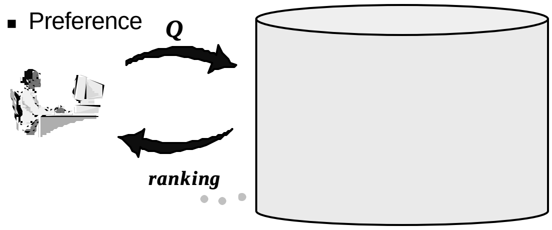
Ranked Query Processing: a) Order-based Paradigm

Kevin Chen-Chuan Chang

The Database and Information Systems Laboratory
at The University of Illinois at Urbana-Champaign
Large Scale Information Management

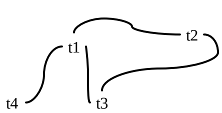
Ranking– Ordering according to the degree of some fuzzy notions:

- Similarity (or dissimilarity)
- Relevance
- Preference



Query models for order-based paradigm– On the better-than graph

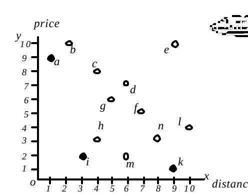
- Better-than graph

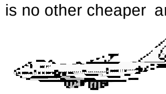


- Best-Matches-Only (BMO) query model
  - Retrieve maximal elements
  - Thus also called maximal vector
- These maximal elements form the “skyline”!
- On better-than graph, how to process BMO?

When multiple dimensions are available--

- Assume the database stores the information of a set of flights
- For each flight
  - Its price
  - Its route (travel-time or distance traveled)
- A user would retrieve all the “interesting” flights
  - A flight is interesting if and only if there is no other cheaper and shorter (route) at the same time





The overall preference combines the dimensions

- P1 LOWEST(price)
  - *a*→ *b*→ *i*→ *c*, *h*→ *g* → *d*, *m*→ *f* → *n* → *k*, *e*→ *l*
- P2 LOWEST(distance)
  - *k* → *m*, *i* → *h*, *n* → *l* → *f* → *g* → *d* → *c* → *a* → *b*, *e*
- P :=({price,distance},<P1ÄP2)
- BMO: Maximal elements of P?
  - Is *a* maximal?
  - Is *b* maximal?
  - Is *c* maximal?

	Distance	Price
a	1	9
b	2	10
c	4	8
d	6	7
e	9	10
f	7	5
g	5	6
h	4	3
i	3	2
k	9	1
l	10	4
m	6	2
n	8	3

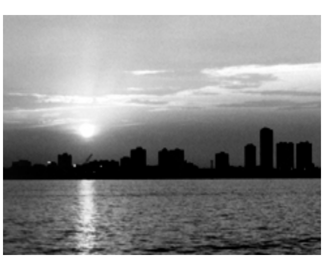
Skyline Operation

- Dominance:
  - A point dominates another point if it is *no worse* in all dimensions, and *better* in at least one dimension
- Skyline:
  - A set of all points in the dataset that are *not dominated* by any other point in the dataset

Why is it called “skyline”?

(Also called: Pareto curve, Maximum Vector)

- What do you see in the Chicago skyline?

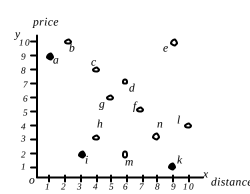


What is skyline: An example

- Query:

  SELECT * FROM flights

  SKYLINE OF price MIN, distance MIN
- What dominates what?
- What points constitute the skyline?



Skyline Algorithms: We will look at a few examples

- Block nested loop (BNL)
- Divide and Conquer
- Bitmap
- NN

Block Nested Loop [Börzsönyi et al., 2001]

- Conceptually: Nested loop joins—
  - Joining the table with itself
  - Compare every pair of points to check dominance

	Price	Distance
a	1	9
b	2	10
c	4	8
d	6	7
e	9	10
f	7	5
g	5	6
h	4	3
i	3	2
k	9	1
l	10	4
m	6	2
n	8	3

	Price	Distance
a	1	9
b	2	10
c	4	8
d	6	7
e	9	10
f	7	5
g	5	6
h	4	3
i	3	2
k	9	1
l	10	4
m	6	2
n	8	3

