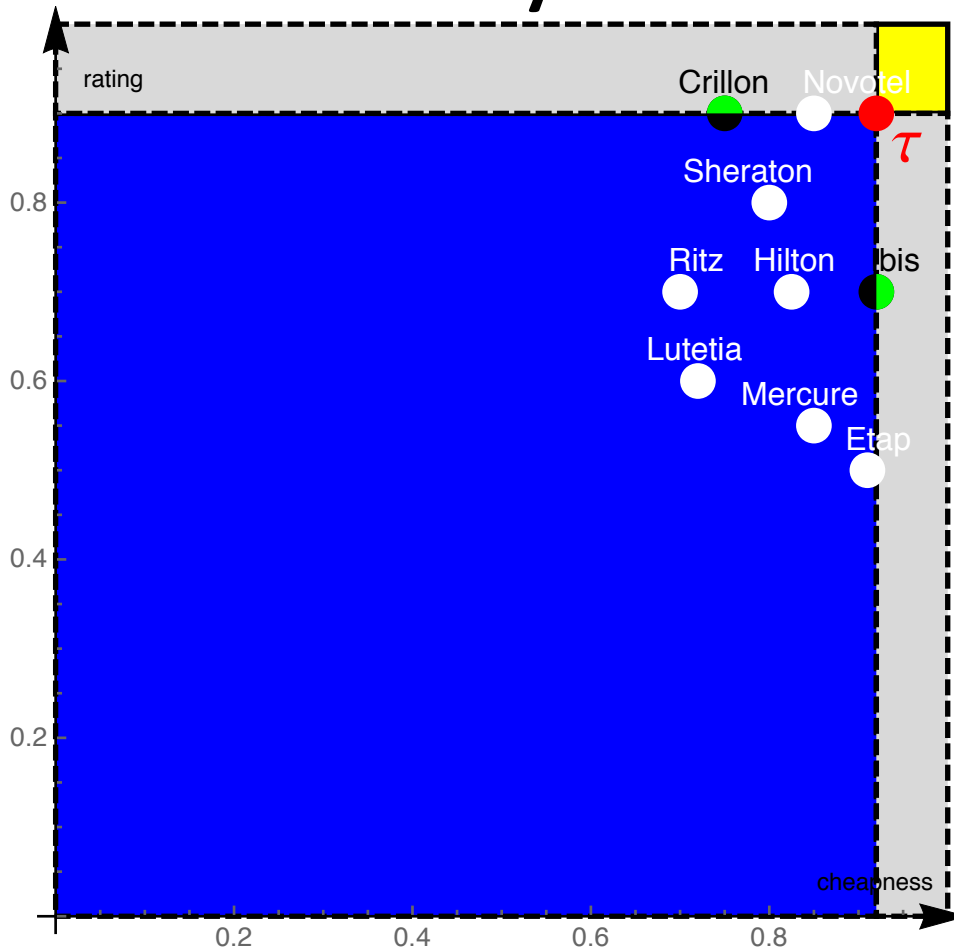
Top 2	Score
Novotel	.875
Crillon	.825

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Now complete the score with random accesses

--> after getting information of every object we can compute the scoring function, and save the best k results

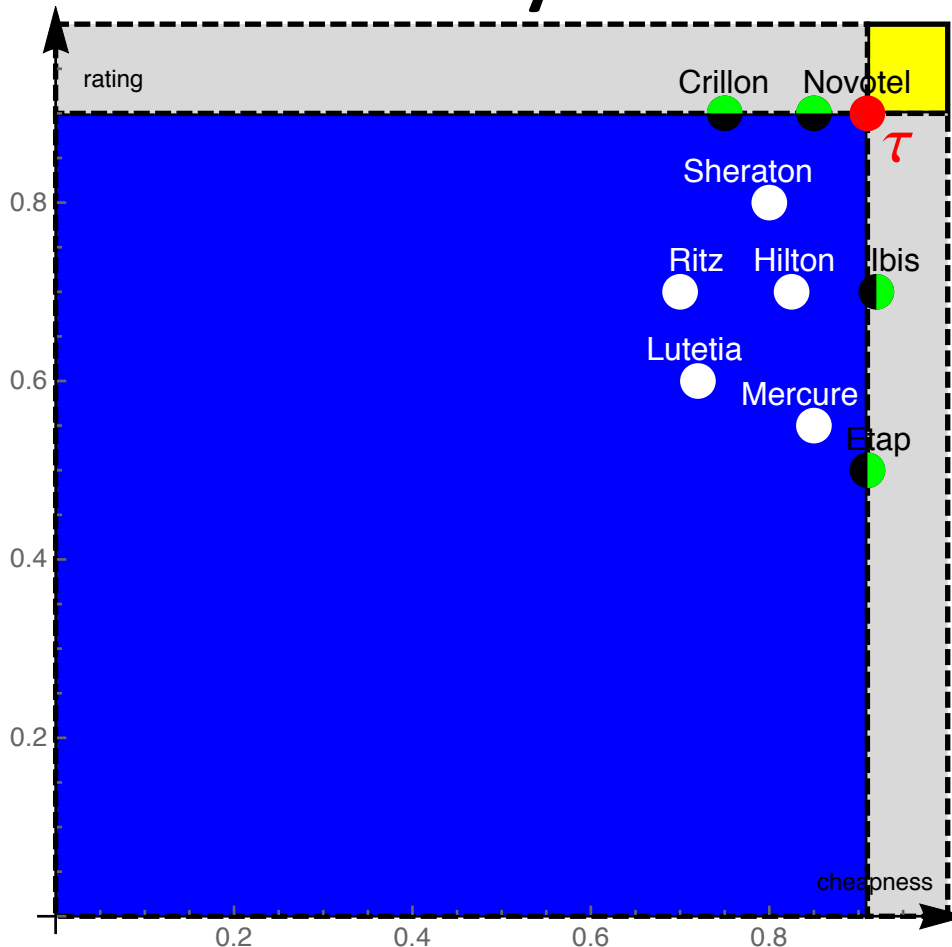
N.B. We didn't exploit in any way the scoring function when doing the sorted access, so that can help us to make the algorithm more efficient

Why does FA work?



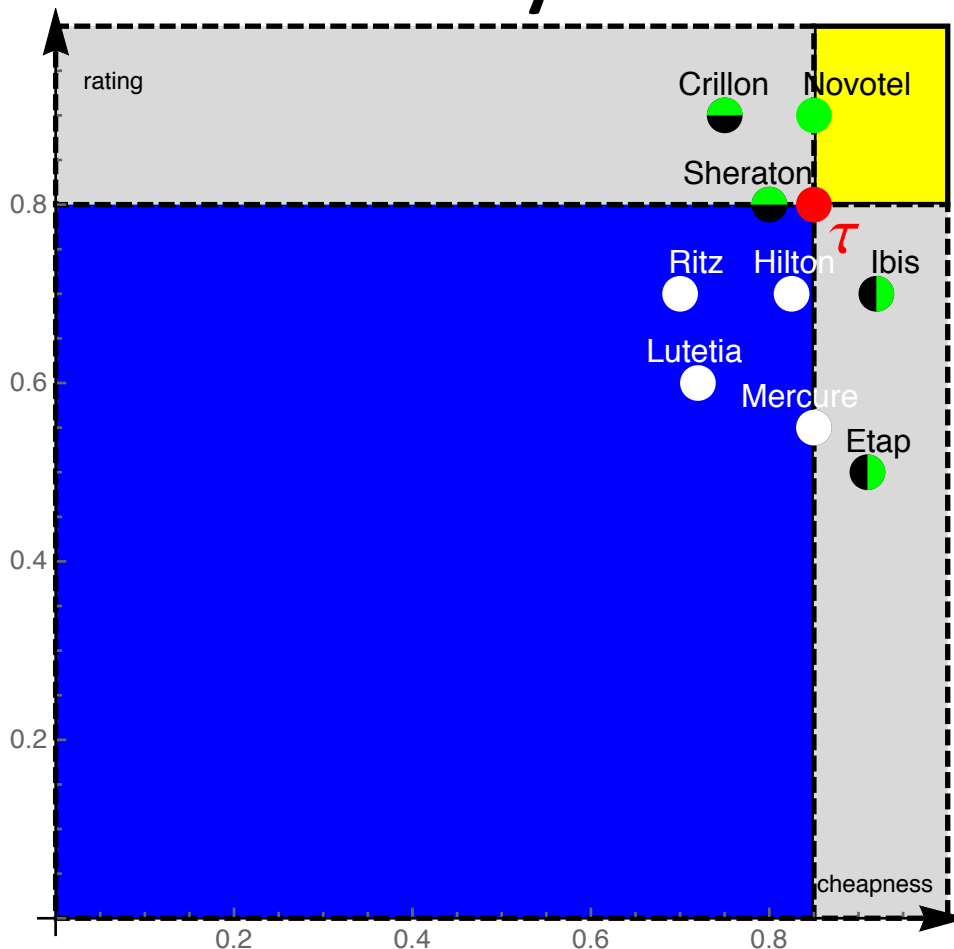
- The **threshold point** (τ) is the point with the smallest seen values on all lists in the sorted access phase
- FA stops when the **yellow region** (fully seen points) contains at least k points
- The **gray regions** contain the points seen in at least one ranking
- None of the points in the **blue region** (unseen points) can beat any point in the **yellow region**

Why does FA work?



