Middleware Queries

Prof. Paolo Ciaccia
http://www-db.deis.unibo.it/courses/SI-LS/

03_MiddlewareQueries.pdf

Enlarging the scenario

- Now that we know how to solve Top-k queries on a DBMS, it's time to move to consider a more general (and challenging!) scenario
- Our new scenario can be intuitively described as follows

  1. We have a number of "data sources"
  2. Our requests (queries) might involve several data sources at a time
  3. The result of our queries is obtained by "aggregating" in some way the results returned by the data sources

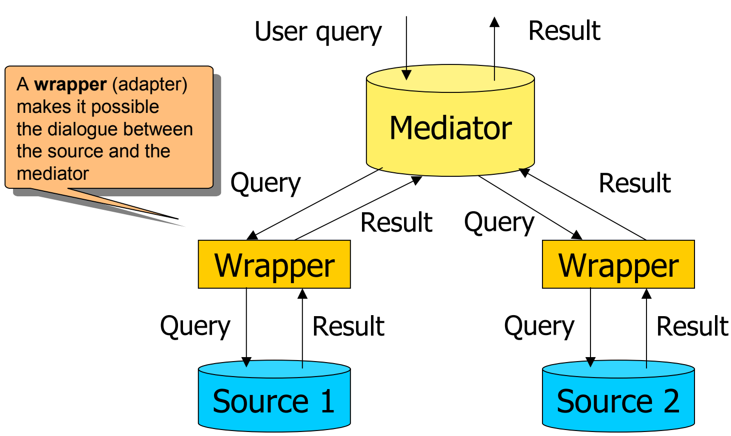
- We call such queries "**middleware queries**" since they necessitate the presence of a "middleware" whose role is to act as a "mediator" (also known as "information agent") between the user/client and the data sources/servers

Data sources

- Sources may be
  - databases (relational, object-relational, object-oriented, legacy, XML)
  - specialized servers (managing text, images, music, spatial data, ecc.)
  - web sites
  - spreadsheets, e-mail archives
  - …
- In several cases, data sources are autonomous and heterogeneous
  - Different data models
  - Different data formats
  - Different query interfaces
  - Different semantics (same query, same data, yet different results)
  - …

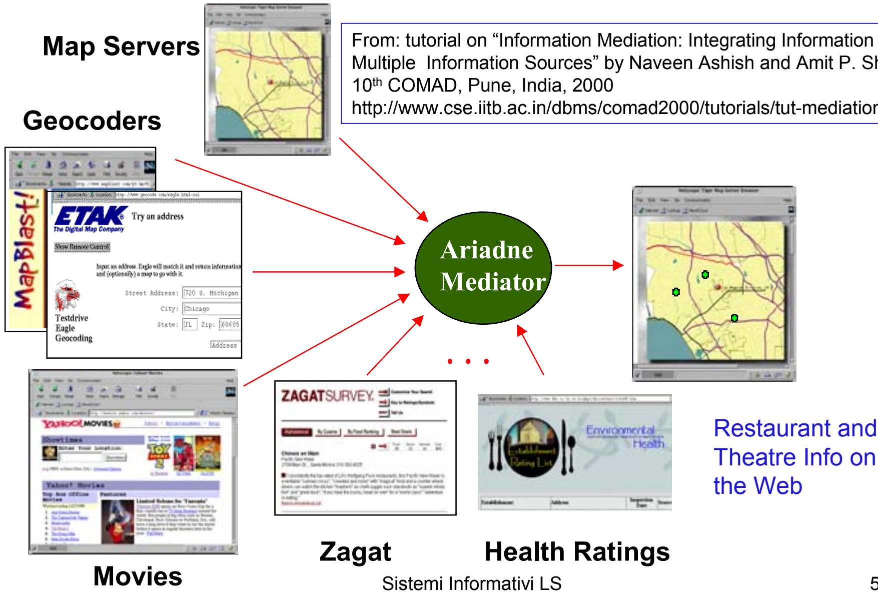
The goal of a mediator is to hide all such differences to the user, so that she can perceive the whole as a single source

The basic architecture



An example

From: tutorial on “Information Mediation: Integrating Information from Multiple Information Sources” by Naveen Ashish and Amit P. Sheth. 10th COMAD, Pune, India, 2000
http://www.cse.iitb.ac.in/dbms/comad2000/tutorials/tut-mediation.ppt



Some links…

- Some projects on mediators (incl. prototypes and software)
  - Ariadne, USC/ISI, http://www.isi.edu/ariadne
  - TSIMMIS, Stanford, http://www-db.stanford.edu/tsimmis/
  - MIX, UCSD, http://feast.ucsd.edu/Projects/MIX/
  - DISCO, U Maryland, http://www.umiacs.umd.edu/labs/CLIP/im.html
  - Garlic, IBM Almaden, http://www.almaden.ibm.com/projects/garlic.shtml
  - Tukwila, U Washington, http://data.cs.washington.edu/integration/tukwila/
  - MOMIS, U Modena e Reggio Emilia, http://dbgroup.unimo.it/Momis/
  - …
- Industrial products/Companies
  - IBM DB2 DataJoiner, http://www-306.ibm.com/software/data/datajoiner/
  - Nimble, http://www.nimble.com
  - Inxight, http://www.inxight.com
  - Fetch, http://www.fetch.com
  - …

