Note Title 3/6/2014

Preferences

Domain

Proc

R

A_1	A_2
I7	250GB
I3	1TB
I7	500GB

1TB

$\vee$

750GB

$\vee$

500GB

$\vee$

250GB

I7

I5

AMD

I3

AMD

Pent

Note Title 3/6/2014

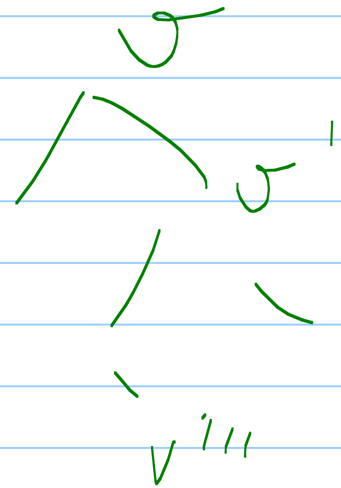
How do you formulate this problem?

$A_1 \quad \ldots \quad A_n$ $\qquad$ $dom(A_i)$

D_1

$\vdots$

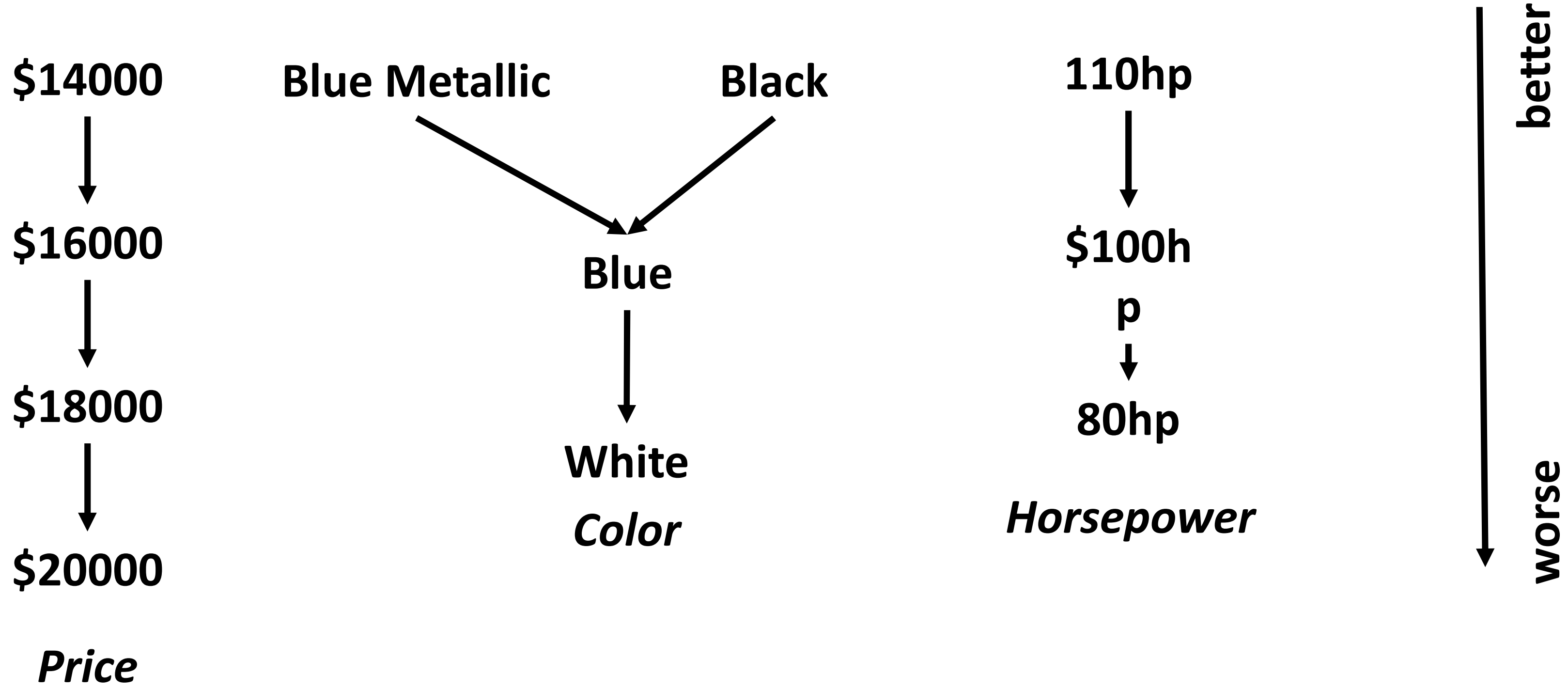
D_m



Skyline Queries

Pareto Optimality

Partial order of preferences



$R \subseteq (D_1 \times D_2 \times \cdots D_n)$

$v \in A_i \mid D_i$ total order.