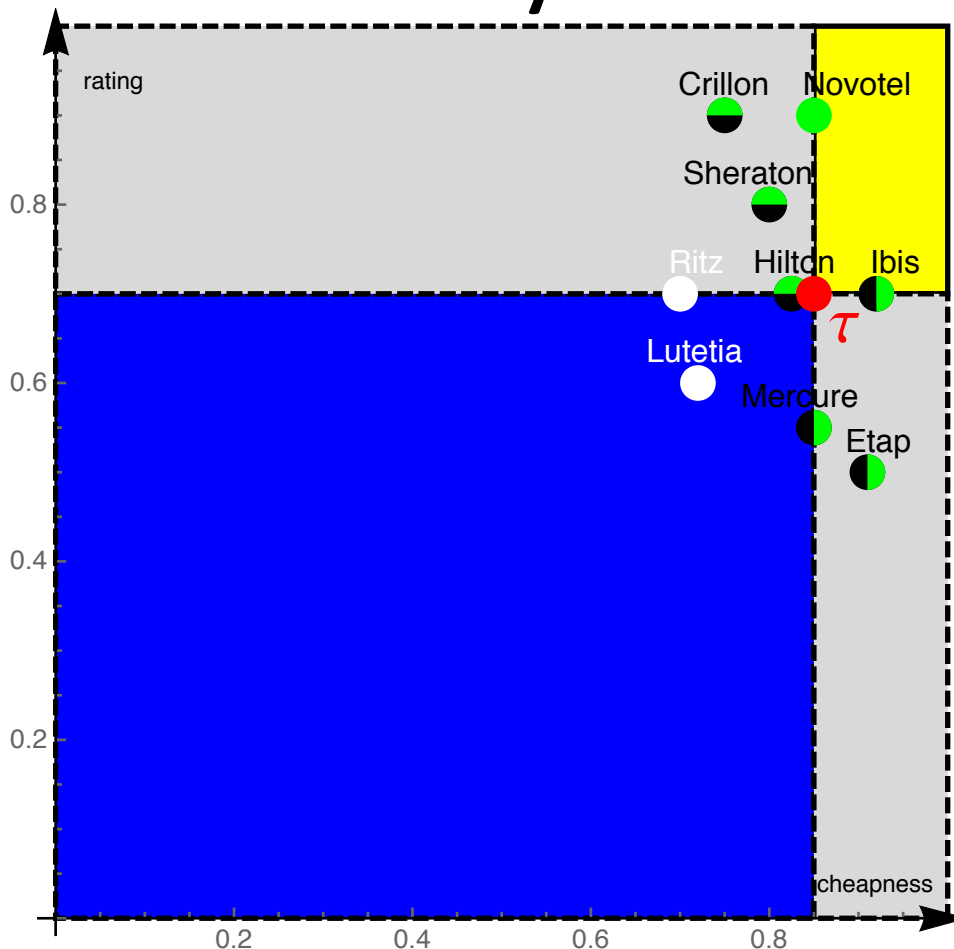
- The **threshold point** (τ) is the point with the smallest seen values on all lists in the sorted access phase
- FA stops when the **yellow region** (fully seen points) contains at least k points
- The **gray regions** contain the points seen in at least one ranking
- None of the points in the **blue region** (unseen points) can beat any point in the **yellow region**

Why does FA work?



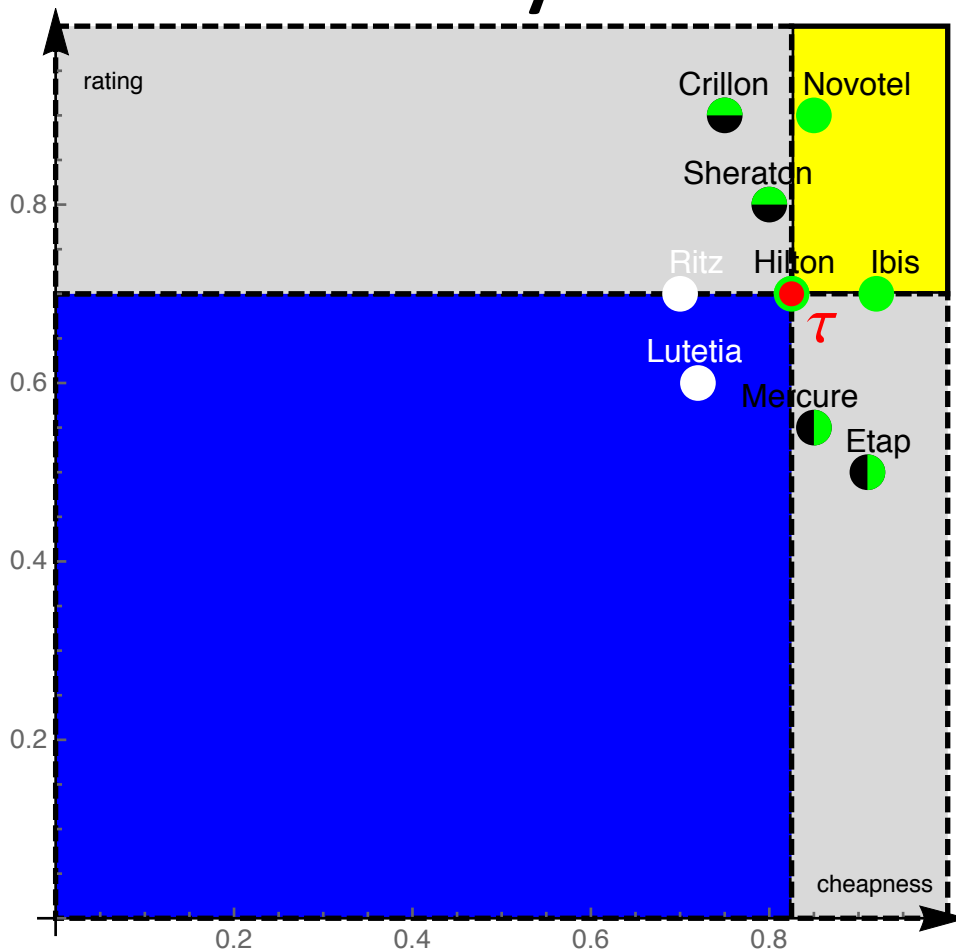
- The **threshold point** (τ) is the point with the smallest seen values on all lists in the sorted access phase
- FA stops when the **yellow region** (fully seen points) contains at least k points
- The **gray regions** contain the points seen in at least one ranking
- None of the points in the **blue region** (unseen points) can beat any point in the **yellow region**

Why does FA work?



- The **threshold point** (τ) is the point with the smallest seen values on all lists in the sorted access phase
- FA stops when the **yellow region** (fully seen points) contains at least k points
- The **gray regions** contain the points seen in at least one ranking
- None of the points in the **blue region** (unseen points) can beat any point in the **yellow region**

Why does FA work?



- The **threshold point** (τ) is the point with the smallest seen values on all lists in the sorted access phase
- FA stops when the **yellow region** (fully seen points) contains at least k points
- The **gray regions** contain the points seen in at least one ranking
- None of the points in the **blue region** (unseen points) can beat any point in the **yellow region**

Limits of FA

- Drawback: specific scoring function not exploited at all
 - In particular, the sorted+random access cost of FA is independent of the scoring function!
- Memory requirements can become prohibitive
 - FA has to buffer all the objects accessed through sorted access
- Small improvements are possible
 - E.g., by interleaving random accesses and score computation, which might save some random access
- Significant improvements require changing the stopping condition

Threshold Algorithm (TA)

[Fagin, Lotem, Naor, PODS 2001]

Input: integer k , a **monotone** function S combining ranked lists R_1, \dots, R_m

Output: the top k <object, score> pairs

1. Do a **sorted access** in parallel in each list R_i
2. For each object o , do **random accesses** in the other lists R_j , thus extracting score s_j
3. Compute overall score $S(s_1, \dots, s_m)$. If the value is among the k highest seen so far, remember o
4. Let s_{Li} be the last score seen under sorted access for R_i
5. Define threshold $T = S(s_{L1}, \dots, s_{Lm})$ --> we define the threshold as the score of the last seen value, which is the worst among those we've seen but cannot be beaten by the next accessed values
6. If the score of the k -th object is worse than T , go to step 1
7. Return the current top- k objects

- TA is **instance-optimal** among all algorithms that use random and sorted accesses (FA is not)
 - The stopping criterion **depends** on the scoring function
- The authors of TA received the **Gödel prize in 2014** for the design of innovative algorithms

Example: hotels in Paris with TA

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
 - Alternate sorted access and random access
 - Maintain a threshold T
 - Stop when k objects are no worse than T

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score

Threshold
value: $T = ??$
point: $\tau = (??, ??)$

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

As soon as we get them by sorted access, we complete their info by random access and compute their score

Top 2	Score
Crillon	.825
Ibis	.81

Threshold
 value: $T = .91$
 point: $\tau = (.92, .9)$
the coordinates

--> is computed by the last seen value in the first and the second table: $0.5 * (.92) + 0.5 * (.9) = 0.91$

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Then make a **random access** for each new hotel

I continue cause the scores of the top 2 are worse than the threshold

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	

Top 2	Score
Novotel	.875
Crillon	.825

Still worse than threshold, continue

Threshold
value: $T = .905$
point: $\tau = (.91, .9)$

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Make one sorted access at a time in each list
 - Then make a **random access** for each new hotel

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
...		...	



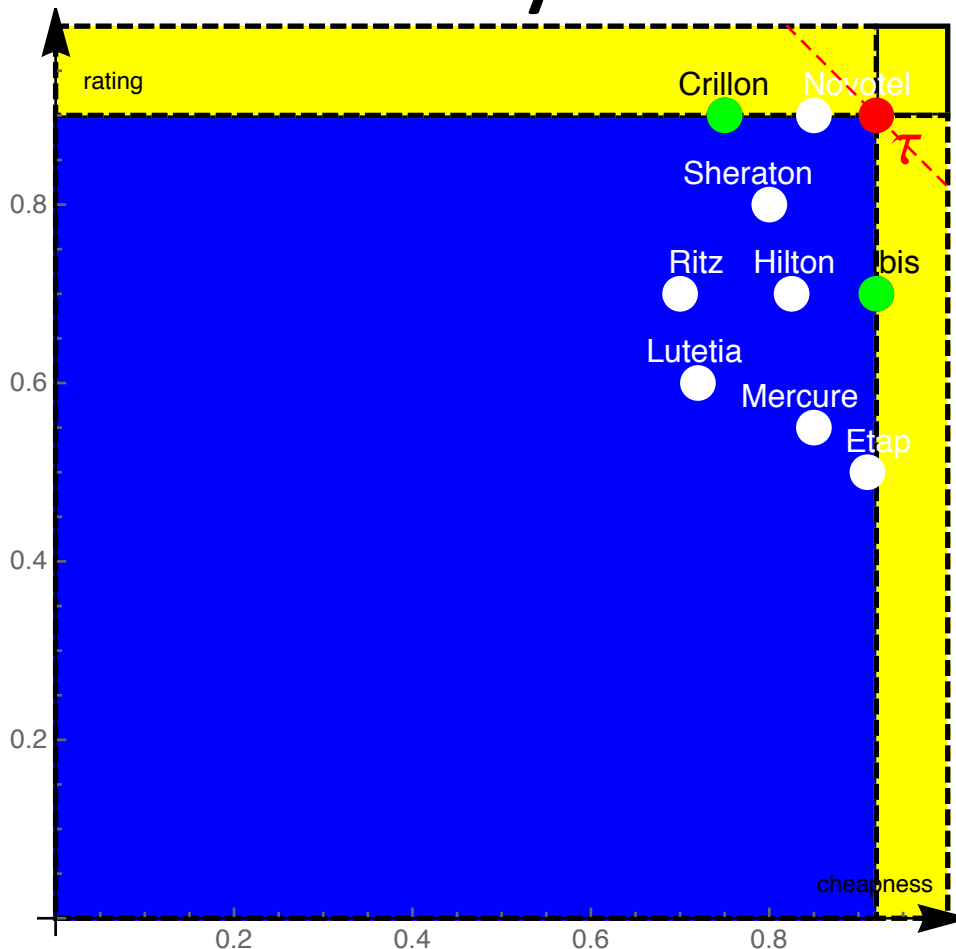
Top 2	Score
Novotel	.875
Crillon	.825

Threshold
value: $T = .825$
point: $\tau = (.85, .8)$

Threshold has been lowered

- Query: hotels with best price and rating
 - Scoring function: $0.5 * \text{cheapness} + 0.5 * \text{rating}$
- Strategy:
 - Stop when the score of the k -th hotel is no worse than the threshold

Why does TA work?