Another (simplified) example

- Assume you want to set up a web site that integrates the information of 2 sources:
  - The 1st source “exports” the following schema:

    CarPrices(<u>CarModel</u>, Price)
  - The schema exported by the 2nd source is:

    CarSpec(<u>Make, Model</u>, FuelConsumption)
- After a phase of “reconciliation”

  CarModel = ‘Audi/A4’ ⇔ (Make,Model) = (‘Audi’,‘A4’)

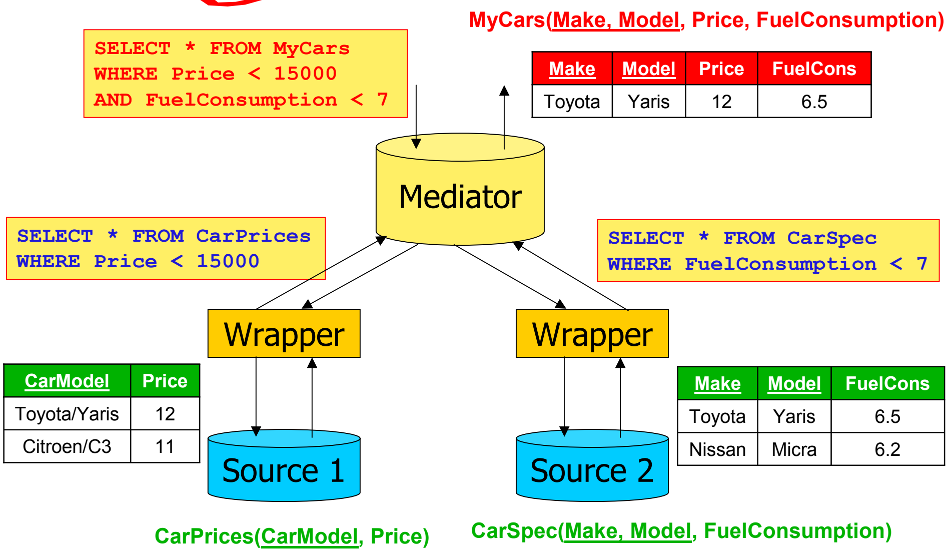
  we can now support queries on both Price and FuelConsumption, e.g.:

  find those cars whose consumption is less than 7 litres/100km and with a cost less than 15,000 €

  How?

  1. send the (sub-)query on Price to the CarPrices source,
  2. send the query on fuel consumption to the CarSpec source,
  3. join the results

The details of query execution



Make	Model	Price	FuelCons
Toyota	Yaris	12	6.5

CarModel	Price
Toyota/Yaris	12
Citroen/C3	11

Make	Model	FuelCons
Toyota	Yaris	6.5
Nissan	Micra	6.2

A further example

- We now want to build a site that integrates the information of (the sites of) m car dealers:
  - Each car dealer site CDj can give us the following information:

    CarDealerj(<u>CarID</u>, Make, Model, Price)

    and our goal is to provide our users with the cheapest available cars, that is, to support queries like:

    For each FIAT model, which is the cheapest offer?

How?

1. send the same (sub-)query to the all the data sources,
2. take the union of the results,
3. for each model, get the best offer and the corresponding dealer

For queries of this kind, the mediator is also often called a “meta-broker” or “meta-search engine”

Query execution (some details omitted)

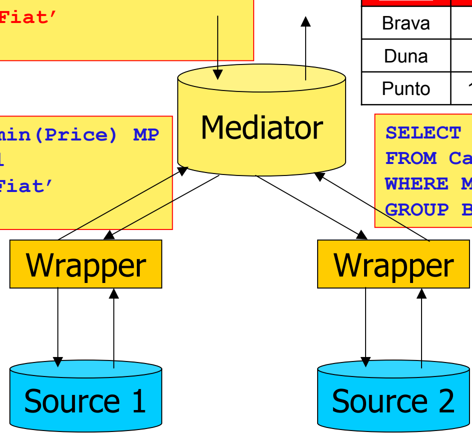
```
SELECT Model,min(Price) MP,Dealer
FROM AllCars
WHERE Make = 'Fiat'
GROUP BY Model
```

AllCars(<u>CarID</u>, Make, Model, Price, Dealer)

Model	MP	Dealer
Brava	8	D1
Duna	7	D2
Punto	10	D2

```
SELECT Model, min(Price) MP
FROM CarDealer1
WHERE Make = 'Fiat'
GROUP BY Model
```

```
SELECT Model, min(Price) MP
FROM CarDealer2
WHERE Make = 'Fiat'
GROUP BY Model
```