Block Nested Loop -- Implementation

- One-pass scan:
  - Scan the table; maintain a *window* of current skyline points
  - Return the window at the end

Scan

	Price	Distance	Skyline	Discarded
a	1	9	a	
b	2	10	**a**	b
c	4	8	a,c	
d	6	7	a,c,d	
e	9	10	**a**,c,d	e
f	7	5	a,c,d,f	
g	5	6	a,c,f, **g**	d
h	4	3	a, **h**	c,f,g
i	3	2	a, **i**	h
k	9	1	a,i,k	
l	10	4	a, **i** ,k	l

- Any problems?

Block Nested Look– Improvements
How if the window overflow?

- Multi-pass algorithm
  - Scan the table, write any overflow to temp file
  - Scan the temp file; repeat till done

Scan ***Pass 1***

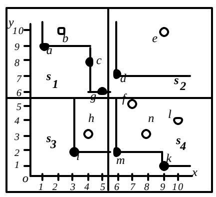
	Price	Distance	Skyline	Discarded	TempFile
a	1	9	a		
b	2	10	**a**	b	
c	4	8	a,c		
d	6	7	a,c,d		
e	9	10	**a**,c,d	e	
f	7	5	a,c,d		f
g	5	6	a,c,**g**	d	
h	4	3	a,**h**	c,g	
i	3	2	a,**i**	h	
k	9	1	a,i,k		
l	10	4	a, **i** ,k	l	

Pass 2

Scan TempFile

Block Nested Look – Improvements
How if the window overflow? [Börzsönyi et al., 2001]

- Divide and conquer
  - Divide all the points into several groups such that each group fits in memory
  - Process the groups separately
  - Merge their results
- Smart merging possible
  - If s3 not empty then disregard s2
  - Use s3 to purge s1, s4



13

However, BNL-based approaches are not incremental – Want progressive processing!

Desired:

- Compute the first few Skyline points almost instantaneously
- Compute more and more results incrementally

14

Bitmap Algorithm: Representation [Tan et. al. 2001]

- For each dimension:
  - n distinct values → n bits
  - A value as a bitmap of all no-higher bits = 1

		d1: price				***d2: dist***			***d3: rating***	
		4	**3**	**2**	**1**	**3**	**2**	**1**	**2**	**1**
a	(1,1,2)	0	0	0	1	0	0	1	1	1
b	(3,2,1)	0	1	1	1	0	1	1	0	1
c	(4,1,1)	1	1	1	1	0	0	1	0	1
d	(2,3,2)	0	0	1	1	1	1	1	1	1

15

Is b = (3, 2, 1) in the skyline?

- Any point with no-worse values in all dimensions?
  - 0110 & 0101 & 1111 = 0100
- Any point with a better value in some dimension?
  - 0010 | 0001 | 1001 = 1011
- Any point satisfying both?
  - 0100 & 1011 = 0000
- So, is b = (3,2,1) in the skyline?

		d1: price				***d2: dist***			***d3: rating***	
		4	**3**	**2**	**1**	**3**	**2**	**1**	**2**	**1**
a	(1,1,2)	0	0	0	1	0	0	1	1	1
b	(3,2,1)	0	1	1	1	0	1	1	0	1
c	(4,1,1)	1	1	1	1	0	0	1	0	1
d	(2,3,2)	0	0	1	1	1	1	1	1	1

16

The Bitmap Algorithm

- for each point *x* in DB:
  - check if *x* is in skyline
  - output *x* if so
- Incremental indeed; bitmap computation efficient
- However, any problem?

17

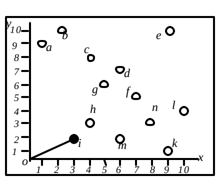
Bitmap Algorithm: Problems

- Bitmaps are not dynamic structures
  - Hard to update
- Bitmaps can have prohibitive space overhead
  - How if there are many distinct values?
    - E.g., How about continuous values?
- No focus of directions at all in skyline search
  - Depend on what points you check first

18

NN – Finding the First Skyline Point [Kossmann et. al. 2002]

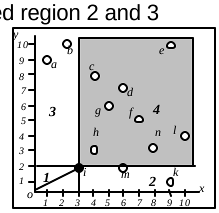
- Start by finding the nearest neighbor of the origin
  - I.e., the point $p = (x, y)$ with the smallest $dist(o, p) = \sqrt{x^2 + y^2}$
  - How to find NN: Use NN algorithm based on R-tree.
- This NN point must be in the skyline
  - Otherwise?