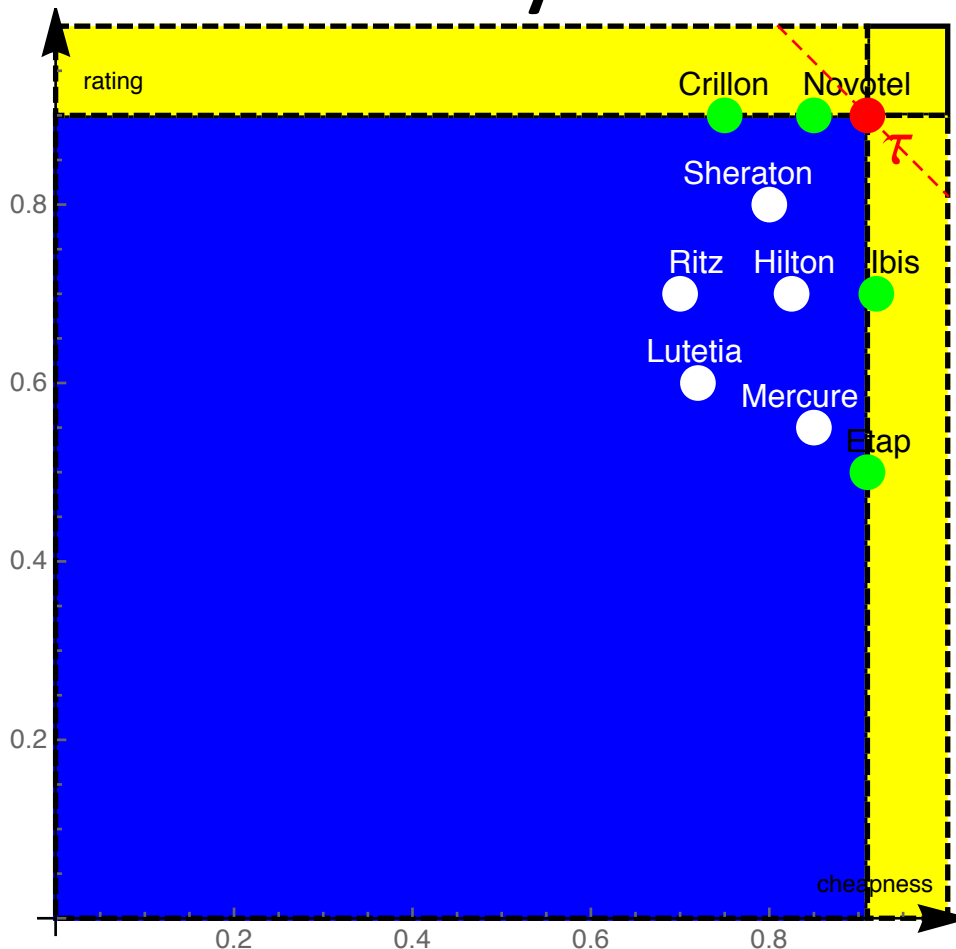


- τ is the **threshold point**
- FA stops when the **yellow region** (fully seen points) contains at least k points at least as good as τ
- None of the points in the **blue region** (unseen points) can beat τ
- The dashed **red line** separates the region of points with a higher score than τ from the rest
 - No hotel is as good as τ or better

The points that live in the upper triangle (bordered with the red dashed line) can never be beaten by the one in the lower part, because that represent the score of the threshold point

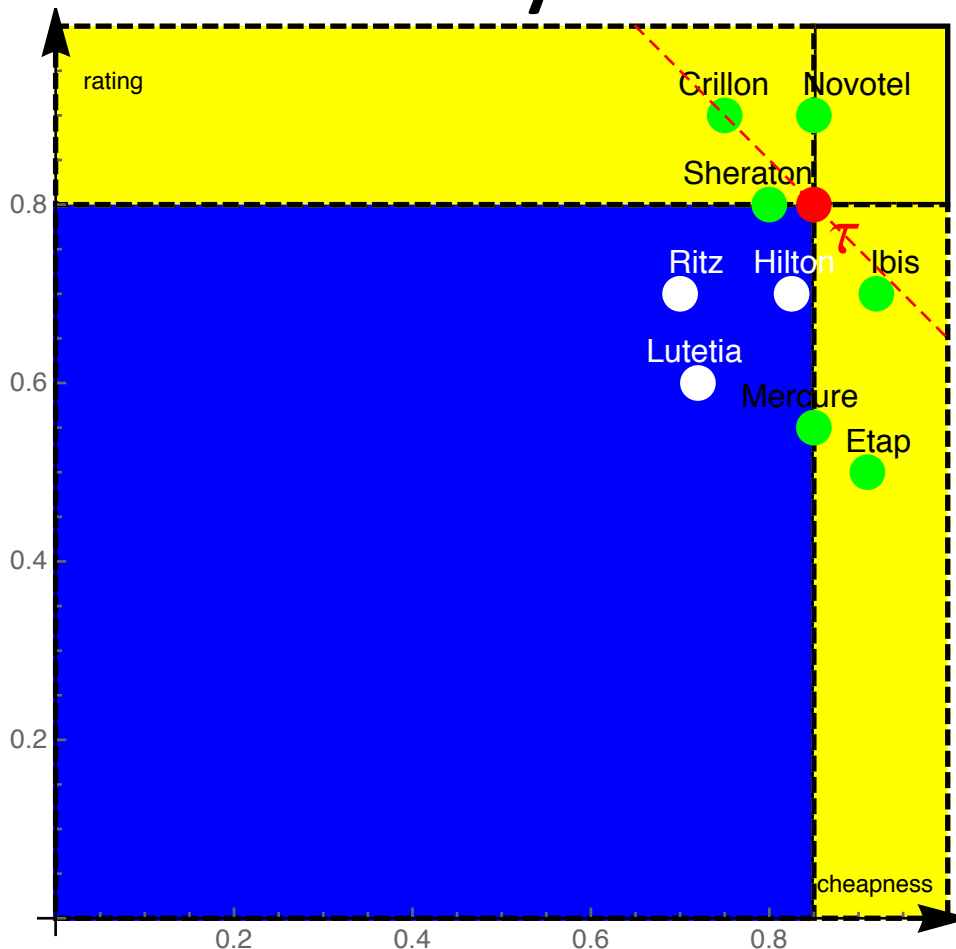
So if I exploit the scoring function i'm able to extend the area where objects that cannot be beaten live.

Why does TA work?



- τ is the **threshold point**
- FA stops when the **yellow region** (fully seen points) contains at least k points at least as good as τ
- None of the points in the **blue region** (unseen points) can beat τ
- The dashed **red line** separates the region of points with a higher score than τ from the rest
 - Still no hotel is as good as τ or better

Why does TA work?



- τ is the **threshold point**
- FA stops when the **yellow region** (fully seen points) contains at least k points at least as good as τ
- None of the points in the **blue region** (unseen points) can beat τ
- The dashed **red line** separates the region of points with a higher score than τ from the rest
 - Now, Crillon is as good as τ and Novotel is better

Performance of TA: Cost model

- In general, TA performs much better than FA, since it can “adapt” to the specific scoring function
- In order to characterize the performance of TA, we consider the so-called **middleware cost**: $\text{cost} = SA * C_{SA} + RA * C_{RA}$,
where:
#sorted accesses* cost of sort.acc + the same with random accesses
 - SA (RA) is the total number of sorted (random) accesses
 - C_{SA} (C_{RA}) is the unitary (base) cost of a sorted (random) access
- In the basic setting, $C_{SA} = C_{RA}$ (=1, for simplicity)
- In other cases, base costs may differ
 - For web sources, usually $C_{RA} > C_{SA}$
 - limit case $C_{RA} = \infty$ (**random access is impossible**) in this case we can use B0, which only works with sorted accesses. It may be too specific cause it only works with S=MAX. So the next slide will present an alternative
 - Some sources might not be accessible through sorted access
 - $C_{SA} = \infty$ (for instance, we do not have an index to process p_j)

The NRA algorithm: preliminaries

[Fagin, Lotem, Naor, PODS 2001]

- NRA (No Random Access) applies when random accesses cannot be executed
 - It returns the top-k objects, but their scores might be uncertain
 - This is to limit the cost of the algorithm
- Idea: maintain, for each object o retrieved by sorted access, a lower bound $S^-(o)$ and an upper bound $S^+(o)$ on its score
- NRA uses a buffer B with unlimited capacity, which is kept sorted according to decreasing lower bound values

L1		L2		L3		$S \equiv \text{SUM}$	B		
OID	p1	OID	p2	OID	p3		OID	lbscore	ubscore
o1	1.0	o2	0.8	o7	0.6		o7	1.5	2.25
o7	0.9	o3	0.75	o2	0.6		o2	1.4	2.3
...		o1	1.0	2.35
							o3	0.75	2.25

The NRA algorithm

The idea is: we keep doing sorted access on all rankings until we have at least k objects that are certainly better than ...

- Maintain a set **Res** of current best k objects
- Halt when no object o' not in **Res** can do better than any of the objects in **Res**
$$\forall o' \notin \text{Res}, \forall o \in \text{Res}: S^+(o') \leq S^-(o)$$
 - I.e., when $S^-(o)$ beats both
 - the max value of $S^+(o')$ among the objects in **B-Res**
 - and the score $S(\tau)$ of the threshold point τ (upper bound to unseen objects)

Input: integer $k \geq 1$, a **monotone** function S combining ranked lists R_1, \dots, R_m

Output: the top- k objects according to S

1. Make a sorted access to each list
2. Store in **B** each retrieved object o and maintain $S^-(o)$ and $S^+(o)$ and a threshold τ
3. Repeat from step 1 as long as $S^-(B[k]) < \max\{ \max\{S^+(B[i]), i > k\}, S(\tau) \}$

NRA: example

R_1

OID	p1
o1	1.0
o7	0.9
o2	0.7
o6	0.2
...	...