Model	MP
Brava	8
Punto	11

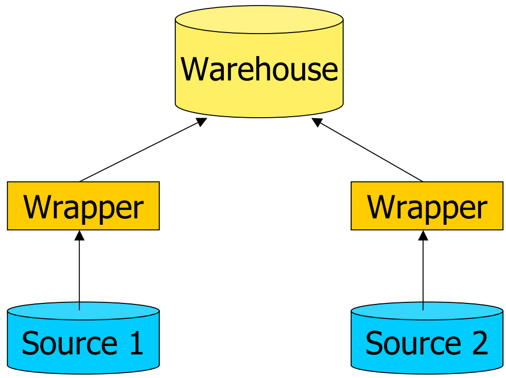
CarDealer1(<u>CarID</u>, Make, Model, Price)

Model	MP
Brava	9
Duna	7
Punto	10

CarDealer2(<u>CarID</u>, Make, Model, Price)

Other possibilities

- With multiple data sources we can have other architectures as well
  - For instance, in a Data Warehouse (DW) all data from the sources are made “homogeneous” and loaded into the global schema of a centralized DW
    - Problems are quite different from the ones we are going to consider…
  - Peer-to-peer (P2P) systems are another relevant case…



The (many) omitted details

- Once one starts to consider a mediator-based architecture, several issues become relevant, e.g.:
  - Which is a suitable query language? A suitable interchange format?
    - Nowadays the answer for the interchange format is: XML
  - Which are the limitations posed by the interfaces of the data sources
    - Can we query using a predicate/filter on the price of cars? On their consumption? Can we formulate queries *at all*?
  - Do we know, say, how a given source ranks objects?
    - E.g., which is the criterion used by Google? and by Altavista?
  - Is there any cost charged by the data sources?
    - Free access? Pay-per-result? Pay-per-query?
- Take also a look at the tutorial by Ashish and Sheth and the links…
- Note that we could make a (much) longer list, and still something would be missing…
- …thus we concentrate on a problem that extends what seen so far…

Top-k middleware queries

- A Top-k middleware query will retrieve the best k objects, given the (partial) descriptions provided for such objects by m data sources
- We make some simplifiying assumptions about our sources
- Relaxing each of these hypotheses leads to slightly different problems (some of them possibly covered by your presentations!)
- We assume that each source:

1. can return, given a query, a ranked list of results (i.e., not just a set)
   - More precisely, the output of the j-th data source DSj (j=1,…,m) is a list of objects/tuples with format

     (ObjID,Attributes,Score)

     where:
     - ObjID is the identifier of the objects in DSj,
     - Attributes are a set of attributes that the query request to DSj
     - Score is a numerical value that says how well an object matches the query on DSj, that is, how "similar" (close) is to our *ideal target object*
     - We also say that this is the "local/partial score" computed by DSj

Random and sorted accesses

2. supports a random access interface:

   $\boxed{\text{getScore}_{DSj}(\mathbf{Q},\text{ObjID}) \rightarrow \text{Score}}$

   A random access retrieves the local score of an object with respect to a query Q

3. supports a sorted access interface:

   $\boxed{\text{getNext}_{DSj}(\mathbf{Q}) \rightarrow (\text{ObjID},\text{Attributes},\text{Score})}$

   A sorted access gets the next best object (and its local score) for a query Q

Some practical issues

- In order to support sorted accesses, one possibility is to use the Next-NN algorithm
- To make things properly work, the concept of "identifier" must be shared among the data sources, that is, they must agree on the identity of an object
  - E.g.: assume we need from DS2 the score of object o25, for which we have already gathered some information from DS1; we must be sure that o25 is indeed the same object in both DS1 and DS2
- This leads us to a 4th assumption:

4. The ObjID is "global": a given object has the same identifier across the data sources
   - In practice this assumption is rarely satisfied (e.g., see our simplified example)
   - The important point is to be able to "match" in some way the descriptions provided by the data sources (see also [WHT+99])

- Further, we also require that each DSj "knows" about a given object:

5. Each source manages a same set of objects

A model with scoring functions

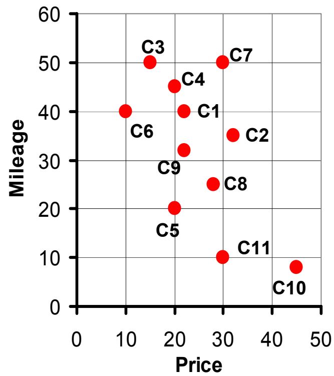
- In order to provide a unifying approach to the problem, we consider:
  - A Top-k query **Q** = (Q1,Q2,…,Qm)
    - Qj is the sub-query sent to the j-th data source DSj
  - Each object o returned by a source DSj has an associated local/partial score sj(o), with sj(o) $\in$ [0,1]
    - Scores are normalized, with higher scores being better
  - The hypercube $[0,1]^m$ is called the "score space"
  - The point s(o) = (s1(o),s2(o),…,sm(o)) $\in [0,1]^m$, which maps o into the score space, is called the "(representative) point" of o
  - The global/overall score gs(o) $\in$ [0,1] of o is computed by means of a **scoring function (s.f.) S** that aggregates the local scores of o:

$$\boxed{S : [0,1]^m \rightarrow [0,1] \qquad gs(o) = S(s(o)) = S(s1(o),s2(o),\ldots,sm(o))}$$