19

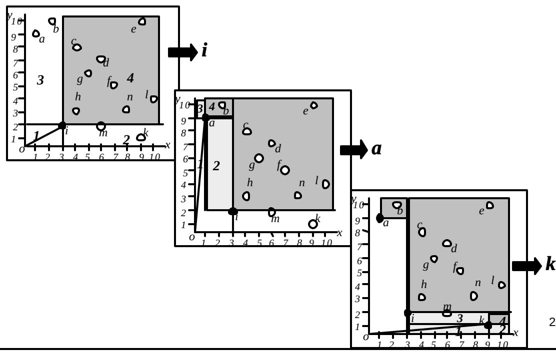
NN– Are there other skyline points?

- Pruning -- What cannot be in the skyline?
  - Those dominated by point *I*
- Iteration – What may be in the skyline?
  - Non-dominated region 2 and 3



20

NN– Iteratively Process All the "ToDo" Regions until All Done



Order-based rank query evaluation-- Still ongoing research.

- How optimal are these algorithms? Further improvement?
- Scale to high dimensionality?
- Generalize to non-BMO type of aggregations?

Thank You!

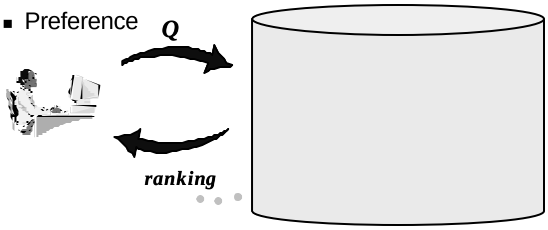
Ranking Query Processing: b) Score-based Paradigms

Kevin Chen-Chuan Chang

The Database and Information Systems Laboratory
at The University of Illinois at Urbana-Champaign
Large Scale Information Management

Ranking– Ordering according to the degree of some fuzzy notions:

- Similarity (or dissimilarity)
- Relevance
- Preference



25

Relational DBMS scenarios– A brief overview

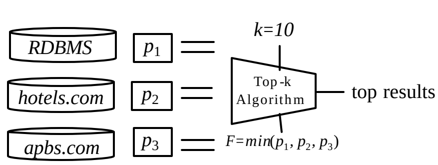
Relational DBMS–

- Value mapping: [Chaudhuri and Gravano, 1999]
  - Mapping top-k scores to Boolean selection ranges
  - May have to restart
- Cardinality mapping: [Carey and Kossmann, 1997, 1998]
  - Pushing “limit k” down query tree
  - May have to restart

26

Our Focus: Middleware scenarios

select h.id, h.address **from** Hotel h
order by $F=min(p_1{:}rating$(h.rate), $p_2{:}cheap$(h.price), $p_3{:}safe$(h.zip))
stop after $k=10$

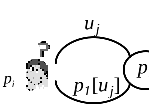


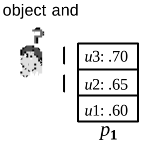
27

Top-k algorithms rely on accesses to evaluate query scores

To each predicate p_i:

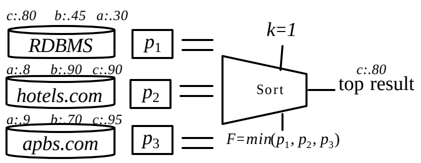
- *Random access:* $ra_i(u_j)$
  - Return score of u_j for p_i
- *Sorted access:* sa_i
  - Return some next best object and its score for p_i



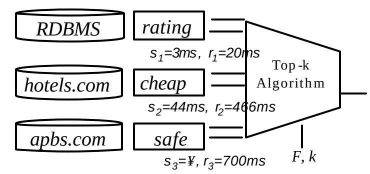


An algorithm performs a sequence of accesses: A simple algorithm

- Sorted access on P1 then random accesses to P2, P2



Goal: Minimize the "access" cost



Access costs dominate in "middleware" scenarios
→ Cost model: aggregate of all access costs