R_2

OID	p2
o2	0.8
o3	0.75
o4	0.5
o1	0.4
...	...

R_3

OID	p3
o7	0.6
o2	0.6
o3	0.5
o5	0.1
...	...

$S \equiv \text{SUM}$

$k = 2$

NRA: example

R_1		R_2		R_3	
OID	p1	OID	p2	OID	p3
o1	1.0	o2	0.8	o7	0.6
o7	0.9	o3	0.75	o2	0.6
o2	0.7	o4	0.5	o3	0.5
o6	0.2	o1	0.4	o5	0.1
...

$S \equiv \text{SUM}$
 $k = 2$

$T = 2.4$

$S(\tau)$

B (1st round)

OID	lbscore	ubscore
o1	1.0	2.4
o2	0.8	2.4
o7	0.6	2.4

$0.8 < \max\{2.4, 2.4\}$

The lower bound correspond to the scoring function computed considering the values discovered so far see associated to a certain object (ex. for o1 we see just 1.0 in the R1 table, so over that S will be computed, we get 1).

The upper bound for an object o is the S function applied to the last occurrence of the object in each table, if for some tables there isn't such occurrence the last seen object is considered. For example, the highest score for object o1 is

R1: there's o1 so it's p1 is considered (1.0)

R2: there's not o1 so the last seen p2 is considered (0.8)

R3: there's not o1 so the last seen p3 is considered (0.6)

o1 upper bound = $1 + 0.8 + 0.6 = 2.4$

Since our second (k-est, in this case is the second) best lower bound is lower than the score of the last seen object and it is lower than the highest upper bound found among the rows below its row, we need to continue

NRA: example

R ₁		R ₂		R ₃	
OID	p1	OID	p2	OID	p3
o1	1.0	o2	0.8	o7	0.6
o7	0.9	o3	0.75	o2	0.6
o2	0.7	o4	0.5	o3	0.5
o6	0.2	o1	0.4	o5	0.1
...

$S \equiv \text{SUM}$
 $k = 2$

$T = 2.25$

B (1st round)

OID	lbscore	ubscore
o1	1.0	2.4
o2	0.8	2.4
o7	0.6	2.4

$0.8 < \max\{2.4, 2.4\}$

B (2nd round)

OID	lbscore	ubscore
o7	1.5	2.25
o2	1.4	2.3
o1	1.0	2.35
o3	0.75	2.25

$1.4 < \max\{2.35, 2.25\}$

highest
upper bound

Continues: Lower bound: for o1 we see just 1.0 in the R1 table, so over that S will be computed

NRA: example

R ₁		R ₂		R ₃	
OID	p1		OID	p2	
o1	1.0		o2	0.8	
o7	0.9		o3	0.75	
o2	0.7		o4	0.5	
o6	0.2		o1	0.4	
...	

$S \equiv \text{SUM}$
 $k = 2$

$T = 1.7$

B (1st round)

OID	lbscore	ubscore
o1	1.0	2.4
o2	0.8	2.4
o7	0.6	2.4

$$0.8 < \max\{2.4, 2.4\}$$

B (2nd round)

OID	lbscore	ubscore
o7	1.5	2.25
o2	1.4	2.3
o1	1.0	2.35
o3	0.75	2.25

$$1.4 < \max\{2.35, 2.25\}$$

B (3rd round)

OID	lbscore	ubscore
o2	2.1	2.1
o7	1.5	2.0
o3	1.25	1.95
o1	1.0	2.0
o4	0.5	1.7

$$1.5 < \max\{2.0, 1.7\}$$

NRA: example

R ₁		R ₂		R ₃	
OID	p1	OID	p2	OID	p3
o1	1.0	o2	0.8	o7	0.6
o7	0.9	o3	0.75	o2	0.6
o2	0.7	o4	0.5	o3	0.5
o6	0.2	o1	0.4	o5	0.1
...

$S \equiv \text{SUM}$
 $k = 2$

$T = 0.7$

B (1st round)

OID	lbscore	ubscore
o1	1.0	2.4
o2	0.8	2.4
o7	0.6	2.4

$0.8 < \max\{2.4, 2.4\}$

B (4th round)

OID	lbscore	ubscore
o2	2.1	2.1
o7	1.5	1.9
o1	1.4	1.5
o3	1.25	1.45
o4	0.5	0.8
o6	0.2	0.7
o5	0.1	0.7

$1.5 \geq \max\{1.5, 0.7\}$

B (2nd round)

OID	lbscore	ubscore
o7	1.5	2.25
o2	1.4	2.3
o1	1.0	2.35
o3	0.75	2.25

$1.4 < \max\{2.35, 2.25\}$

B (3rd round)

OID	lbscore	ubscore
o2	2.1	2.1
o7	1.5	2.0
o3	1.25	1.95
o1	1.0	2.0
o4	0.5	1.7

$1.5 < \max\{2.0, 1.7\}$

NRA: observations

Notice that NRA does not guarantee to find the exact score but guarantees to find the top-k objects

- NRA's cost does not grow monotonically with k , i.e.
 - it might be cheaper to look for the top- k objects rather than for the top- $(k-1)$ ones!

Example:

- $k = 1$: the winner is $o2$ ($S(o2) = 1.2$), since the score of $o1$ is $S(o1) = 1.0$ and that of all other objects is 0.6. NRA has to reach depth $N-1$ to halt
- $k = 2$: to discover that the top-2 objects are $o1$ and $o2$ only 3 rounds are needed

R_1		R_2	
OID	p1	OID	p2
o1	1.0	...	0.3
o2	1.0
...	0.3	...	0.3
...	...	o2	0.2
...	0.3	o1	0

$S \equiv \text{SUM}$

- NRA is instance-optimal among all algorithms that do not make random accesses
 - optimality ratio is m

The overall picture

Algorithm	scoring function	Data access	Notes
B_0	MAX	sorted	instance-optimal
FA	monotone	sorted and random	cost independent of scoring function
TA	monotone	sorted and random	instance-optimal
NRA	monotone	sorted	instance-optimal, no exact scores

Summary on top-k 1-1 join queries

- There are several algorithms to process a top-k 1-1 join query
- Common assumption: the scoring function S is **monotone**
- Simplest case: MAX
- FA's stopping condition does not use the scoring function
- TA's stopping condition is based on a threshold T , which provides an upper bound to the scores of all unseen objects
- TA is instance-optimal, FA isn't
- NRA does not execute random accesses at all

Ranking queries – main aspects

Pros:

- Very **effective** in identifying the best objects
 - Wrt. a specific **scoring function**
- Excellent **control of the cardinality** of the result
 - k is an input parameter of a top- k query

Con ---->

- For a user, it is **difficult to specify** a scoring function
 - E.g., the weights of a weighted sum
- Computation is very **efficient**
 - E.g., $O(N \log k)$ for local, unordered datasets of N elements
 - Instance optimality in some settings
- Easy to express the **relative importance of attributes**

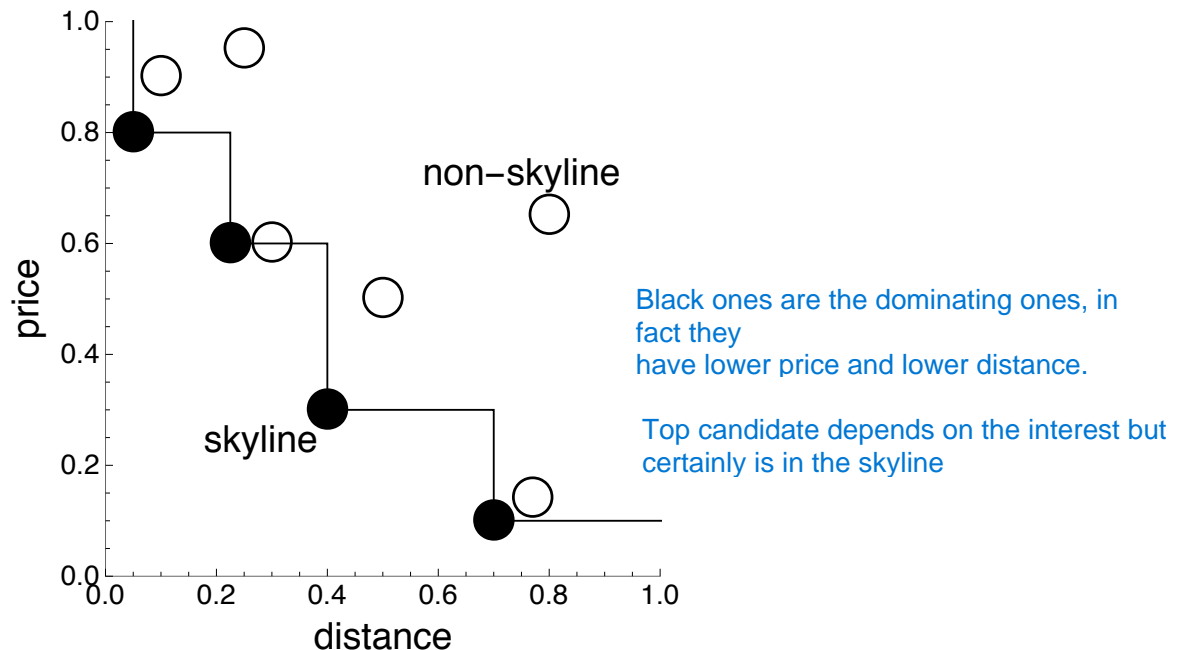
Skyline queries

Skylines: background

- Find good objects according to several different perspectives
 - e.g., attribute values A_1, \dots, A_m
- No need to specify weights!
 - based on the notion of **dominance**
- Tuple t **dominates** tuple s , indicated $t < s$, iff
 - $\forall i. 1 \leq i \leq m \rightarrow t[A_i] \leq s[A_i]$ (t is nowhere worse than s)
 - $\exists j. 1 \leq j \leq m \wedge t[A_j] < s[A_j]$ (and better at least once)
 - Typically, lower values are considered better
 - Opposite convention wrt. top-k queries, but it's just a convention that can be changed!