$a \in D_i \mid b \in D_i$

$a \mid_i b$

$a \succ_i b$

↳ dominates b

red $\sim_i$ orange

$(a \sim_i b)$

$a \succsim_i b$.

$a \succsim_i$ $(a \quad b)$

$$S = \{o_1, o_2 \ldots o_p\}$$

s.t. $o_i > o_j$

$o_j > o_i$

$\forall\, o_i, o_j \in S$

$$\boxed{o_1 > o_2}$$

if $\exists\, i \in \{1, 2 \ldots n\}$

st $o_{1i} > o_{2i}$

$\forall\, i \in \{1, 2 \ldots n\}$

$$o_{1i} \gtrsim_i o_{2i}$$

$$S_{\text{skyln}} = \{o_i \in R \mid \neg o_j : o_j > o_i\}$$

Pareto Semantics

- $R \subseteq D_1 \times \ldots \times D_n$ over *n* attributes.
- A preference P_i on an attribute A_i with domain D_i is a strict partial order over D_i.

  If some attribute value $a \in D_i$ is preferred to some other value $b \in D_i$, then $(a,b) \in P_i$. This is often written as $a >_i b$ (read "a dominates b wrt P_i)
- An equivalence Qi on some attribute is an equivalence relation on Di compatible with Pi.

  If two attribute values $a, b \in D_i$ are equivalent, $(a,b) \in Q_i$, we write $a \approx_i b$
- If an attribute value $a \in D_i$, is either preferred over, or equivalent to another value $b \in D_i$, we write $a >\approx_i b$

Skyline Set

- Dominance Relationships

$$o_1 > o_2 \Leftrightarrow \exists\, k \in \{1, \ldots, n\}: o_{1,k} >_i o_{2,k} \wedge k \in \{1, \ldots, n\}: o_{1,k} >\approx_i o_{2,k}$$