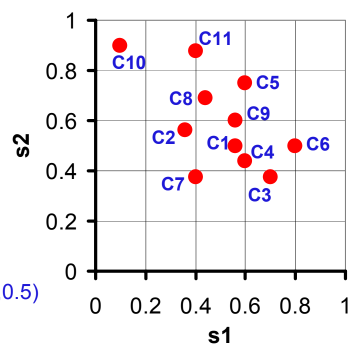
  - If preferences need to be explicitly represented, we can write gs(o;W) and S(s(o);W) or $gs_W(o)$ and $S_W(s(o))$ to make clear that the global score of o depends on W

The “score space”

- Let’s go back to the 2-dimensional (2-D) attribute space **A** = (Price,Mileage)
- Let Q1 be the sub-query on Price, and Q2 the sub-query on Mileage
- For object o we can set: s1(o) = 1 – o.Price/MaxP, s2(o) = 1 – o.Mileage/MaxM
- Let’s take MaxP = 50,000 and MaxM = 80,000
- Thus, objects in A are mapped into the score space as in the figure on the right
  - Note that the relative order (ranking) on each coordinate remains unchanged!



C6:
(10,40) → (0.8,0.5)



Some common scoring functions

- Staying with our example, assume we want to equally weigh Price and Mileage
- We could simply set gs(o) = AVG(s(o)) = (s1(o) + s2(o))/2, i.e., the average of the partial scores
- Doing so, however, we do not consider that partial scores have been normalized
  - In our case this would lead to minimize Price/MaxP + Mileage/MaxM
- Then, in order to minimize Price + Mileage we should use a weighted average:

$$\mathrm{gs(o)} \quad = \mathrm{WAVG(s(o))} = \frac{\mathrm{MaxP} \times \mathrm{s1(o)} + \mathrm{MaxM} \times \mathrm{s2(o)}}{\mathrm{MaxP} + \mathrm{MaxM}} = 1 - \frac{\mathrm{o.Price} + \mathrm{o.Mileage}}{\mathrm{MaxP} + \mathrm{MaxM}}$$

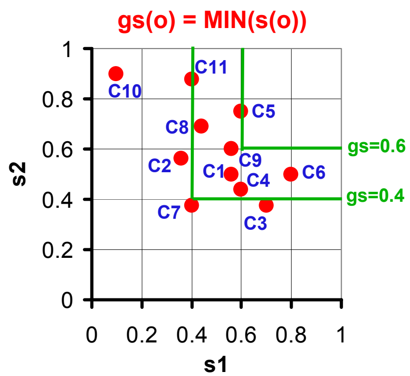
- Besides using a (weighted) average of partial scores, we could also be somewhat more “conservative”, by setting: gs(o) = MIN(s(o)) = MIN{s1(o),s2(o)}

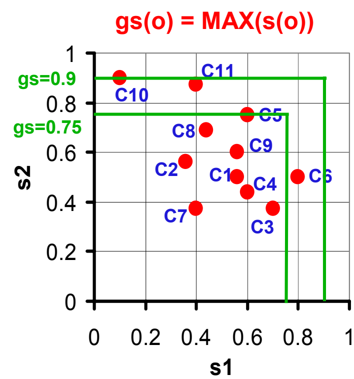
> Remind: (even with MIN) we always want to retrieve the k objects with the highest global scores

- For the “car dealers” example, on the other hand, a suitable scoring function is gs(o) = MAX(s(o)) = MAX{s1(o),s2(o)}

Equallly scored objects

- Similarly to iso-distance curves in an attribute space, we can define iso-score curves in the score space, in order to highlight the sets of points with a same global score





Distance and scoring functions

- It is clear that distances (from a “target point”) and scores are negatively correlated:

  Distance ≡ dissimilarity **Score ≡similarity**

- Assume we want to use a distance function d on **A**

  Can we derive S such that d and S yield the same ranking of objects?

- The answer is trivially “Yes!”, provided:
  1. We know how data sources evaluate partial scores (i.e., the sj() functions)
  2. We get from each DSj the attribute values used to compute the partial scores sj(o)
- The 2nd requirement is indeed necessary

  Example: Let d = (A1 + A2 + A3 + A4)/4, q = (0,0,0,0), and assume that all attribute values are in the range [0,1]

  Let s1(o) = 1 – (o.A1^2 + o.A2^2 + o.A3^2)/3 and s2(o) = 1 – o.A4

  If DS1 does not return at least the values of 2 attributes, there is no way to define S with the same behavior of d!