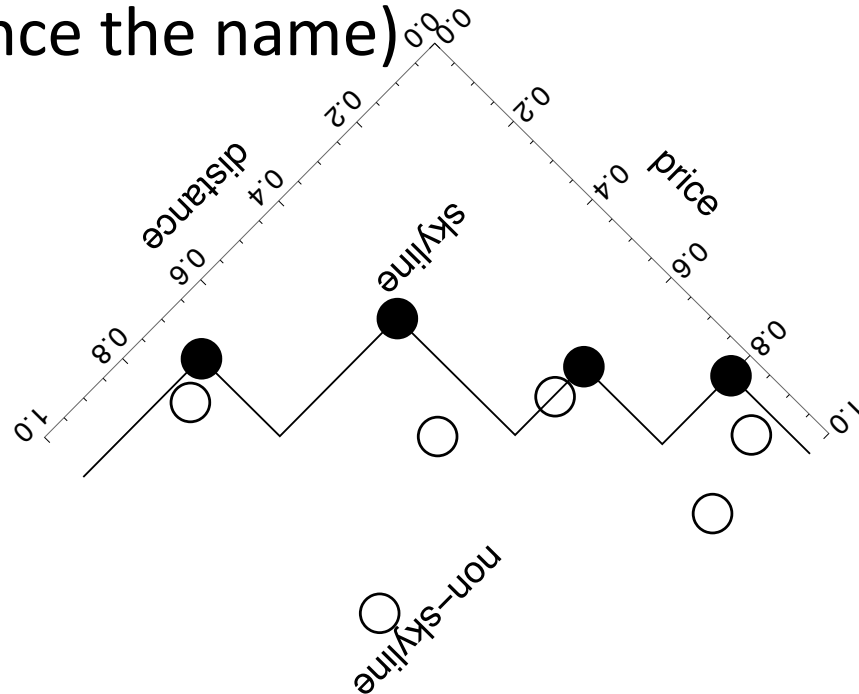
Skylines

- The **skyline** of a relation is the set of its non-dominated tuples. Aka:
 - **Maximal vectors problem** (computational geometry)
 - **Pareto-optimal solutions** (multi-objective optimization)



Skylines

- The **skyline** of a relation is the set of its non-dominated tuples. Aka:
 - **Maximal vectors problem** (computational geometry)
 - **Pareto-optimal solutions** (multi-objective optimization)
- In 2D, the shape resembles the contour of the dataset (hence the name)



Skylines: properties

- A tuple t is in the skyline iff it is the top-1 result w.r.t. at least one monotone scoring function
 - i.e., the skyline is the set of **potentially optimal** tuples!
- Skyline \neq Top- k query
 - there is no scoring function that, on all possible instances, yields in the first k positions the skyline points
 - based on the notion of **dominance**

Skyline queries in SQL

[Börzsönyi et al., ICDE 2001]

- Only a proposed syntax (not part of the standard):

```
SELECT ... FROM ... WHERE ...  
GROUP BY ... HAVING ...  
SKYLINE OF [DISTINCT] d1 [MIN | MAX | DIFF],  
                ..., dm [MIN | MAX | DIFF]  
ORDER BY ...
```

- Example

```
SELECT * FROM Hotels WHERE city = 'Paris'  
SKYLINE OF price MIN, distance MIN
```

price
I want to minimize (low values are good)

Translation into naively nested SQL

```
SELECT * FROM Hotels WHERE city = 'Paris'  
SKYLINE OF price MIN, distance MIN
```

- **Can be easily translated to actual SQL:**

```
SELECT * FROM Hotels h  
WHERE h.city = 'Paris' AND NOT EXISTS (  
    SELECT * FROM Hotels h1  
    WHERE h1.city = h.city AND  
        h1.distance <= h.distance AND  
        h1.price <= h.price AND  
        (h1.distance < h.distance OR  
        h1.price < h.price))
```

- **Very slow! We need a better way...**

Quadratic complexity (comparing every n tuples to every $n-1$ other one) --> not good in db

Skylines – Block Nested Loop (BNL)

[Börzsönyi et al., ICDE 2001]

Input: a dataset D of multi-dimensional points

Output: the skyline of D

1. Let $W = \emptyset$
2. for every point p in D
3. if p not dominated by any point in W
4. remove from W the points dominated by p
5. add p to W
6. return W

- Computation is $O(n^2)$ where $n = |D|$
- Very inefficient for large datasets

Skylines – Sort-Filter-Skyline (SFS)

[Chomicki et al., ICDE 2003]

Input: a dataset D of multi-dimensional points

Output: the skyline of D

1. Let $S = D$ sorted by a monotone function of D 's attributes
2. Let $W = \emptyset$
3. for every point p in S
4. if p not dominated by any point in W
5. add p to W
6. return W

If we sort the dataset it becomes a dataset in a topological order with respect to dominance: you can dominate a tuple which is sorted above you.

Here we sorted so we don't need to make a check about removing points that are dominated by the new one, which was present in the previous algorithm

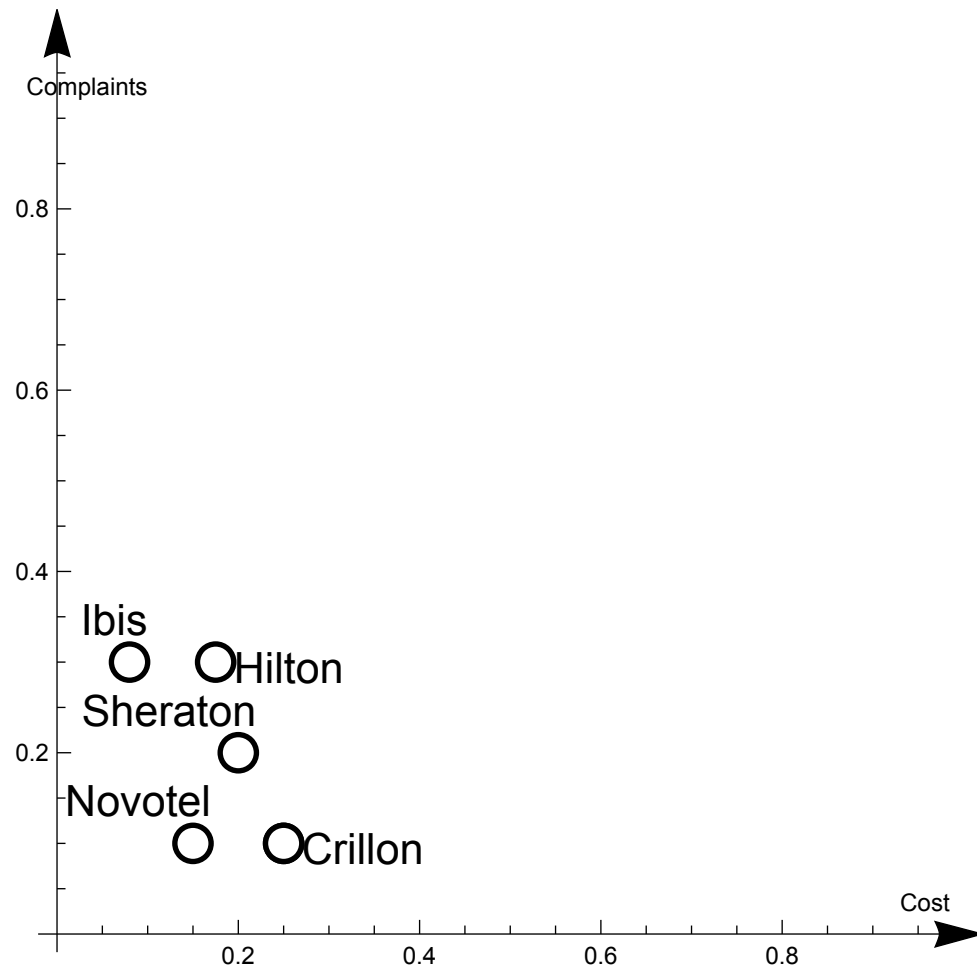
- Pre-sorting pays off for large datasets, thus SFS performs much better than BNL
 - If the input is sorted, then a later tuple cannot dominate any previous tuple!
 - Will never compare two non-skyline points
 - Can immediately output any points in W as part of the result
 - But still $O(n^2)$

Table after join:

Hotels	Cost	Complaints
Crillon	.25	.1
Ibis	.08	.3
Hilton	.175	.3
Sheraton	.2	.2
Novotel	.15	.1

SFS

- Example dataset
 - (low values are good)

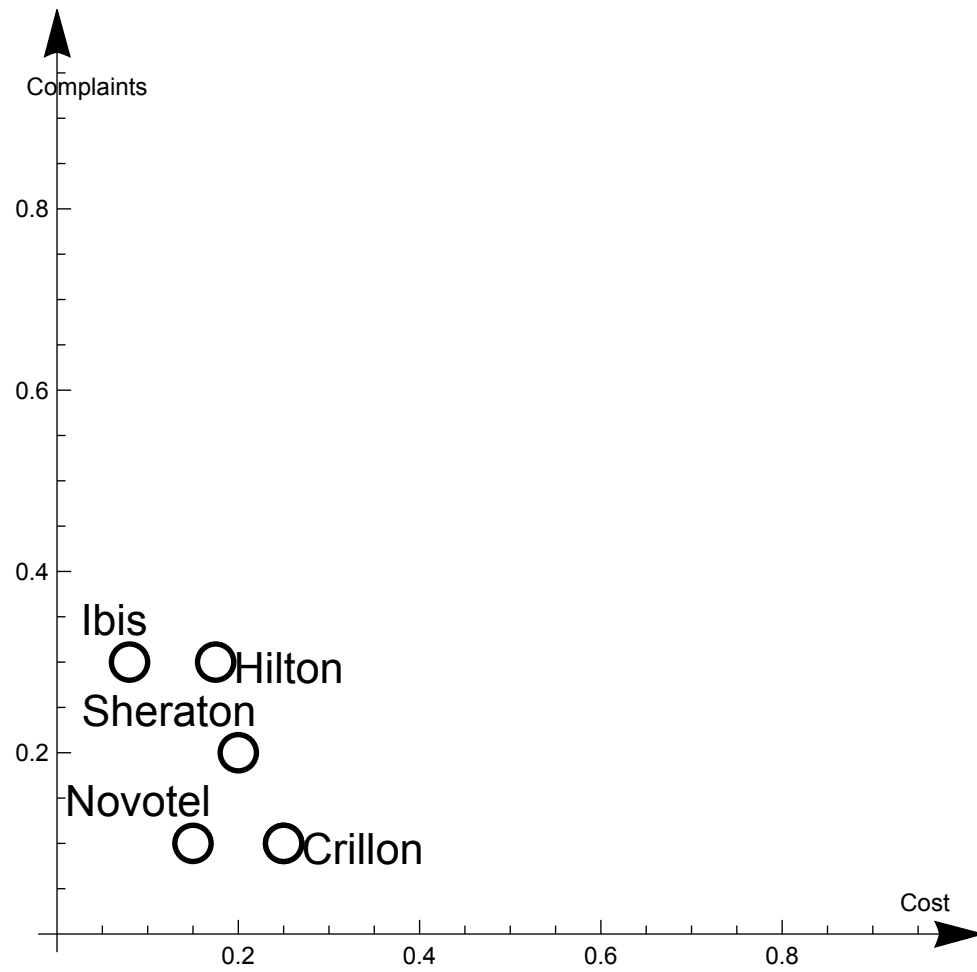


Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

- Sorted dataset

- E.g., by Cost + Complaints

Not a good choice, it is better to first order by the first attribute, then the second ecc...