Assumption: Monotonic scoring functions

- *Monotonic:*
  - $f(x_1, \ldots, x_n) \le f(x_1', \ldots, x_n')$ if $x_i \le x_i'$ for all i
- Why good for query evaluation?
  - Gives bounds for pruning data
  - Gives a simple function "surface" to maximize f
- Reasonable?
  - Analogy: Negation rarely used in Boolean queries
  - But, new "function-inference" front-ends also found this to be violated in many cases

The Naïve Algorithm

- Get all $p_i[u]$ score for every object u
  - e.g., by complete sorted accesses
- Compute $F[u] = F\ (p_1[u], \ldots, p_m\ [u])$ for every u
- Sort
- Return top k
- Obviously expensive. Can we do better?
  - Note k is typically small

FA– *Fagin’s Algorithm* (or the “First Algorithm”) [Fagin, 1996] [Wimmers et al., 1999]

Scenario: Sorted + Random Access Available

- Go in the lists with SA in parallel
- Do complete RA for every seen object to complete scores
- Maintain a buffer of current top-k objects
- Maintain a threshold ***T***:
  - Upper-bound for all the unseen objects
- Stop:
  - When all lists so far share at least *k* objects
- Return the current top-k objects

FA– Fagin’s Algorithm

- Scoring function: $F = p_1 + p_2$

ID	p_1
5	.50
1	.40
3	.30
2	.20
4	.10

ID	p_2
3	.50
2	.40
1	.30
4	.20
5	.10

Intersection

{}
{}
{1, 3}

3: (.80)
1: (.70)
5: (.60)
2: (.60)

Buffer

Why is FA correct?

- At stop time, all seen objects are compared
- Can unseen objects have higher scores?
  - e.g., How about object 4? Upper bound?

ID	p_1
5	.50
1	.40
3	.30
2	.20
4	.10

ID	p_2
3	.50
2	.40
1	.30
4	.20
5	.10

Intersection

{}
{}
{1, 3}

3: (.80)
1: (.70)
5: (.60)
2: (.60)

Buffer

How is FA “optimal”? Can you make it more efficient?

- FA:
  - For string, monotone F, sorted accesses optimal up to a constant factor, with high probability.
- Can you stop earlier than round 3?

ID	p_1
5	.50
1	.40
3	.30
2	.20
4	.10

ID	p_2
3	.50
2	.40
1	.30
4	.20
5	.10

Intersection

{}
{}
{1, 3}

3: (.80)
1: (.70)
5: (.60)
2: (.60)

Buffer

Then, there have been various algorithms, for different scenarios...

Sorted Access	*Random Access* *r* =1 (*cheap*)	*r* = *h* (*expensive*)	*r* = ∞ (*impossible*)
s =1 (*cheap*)	FA, TA, QuickCombine	CA, SR-Combine	NRA, StreamCombine
s = *h* (*expensive*)		FA, TA, QuickCombine	NRA, StreamCombine
s = ∞ (*impossible*)	TAz, MPro, Upper	TAz, MPro, Upper	

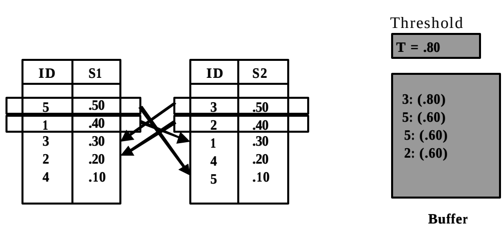
Improving FA: TA [Fagin et al., 2001], Quick-combine [Guentzer et al., 2000], Multi-step [Nepal and Ramakrishna, 1999]

Scenario: Sorted + Random Access Available

- Go in the lists with SA in parallel
- Do complete RA for every seen object to complete scores
- Maintain a buffer of current top-k objects
- Maintain a threshold ***T***:
  - Upper-bound for all the unseen objects
- Stop:
  - When all current top-k objects scored greater than ***T***
- Return these objects as top-k

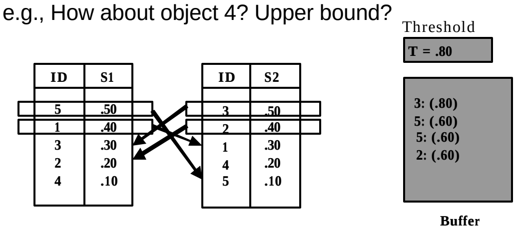
TA, Quick-combine, Multi-step

$F = p_1 + p_2$



Why is TA correct?