Non-cooperative data sources

- If the mediator ignores some aspects concerning how data sources compute local scores, it might not be possible to rank objects exactly as needed
- Further, there are other “problematic” scenarios (<u>not exactly fitting our model</u>)
  - This can impact both the efficiency and the correctness of the solution (see also [GG97] for the case of a single source and [YPM03])

Example (adapted from [GG97]):

- consider a query that wants to rank houses using the scoring function S = 0.5*sGarden + 0.5*sBedrooms, where sGarden and sBedrooms are both score values in [0,1] (higher values are better)
- The query is submitted to a mediator that searches, within a <u>single data source DS</u>, the “best” house according to S
- However, DS ranks houses **always** using S’ = 0.2*sGarden + 0.8*sBedrooms, that is, DS weighs more a good match on bedrooms than on the garden area
- Assume that DS has a house h with sGarden(h) = 1, sBedrooms(h) = 0.4, thus S(1,0.4) = 0.7 and S’(1,0.4) = 0.52
- Also assume that all the other houses h’ of DS have sGarden(h’) = 0.6, sBedrooms(h’) = 0.6, thus S(0.6,0.6) = 0.6 and S’(0.6,0.6) = 0.6
- It follows that **h is the best house for S, and the worst one for S’!!**

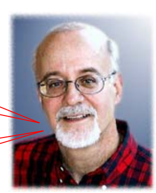
The simplest case: MAX

- Going back to our model, it’s time to ask “the big question”:

How can we compute the Top-k results, according to a scoring function S, of a middleware query **Q**?

- For the particular case S ≡ MAX the solution is really simple [Fag96]:

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

How B_0 works

1. For each data source DSj execute k sorted accesses (i.e. $getNext_{DSj}(\mathbf{Q})$)
2. Let Obj(j) be the set of objects returned by the j-th source
   - Thus, Obj(j) consists of the k objects with maximum values of sj
3. Let $Obj = \cup_j Obj(j)$ be the union of all such results
4. For each object $o \in Obj$ compute gs(o) as the maximum over all the available partial scores
   - Note that some partial scores for o might be missing if o is not one of the Top-k objects for a sub-query
5. Return the k objects with the highest global scores

ObjID	s1
o7	0.7
o3	0.65
o4	0.6
o2	0.5

ObjID	s2
o2	0.9
o3	0.6
o7	0.4
o4	0.2

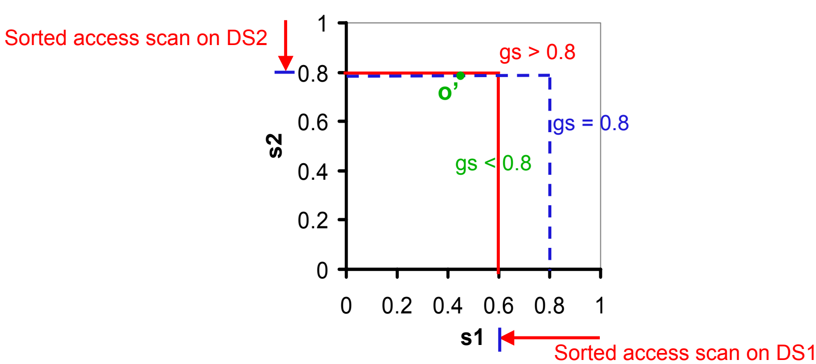
ObjID	s3
o7	1.0
o2	0.8
o4	0.75
o3	0.7

ObjID	gs
o7	**1.0**
o2	**0.9**
o3	0.65

k = 2

Why B_0 works: graphical intuition

- By hypothesis, in the figure we have at least k objects o with $gs(o) \geq 0.8$
  - This holds because at least one sorted access scan (on DS2, in the figure) stops after retrieving at the k-th step an object with local score = 0.8
- An object, like o', that has not been retrieved by any sorted access scan (thus $o' \notin Obj$), cannot have a global score higher than 0.8!



Why B_0 works: a tricky aspect (1)

- Let Res be the set of Top-k objects computed by B_0 (Res $\subseteq$ Obj)
- As seen, if o $\notin$ Obj then o cannot be better than any object in Res
- Due to the semantics of Top-k queries we need to show that:

  1. There are no o' $\notin$ Res, o $\in$ Res s.t. gs(o') > gs(o) (i.e., Res is correct)
  2. For each object o $\in$ Res, the algorithm correctly computes gs(o)

- The tricky point is that we have evidence that

  if o $\in$ Obj – Res, then it is not guaranteed that gs(o) is correct

  (e.g., see o3 in the example)

Are we missing some information
that is relevant to determine the result?

NO!

Why B_0 works: a tricky aspect (2)

- We first show that **if o $\in$ Res, then gs(o) is correct**
  - Let $gsB_0(o)$ be the global score, as computed by B_0, for an object o $\in$ Obj
  - Clearly $gsB_0(o) \leq gs(o)$ (e.g., $gsB_0(o3) = 0.65 \leq gs(o3) = 0.7$)

  - Let o2 $\in$ Res and assume by contradiction that $gsB_0(o2) < gs(o2)$
  - This is also to say that there exists DSj s.t. o2 $\notin$ Obj(j) and gs(o2) = sj(o2)
  - In turn this implies that there are k objects o $\in$ Obj(j) s.t.