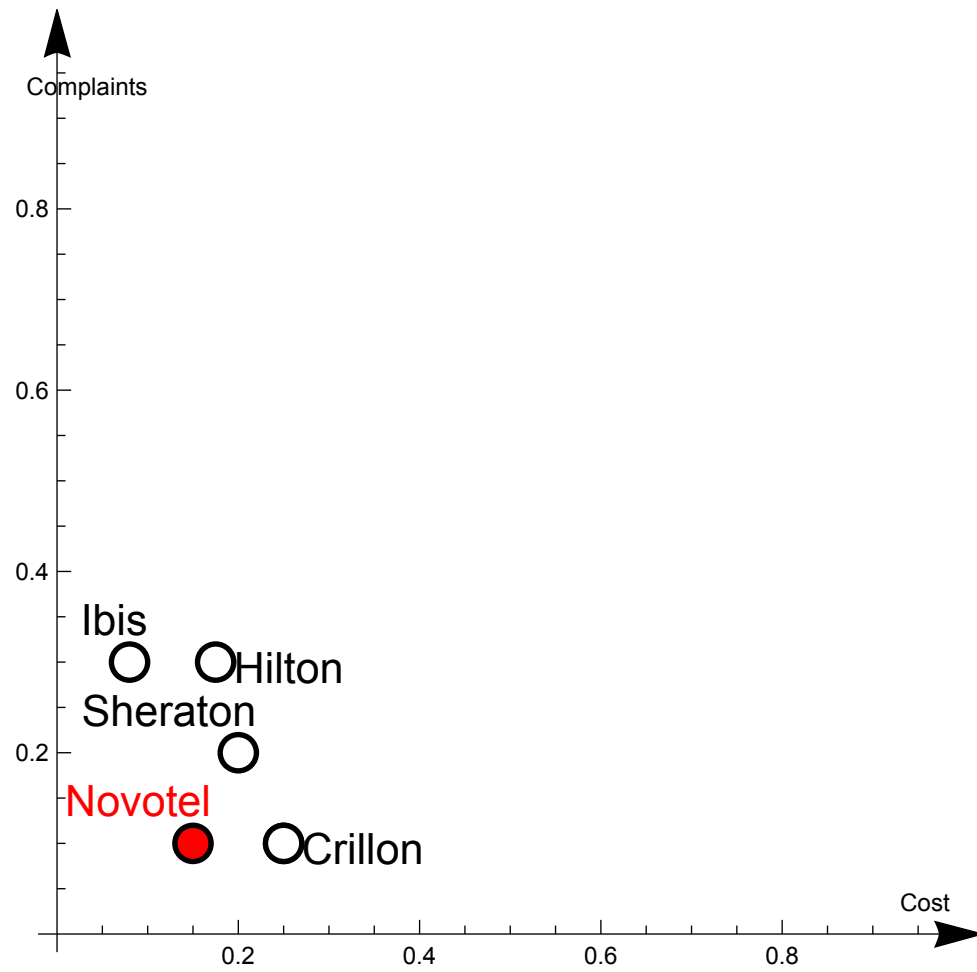
First one certainly part of the skyline (it is not dominated by any point)

Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

Window

Hotels	Cost	Complaints
Novotel	.15	.1

- Sorted dataset
 - Add if not dominated by any point in the window



Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

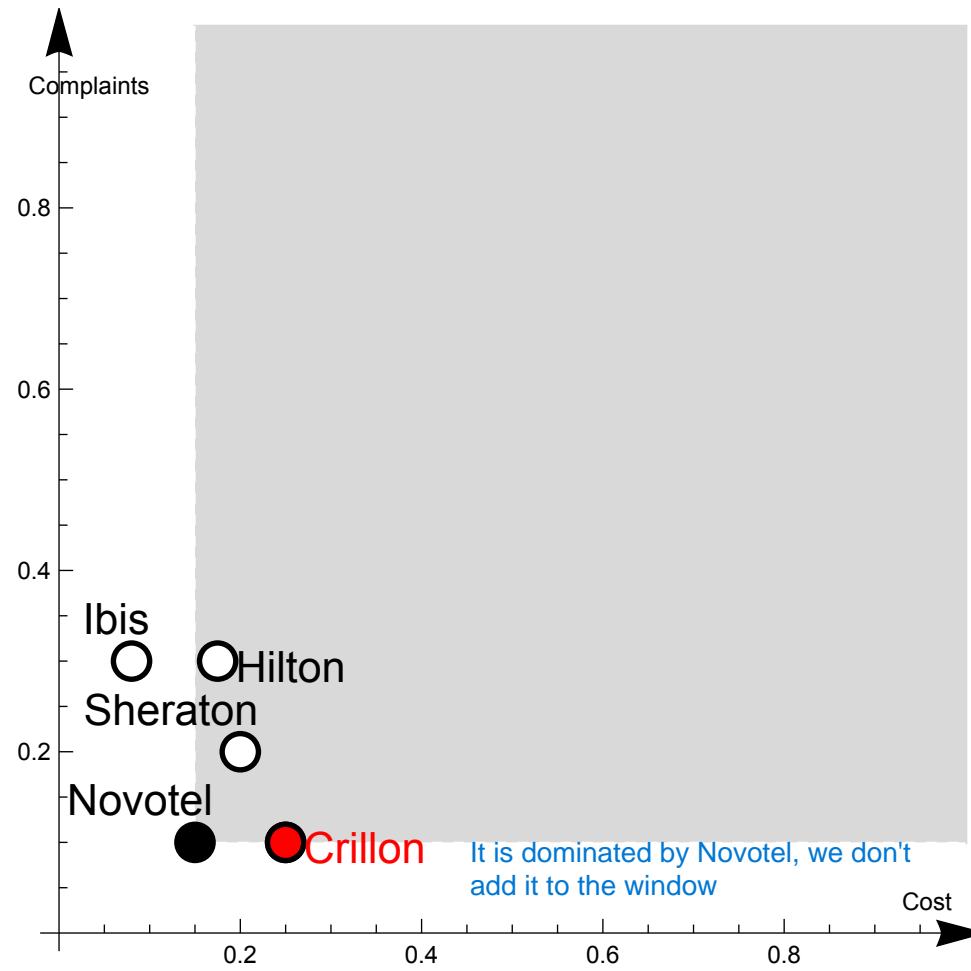
Window

Hotels	Cost	Complaints
Novotel	.15	.1

Region of points dominated by novotel

Gray area: dominance region of the point(s) in the window

- Sorted dataset
 - Add if not dominated by any point in the window

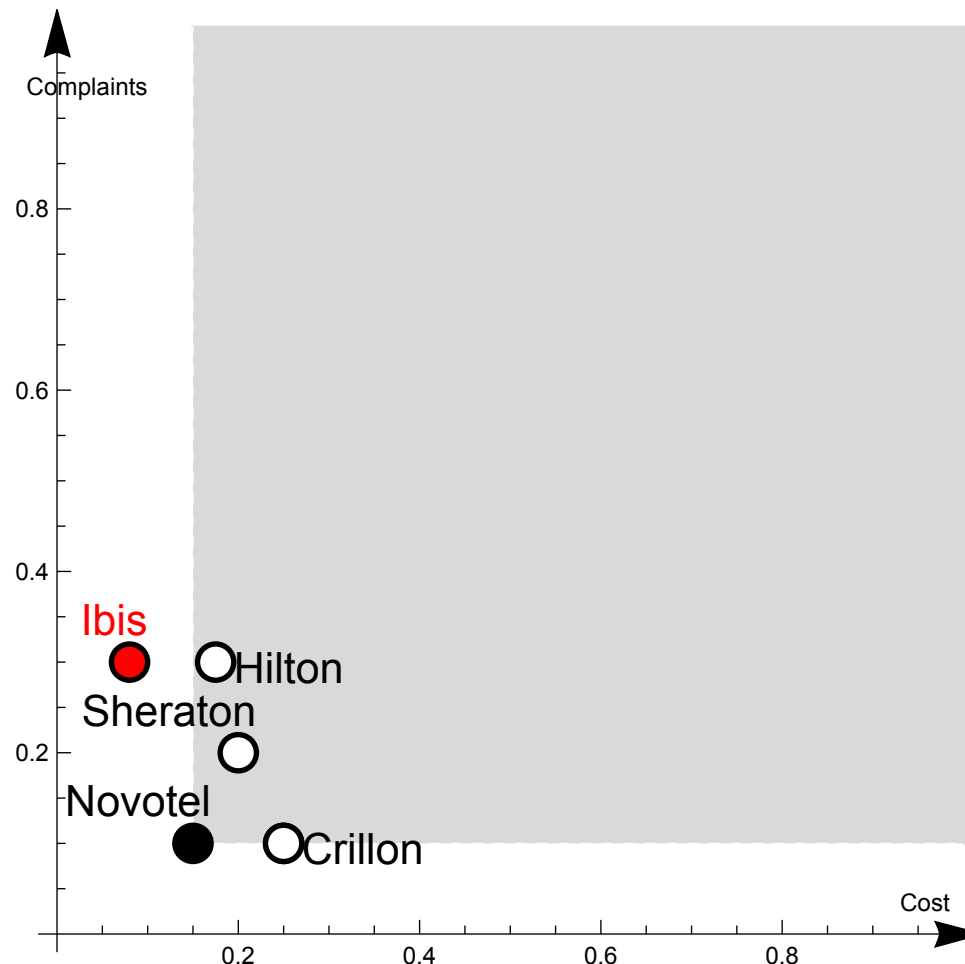


Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

Window

Hotels	Cost	Complaints
Novotel	.15	.1
Ibis	.08	.3

- Sorted dataset
 - Add if not dominated by any point in the window

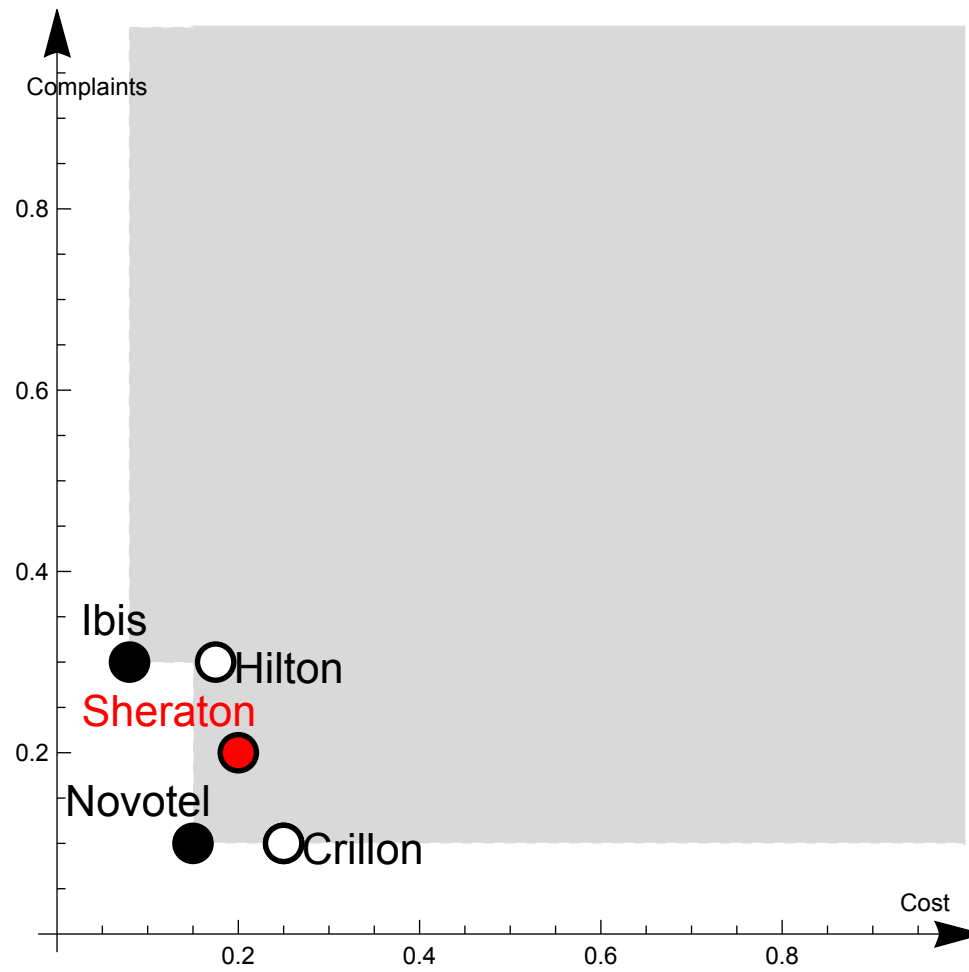


Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

Window

Hotels	Cost	Complaints
Novotel	.15	.1
Ibis	.08	.3

- Sorted dataset
 - Add if not dominated by any point in the window

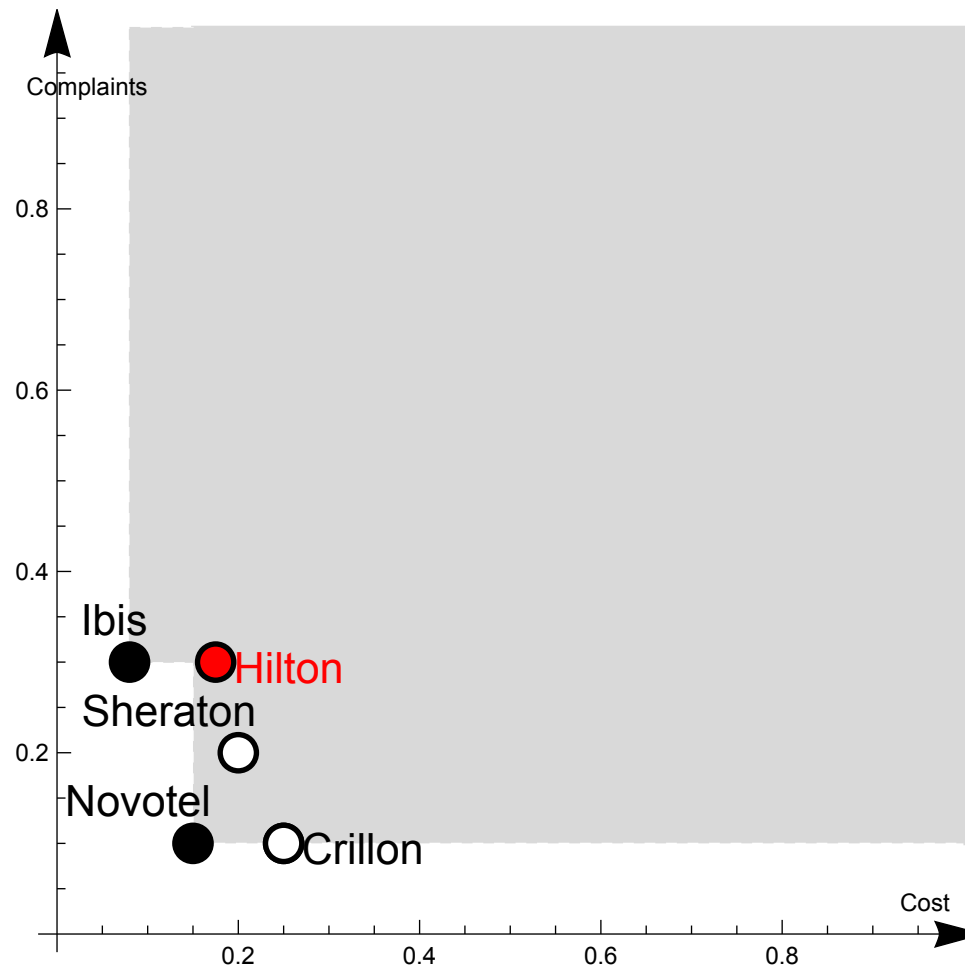


Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

Window

Hotels	Cost	Complaints
Novotel	.15	.1
Ibis	.08	.3

- Sorted dataset
 - Add if not dominated by any point in the window

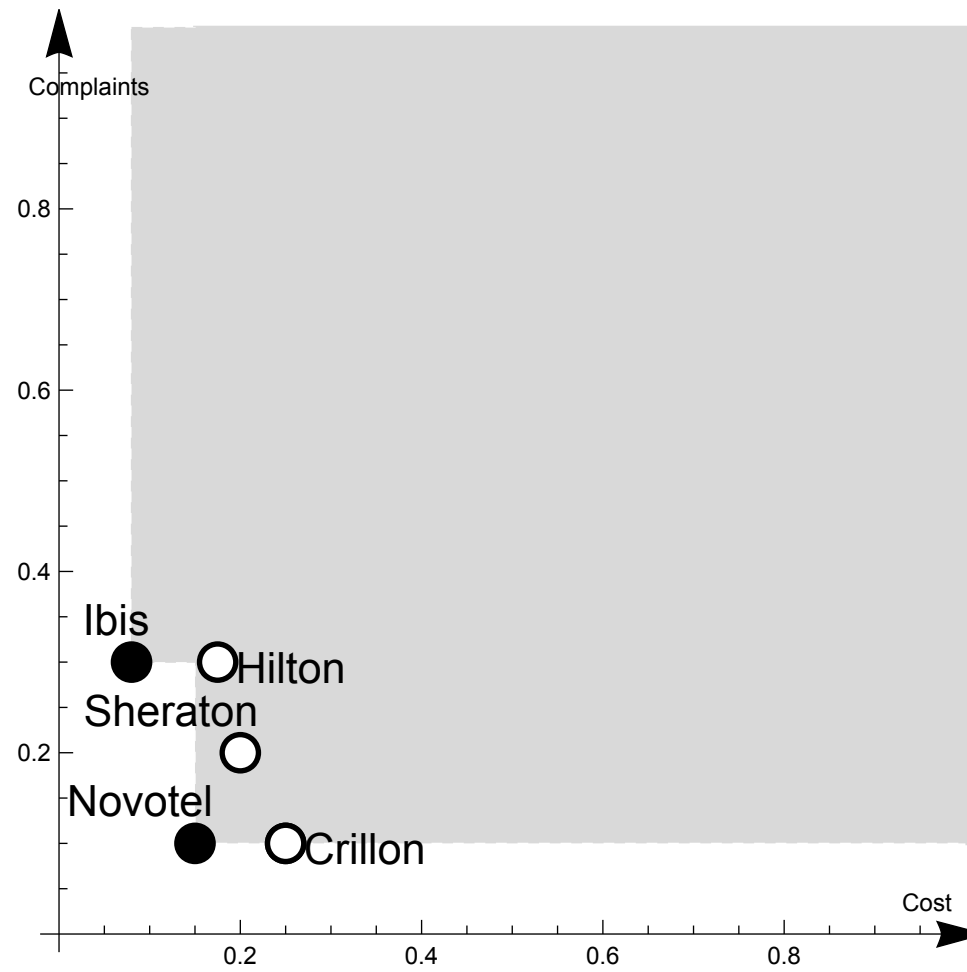


Hotels	Cost	Complaints
Novotel	.15	.1
Crillon	.25	.1
Ibis	.08	.3
Sheraton	.2	.2
Hilton	.175	.3

This is the skyline

Hotels	Cost	Complaints
Novotel	.15	.1
Ibis	.08	.3

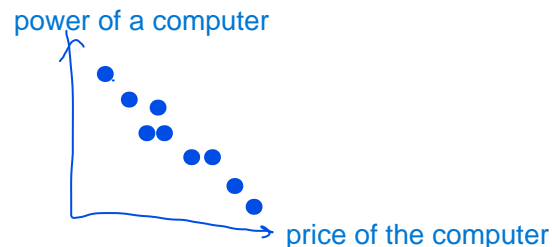
- Sorted dataset
 - Add if not dominated by any point in the window