where $o_{j,k}$ denotes the k-th component of the database tuple o_j

- Sykline Set

$$Sky := \{o_i \in R \mid \neg\, o_j : o_j > o_i\}$$

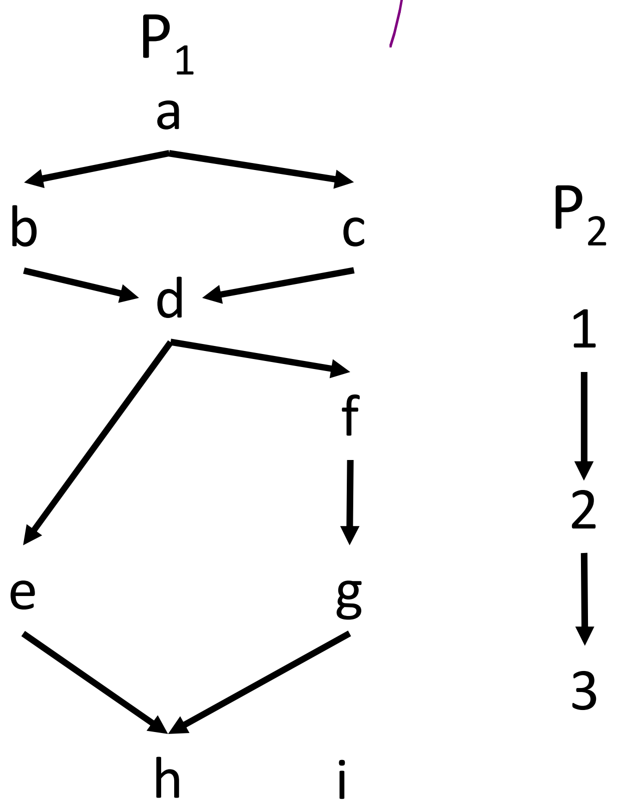
- Full Product order

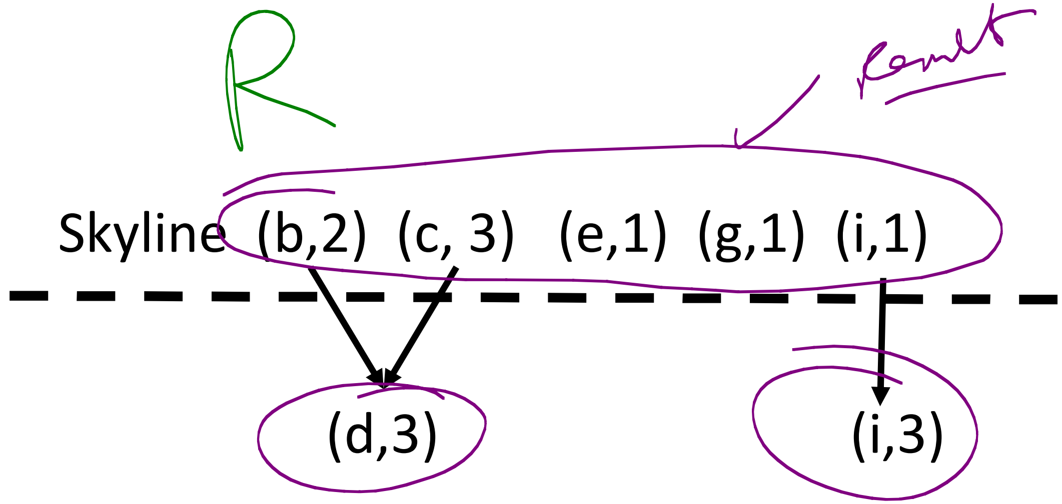
$$P \subseteq (D_1 \times D_2 \times \ldots \times D_n) \times (D_1 \times D_2 \times \ldots \times D_n)$$

Where for any $(o_i,o_j) \in P$, $o_i > o_j$ holds

	BP	Sugar	$\geq$ min
p_1	0	0	
p_2			
$\vdots$			
p_n			

Example





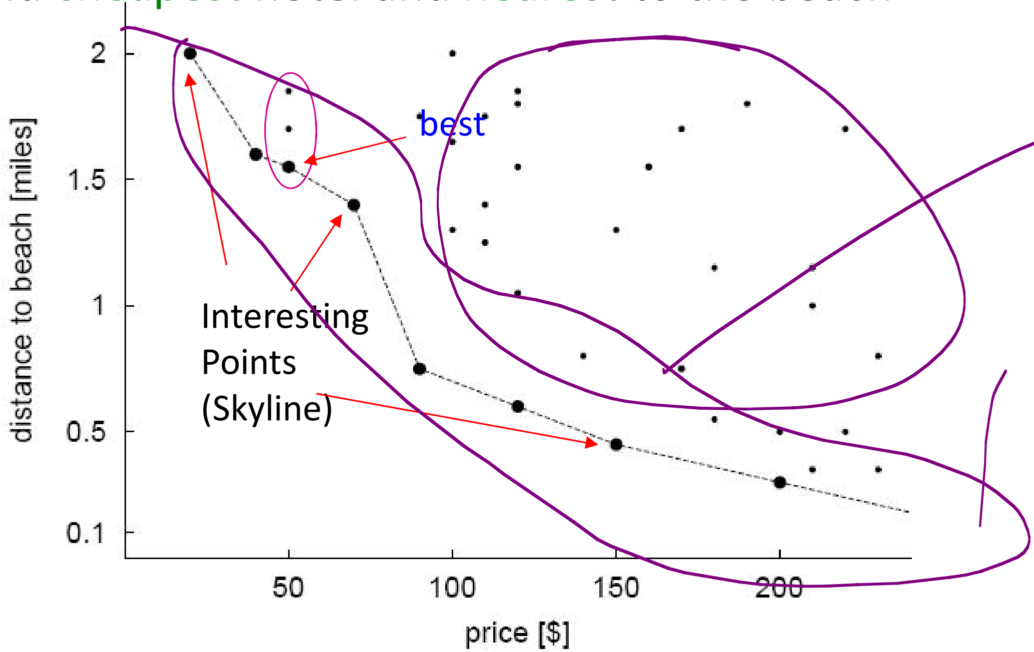
Skyline:

(b,2), (c,3)

(b,2) > (d,3)

Skyline

Find cheapest hotel and nearest to the beach



- Minimize price (x-axis)
- Minimize distance to beach (y-axis)
- Points not dominated by other points
- Skyline contains everyone favorite hotel regardless of preferences

Skyline Exercise

• S = Service, F= food, and D=décor. Each scored from 1-30, with 30 as the best.