    $\mathbf{gsB_0(o2)} < gs(o2) = sj(o2) \leq sj(o) \leq \mathbf{gsB_0(o)} \leq gs(o) \qquad \forall o \in Obj(j)$
  - Thus o2 cannot belong to Res, a contradiction ■

ObjID	sj
….	….
o	**sj(o)** $\leq gsB_0(o) \leq gs(o)$
…	…
…	…
o2	$gsB_0(o2) < gs(o2) =$ **sj(o2)**

Obj(j) contains k objects

?? Impossible when o2 $\in$ Res

Why B_0 works: a tricky aspect (3)

- Now we show that **if o ∈ Obj - Res, then, even if $gsB_0(o) < gs(o)$, the algorithm correctly computes the Top-k objects**
  - Consider an object, say o3, s.t. o3 ∈ Obj – Res
  - If $gsB_0(o3) = gs(o3)$ then there is nothing to demonstrate ☺
  - On the other hand, assume that at least one partial score of o3, sj(o3), is not available, and that $gsB_0(o3) < gs(o3) = sj(o3)$. Then

$$gsB_0(o3) < \mathbf{gs(o3)} = sj(o3) \leq sj(o) \leq \mathbf{gsB_0(o)} \leq \mathbf{gs(o)} \qquad \forall o \in Obj(j)$$

  - Since each object in Res has a global score at least equal to the lowest score seen on the objects in Obj(j), it follows that it is impossible to have gs(o3) > gs(o') if o' ∈ Res ■

ObjID	sj
….	….
o	**sj(o)** $\leq gsB_0(o) \leq gs(o)$
…	…
…	…
o3	$gsB_0(o3) < gs(o3) =$ **sj(o3)**

Obj(j) contains k objects

Impossible to have gs(o3) > gs(o'), o' ∈Res

Why B_0 **doesn't** work for other scoring f.'s

- Let's take S ≡ MIN and k = 1
- We apply B_0 to the following data:

ObjID	s1
o7	0.9
o3	0.65
o2	0.6
o1	0.5
o4	0.4

ObjID	s2
o2	0.95
o3	0.7
o4	0.6
o1	0.5
o7	0.5

ObjID	s3
o7	1.0
o2	0.8
o4	0.75
o3	0.7
o1	0.6

and obtain:

ObjID	gs
o2	**0.95**
o7	0.9

WRONG!!

- Ok, we are clearly wrong, since we are not considering ALL the partial scores of the objects in Obj (Obj = {o2,o7} in the figure) ☺
- Then, we can perform random accesses to get the missing scores:

$$getScore_{DS1}(\mathbf{Q},o2),\ getScore_{DS3}(\mathbf{Q},o2),\ getScore_{DS2}(\mathbf{Q},o7)$$

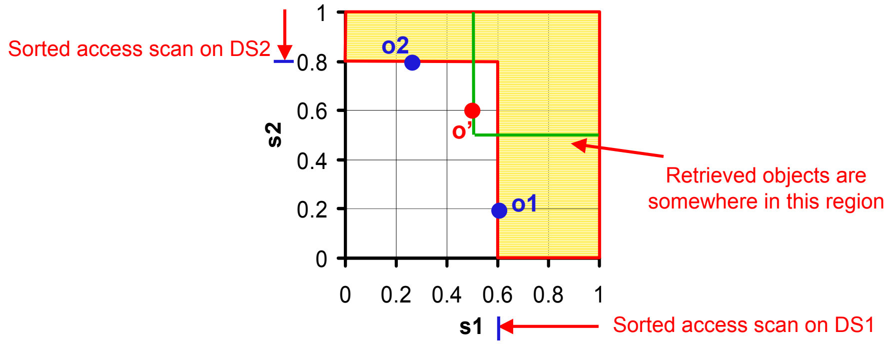
and obtain:

ObjID	gs
o2	**0.6**
o7	0.5

STILL WRONG!!? ☹

Why B_0 **doesn't** work: graphical intuition

- Let's take S ≡ MIN and k = 1
- When the sorted accesses terminate, we don't have any lower bound on the global scores of the retrieved objects (i.e., it might also be gs(o) = 0!)
- An object, like o', that has not been retrieved by any sorted access scan can now be the winner!
- Note that, in this case, o' would be the best match even for S ≡ AVG

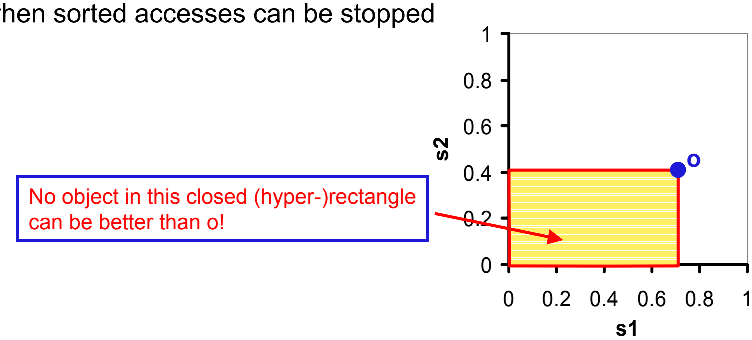


The A_0 algorithm: monotone scoring f.'s

- The A_0 algorithm [Fag96] solves the problem for **any** monotone s.f.:

Monotone scoring function:
- An m-ary scoring function S is said to be monotone if
  $x1 \leq y1, x2 \leq y2, \ldots, xm \leq ym \quad \Rightarrow \quad S(x1,x2,\ldots,xm) \leq S(y1,y2,\ldots,ym)$

- A_0 exploits the monotonicity property in order to understand when sorted accesses can be stopped



The A_0 algorithm

ObjID	s1
o7	0.7
o3	0.5
…	…

ObjID	s2
o3	0.8
o2	0.7
…	…

- A_0 works in 3 distinct phases: k = 1

1. Sorted access phase
   - Perform on each DSj a sequence of sorted accesses, and stop when the set **M = $\cap_j$ Obj(j)** contains at least k objects
2. Random access phase
   - For each object o $\in$ Obj = $\cup_j$ Obj(j) perform random accesses to retrieve the missing partial scores for o
3. Final computation
   - For each object o $\in$ Obj compute gs(o) and return the k objects with the highest global scores

M = {o3}
Obj = {o2,o3,o7}

↓

getScore$_{DS1}$(**Q**,o2)
getScore$_{DS2}$(**Q**,o7)

↓

compute top-k results according to S

How A_0 works

- Let's take k = 1
- Now we apply A_0 to the following data:

ObjID	s1
o7	0.9
o3	0.65
o2	0.6
o1	0.5
o4	0.4

ObjID	s2
o2	0.95
o3	0.7
o4	0.6
o1	0.5
o7	0.5

ObjID	s3
o7	1.0
o2	0.8
o4	0.75
o3	0.7
o1	0.6

and after the sorted accesses obtain:

M = {o2}
Obj = {o2,o3,o4,o7}

- After performing the needed random accesses we get:

S ≡ MIN

ObjID	gs
o3	**0.65**
o2	0.6
o7	0.5
o4	0.4

RIGHT!!

☺

S ≡ AVG

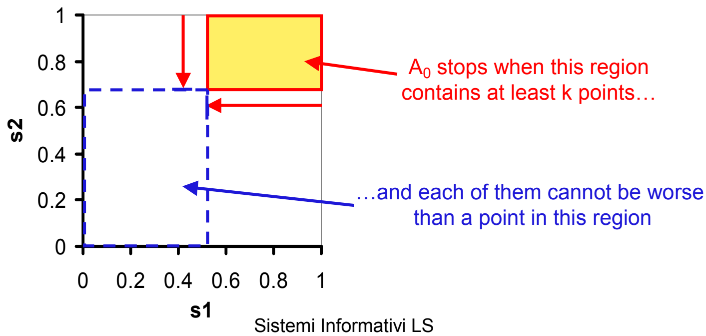
ObjID	gs
o7	**0.8**
o2	0.783
o3	0.683
o4	0.583

RIGHT!!

Why A_0 is correct: formal and intuitive

- The correctness of A_0 follows from the assumption of monotonicity of S

Proof: Let Res be the set of objects returned by the algorithm.
It is sufficient to show that
if o' $\notin$ Obj, then o' cannot be better than any object o $\in$ Res.
Let o be any object in Res. Then, there is at least one object o'' $\in$ M for which it is gs(o'') $\leq$ gs(o), otherwise o would not be in Res.
Since o' $\notin$ Obj, for each DSj it is sj(o') $\leq$ sj(o''), and from the assumption of monotonicity of S it is gs(o') $\leq$ gs(o''); it follows that gs(o') $\leq$ gs(o) ■



A_0: performance and optimality

- When the sub-queries are independent (i.e., ranking on Qi is independent of the ranking on Qj) it can be proved that the cost of A_0 (no. of sorted and random accesses) for a DB of N objects is, with arbitrarily high probability:

$$O\left(N^{(m-1)/m}\, k^{1/m}\right)$$

- This also represents a lower bound on the cost of any algorithm when S is *strict*, that is:

$$S(x1,x2,\ldots,xm) = 1 \iff x1 = 1, x2 = 1, \ldots, xm = 1$$

- Note that MIN and AVG are strict, whereas MAX is not
- In this sense A_0 is optimal, which means that any algorithm can only improve over A_0 by only a constant factor
- …however the next algorithm we see is even better (!?)