Skylines – main aspects

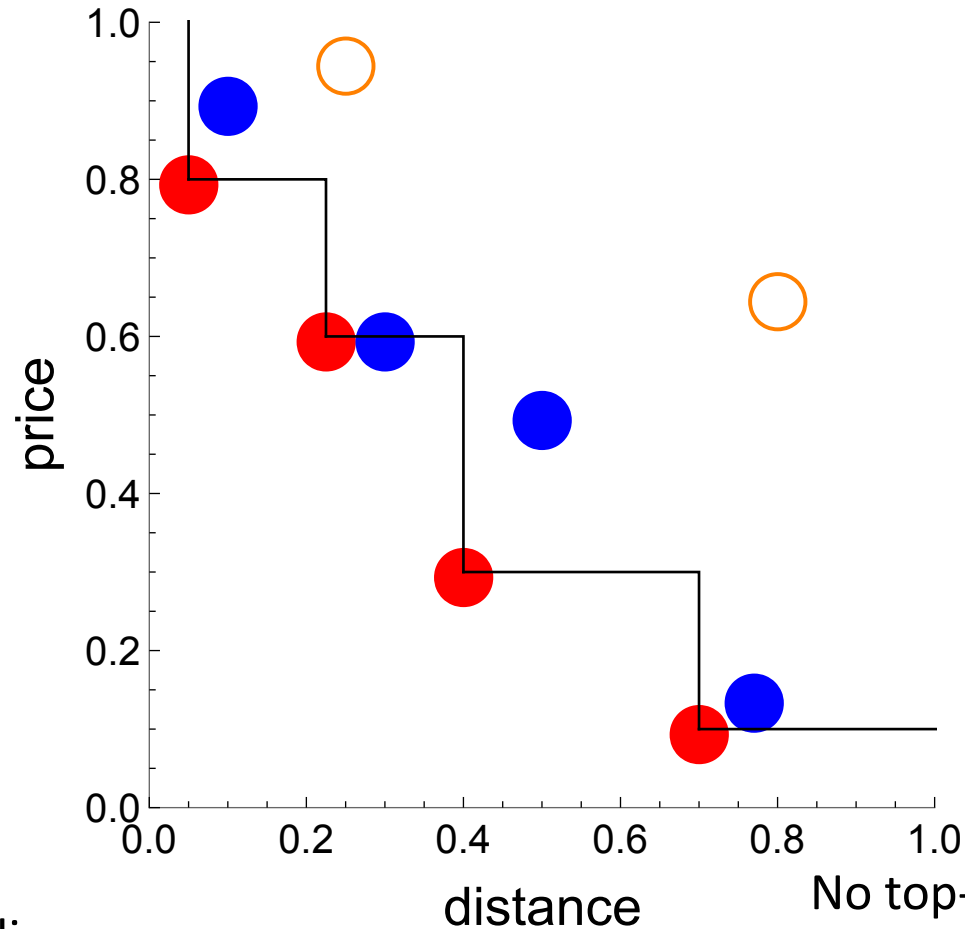
- Pros:
 - **Effective** in identifying potentially interesting objects if nothing is known about the preferences of a user
 - **Very simple** to use (no parameters needed!)
- Cons:
 - We can't have a good control on the cardinality (size of the output)
 - **Too many objects** returned for large, anti-correlated datasets
 - **Computation** is essentially **quadratic** in the size of the dataset (and thus **not so efficient**)
 - **Agnostic wrt. known user preferences** (e.g., price is more important than distance) good and bad at the same time
- Extension: **k-skyband** = set of tuples dominated by less than k tuples
 - This extension can limit the cons of having too many objects in return
 - Skyline = 1-skyband
 - Every top-k result set is contained in the k-skyband

Skyline is ineffective in anticorrelated datasets



Comparing different approaches

Example: skyline/k-skyband query



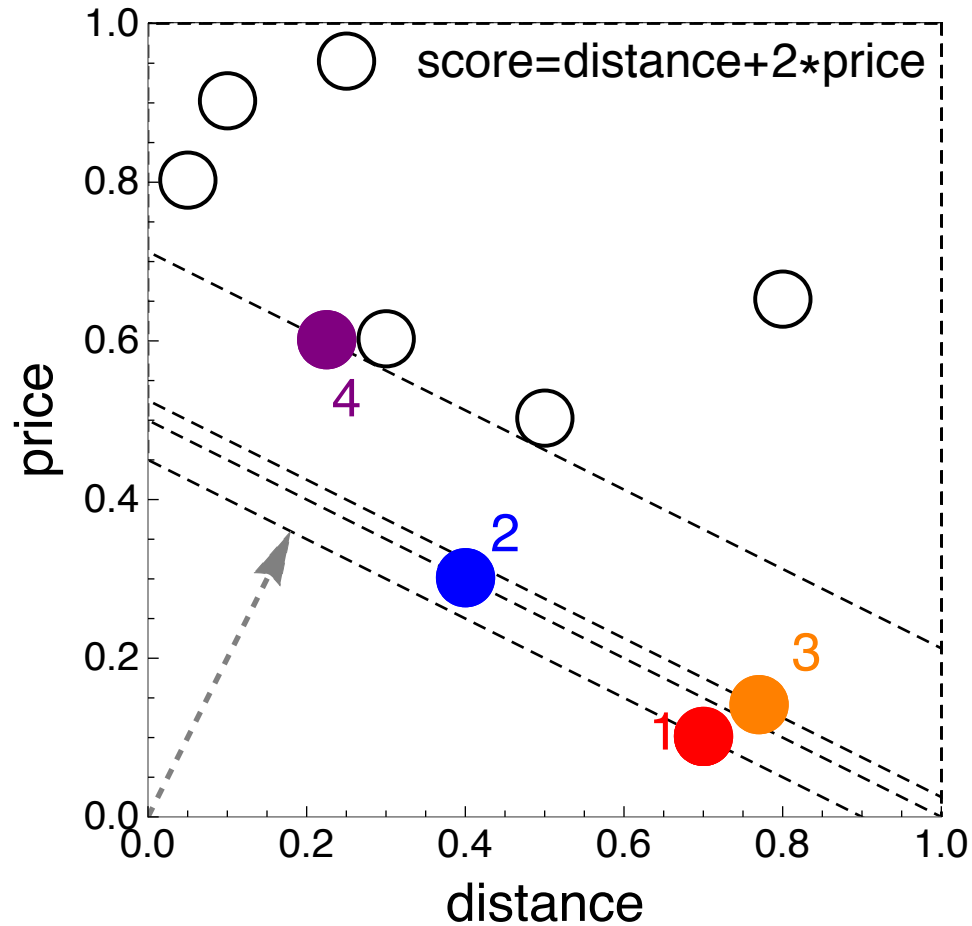
skyline



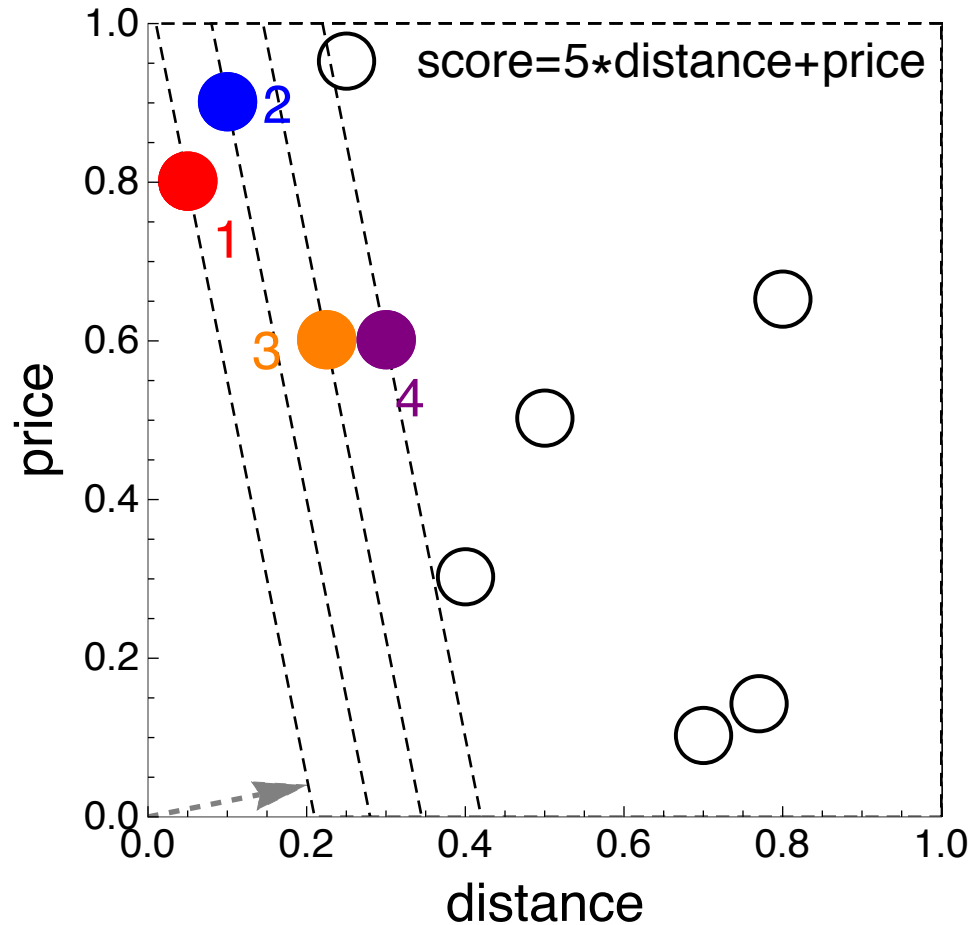
2-skyband = 3-skyband

No top-2 or top-3 query
will return a  point

Example: ranking query



Example: another ranking query



Comparing different approaches

	Ranking queries	Skyline queries
Simplicity	No	Yes
Overall view of interesting results	No	Yes
Control of cardinality	Yes	No
Trade-off among attributes	Yes	No

Main References

Historical papers

- Jean-Charles de Borda
Mémoire sur les élections au scrutin. Histoire de l'Académie Royale des Sciences, Paris 1781
- Nicolas de Condorcet
Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix, 1785
- Kenneth J. Arrow
A Difficulty in the Concept of Social Welfare. Journal of Political Economy. 58 (4): 328–346, 1950

Rank aggregation and ranking queries

- Ronald Fagin, Ravi Kumar, D. Sivakumar
Efficient similarity search and classification via rank aggregation. SIGMOD Conference 2003: 301-312
- Ronald Fagin
Combining Fuzzy Information from Multiple Systems. PODS 1996: 216-226
- Ronald Fagin
Fuzzy Queries in Multimedia Database Systems. PODS 1998: 1-10
- Ronald Fagin, Amnon Lotem, Moni Naor
Optimal Aggregation Algorithms for Middleware. PODS 2001

Skylines and k-Skybands

- Stephan Börzsönyi, Donald Kossmann, Konrad Stocker
The Skyline Operator. ICDE 2001: 421-430
- Jan Chomicki, Parke Godfrey, Jarek Gryz, Dongming Liang
Skyline with Presorting. ICDE 2003: 717-719
- Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger
Progressive skyline computation in database systems. ACM Trans. Database Syst. 30(1): 41-82 (2005)