- At stop time, all seen objects are compared
- Can unseen objects have higher scores?
  - e.g., How about object 4? Upper bound?



Observations: Any Problem with TA?

- How does it handle SA?
  - Equal-depth parallel SA to every list
- How does it handle RA?
  - Exhaustive RA for every seen object
  - How if RA expensive? (Algorithm CA)
  - How if RA not possible? (Algorithm NRA)

41

How if random accesses not supported?

- The combined score of an object has two parts:
  - Upper bound score:
    - From seen exact scores and unseen max score
  - Lower bound score:
    - From seen exact scores and unseen min score
- An object is in top-k if
  - Its lower bound score is greater than the upper bound scores of all unseen objects

42

NRA [Fagin et al., 2001], Stream-combine [Guentzer et al., 2001] -- When random accesses not possible

- Scoring function: $F = p_1 + p_2$

ID	S1
5	.50
1	.40
3	.30
2	.20
4	.10

ID	S2
3	.50
2	.40
1	.30
4	.20
5	.10

3: (.80 – .80)
1: (.70 – .70)
5: (.50 – .80)
2: (.40 – .70)

Buffer

43

In contrast, how if sorted accesses not possible?

Scenario: When SA not supported

- Perform random "probes" when necessary
  - The object with current highest score
- Schedule predicates to minimize probes
- Return an object as top-k when
  - It is fully probed
  - Its score is higher than the (upper bounds of) the rest not in top-k

44

MPro [Chang and Hwang, 2002], Upper [Bruno et. al. 2002] – When sorted accesses not possible

Upper-bound of the unseen scores

U_{unseen} = $\boxed{0.7}$

ID	x	p_1	p_2	*Min*(x, p_1, p_2)
a	0.90	85	.75	.75
b	0.80	78	.90	.78
c	0.70	.75	.20	.20
d	0.60	.90	.90	.60
e	0.50	.70	.80	.50

b: 0.78
a: 0.75
c: 0.7

Candidates Queue

Probe optimization– Is the cost of random probes minimal?

- What object to probe next?
  - By *necessary probes* to analytically determine [Chang and Hwang, 2002]
    - Current top object *must* be further probed (by any algorithm)
- For such object, what predicate to probe next?
  - MPro: Global scheduling – one schedule for all
    - Cost-based optimization based on selectivity and cost
  - Upper: Local scheduling – schedule for each obj
    - Use expected scores of unknown objects

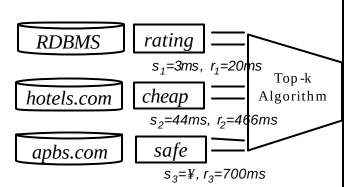
So, what do we have so far…

Sorted Access	*Random Access*		
	r =1 (*cheap*)	*r* = *h* (*expensive*)	*r* = ∞ (*impossible*)
s =1 (*cheap*)	FA, TA, QuickCombine	CA, SR-Combine	NRA, StreamCombine
s = *h* (*expensive*)		FA, TA, QuickCombine	NRA, StreamCombine
s = ∞ (*impossible*)	TAz, MPro, Upper	TAz, MPro, Upper	

☞ *What do you think?*

Challenge: Various Cost Scenarios

- Vary in *capabilities*
- Vary in *costs*:
  - over sources
  - over access types
  - over time



☞ *Thus requires "generality" over cost scenarios and "adaptivity" to the given runtime setting*

Score-based ranked query evaluation – Still ongoing research

- A unified algorithms for all?
  - Currently: ad-hoc algorithms for each scenario
  - Do not cover all scenarios
- How optimal are these algorithms?
  - Cost-based optimization studied at MPro
- Unified, cost-based optimization?

Thank You!