Instance optimality

- Although A_0 is optimal (in a high-probability sense) for strict monotone scoring functions, it is evident that, for a given DB, its cost is always the same, regardless of S!
- Intuitively, this is because A_0 does not consider S until the final step, when global scores are to be computed
- Fagin, Lotem, and Naor [FLN01,FLN03] have derived another algorithm, called TA (Threshold Algorithm), which is optimal in a much stronger sense than A_0, namely TA is instance optimal:

Instance optimality:

- Given a class of algorithms **A** and a class **D** of DB's (inputs of the algorithms), an algorithm A $\in$ **A** is instance-optimal over **A** and **D** if for every B $\in$ **A** and every DB $\in$ **D** it is

  cost(A,DB) = O(cost(B,DB))

- Thus, unlike A_0, TA can "adapt" to what it sees (the specific DB at hand)
- We take cost = no. of sorted and random accesses, but other definitions are possible as well

The TA algorithm

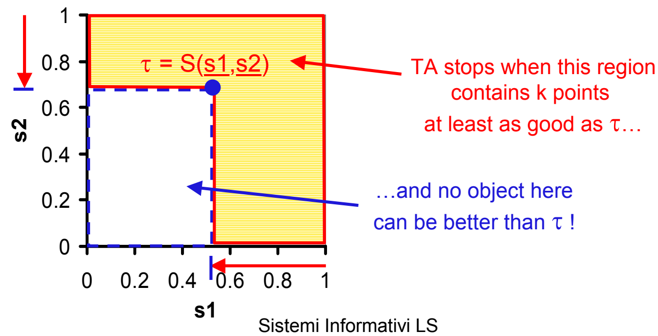
- TA works by interleaving sorted and random accesses:

1. Perform on each DSj a sorted access;
   for each new object o seen under sorted access, perform random accesses to retrieve the missing partial scores for o and compute gs(o);
   If gs(o) is one of the k highest scores seen so far keep (o,gs(o)), otherwise discard o and its score
2. Let sj be the lowest score seen so far on DSj;
   Let τ = S(s1,s2,…,sm) be the *threshold score*
3. If the current Top-k objects are such that for each of them gs(o) $\geq \tau$ holds, then stop, otherwise repeat from 1.

Why TA is correct: formal and intuitive

- The correctness of TA still follows from the assumption of monotonicity of S

Proof: Let Res be the set of objects returned by TA.
As with A_0 it is sufficient to show that
if o' $\notin$ Obj, then o' cannot be better than any object o $\in$ Res.
Since o' has not been seen under sorted access, for each j it is sj(o') $\leq$ <u>sj</u>.
Due to the monotonicity of S this implies gs(o') $\leq \tau$.
By definition of Res, for each object o $\in$ Res it is gs(o) $\geq \tau$, thus gs(o') $\leq$ gs(o) ■



How TA works

- Let's take S ≡ MIN and k = 1

ObjID	s1	ObjID	s2	ObjID	s3
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

gs(o2) = 0.6; gs(o7) = 0.5. **τ = 0.9**
gs(o3) = 0.65. τ = 0.65

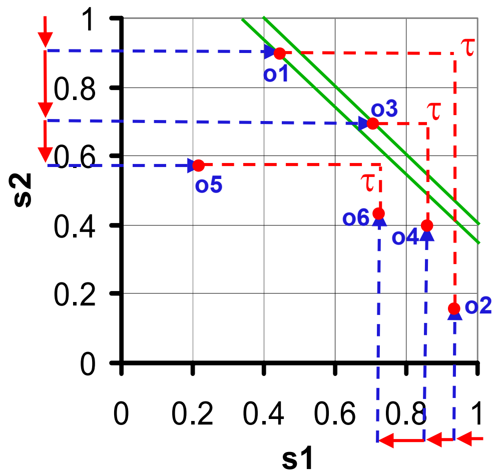
- Let's take S ≡ AVG and k = 2

ObjID	s1	ObjID	s2	ObjID	s3
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

gs(o2) = 0.783; **gs(o7) = 0.8**. **τ = 0.95**
gs(o3) = 0.683. **τ = 0.716**

The geometric view

- Let's take S ≡ AVG and k = 2



gs(o3) >gs(o1) > τ
Stop!

Main facts about TA

- TA is instance optimal over all DB's and over all "reasonable" algorithms
- More precisely, TA is instance optimal with respect to (w.r.t.) all algorithms that do not make (lucky) "wild guesses":

An algorithm A makes wild guesses if it makes a random access for object o without having seen before o under sorted access

- Note that algorithms making wild guesses are only of theoretical interest
- Also observe that instance optimality is a much stronger notion than optimality in the average or worst case
  - E.g., binary search is optimal in the worst case, but it is not instance optimal
- A further important observation about TA is that, unlike A_0, it only requires O(k) space in main memory to buffer the current Top-k results

Going beyond TA

- Besides TA, [FLN01] considers other algorithms that are suitable for different scenarios:

NRA (No Random Accesses): this applies when random accesses are impossible

  - E.g., Web search engines do not support the getScore () interface

CA (Combined Algorithm): this takes into account the case when the "costs" of sorted and random accesses are different

- More recently, [BGM02,BGM04] have introduced Upper, which aims to minimize the number of random accesses



Upper is especially suited to Web-accessible DB's; in [BGM04] we also study parallel algorithms to reduce response times over Internet sources

What else?

- Several aspects related to the processing of Top-k middleware queries are still, as of 2004, active areas of research
- These include:
  - Providing best-matching results for mobile devices (palmtops, PDA's, etc.)
  - Managing the case of "incomplete information" (not all scores are available)
  - Precomputation/caching of results to speed-up subsequent queries
  - Trading-off completeness of results for speed of execution (i.e., approximate queries)
  - …

Any other idea?