• QUESTION: What restaurants are in the Skyline if we want best for service, food, decor and be the lowest priced ?

restaurant	S	F	D	price
Summer Moon	21	25	19	47.50
Zakopane	24	20	21	56.00
Brearton Grill	15	18	20	62.00
Yamanote	22	22	17	51.50
Fenton & Pickle	16	14	10	17.50
Briar Patch BBQ	14	13	3	22.50

Example 2: List of restaurant in FoodGuide

• ANSWER: No restaurant better than all others on every criterion individually

• While no one best restaurant, we want to eliminate restaurants which are worse on all criteria than some other

Result

restaurant	S	F	D	price
Summer Moon	21	25	19	47.50
Zakopane	24	20	21	56.00
Yamanote	22	22	17	51.50
Fenton & Pickle	16	14	10	17.50

Fig. 2. Restaurants in the skyline.

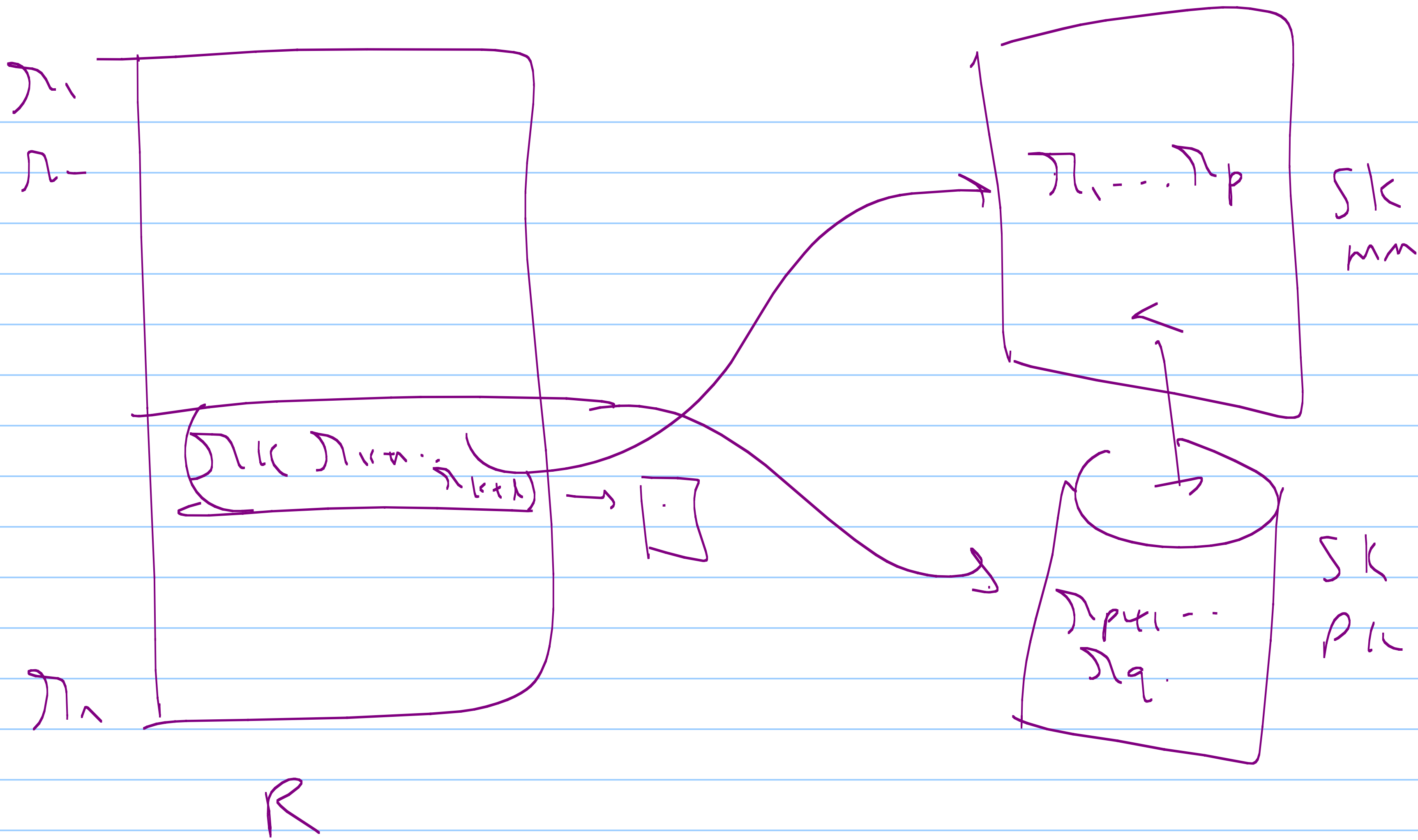
- Skyline Query

select * from FoodGuide
skyline of S max, F max, D max, price min

A_1 – · A_n
max min
max min

- Can we write an SQL query without using Skyline operator?

Answer: Yes, but cumbersome, expensive to evaluate, huge result set



r_1
r_2
r_n
R
r_k r_{k+1} .. r_{k+l}
$r_1 \ldots r_p$
SK
mm
$r_{p+1} \ldots$
r_q
SK
PK

100 million

Index

PhD → (MT, MS)

> B > 12

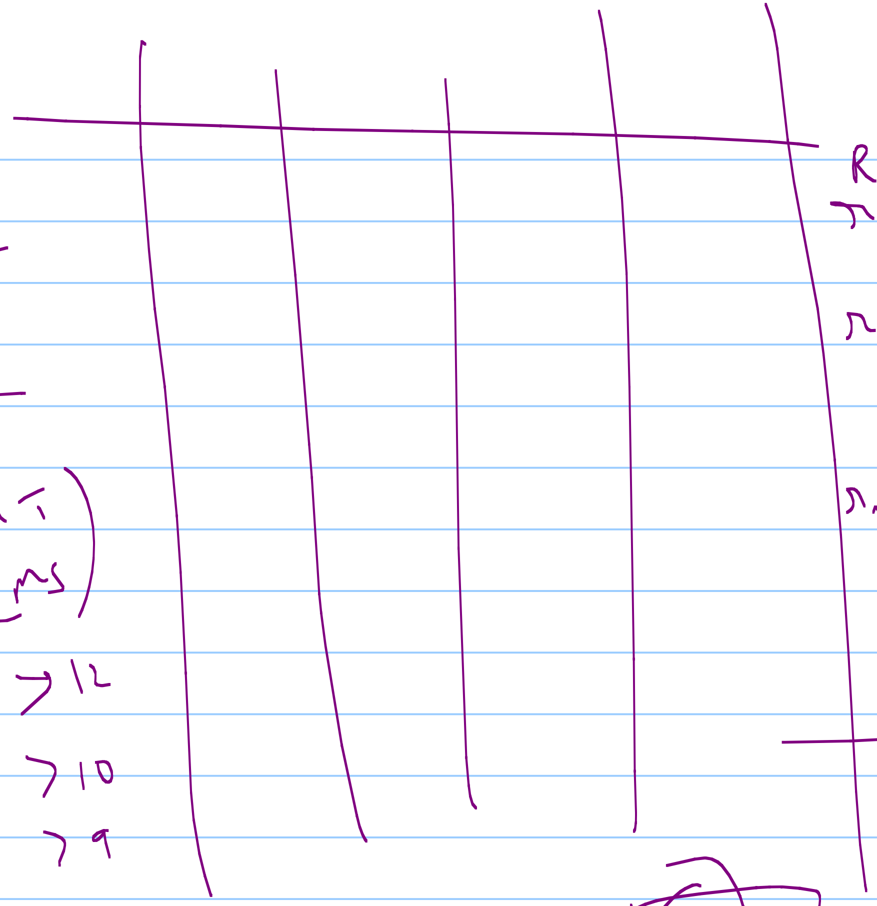
> 10

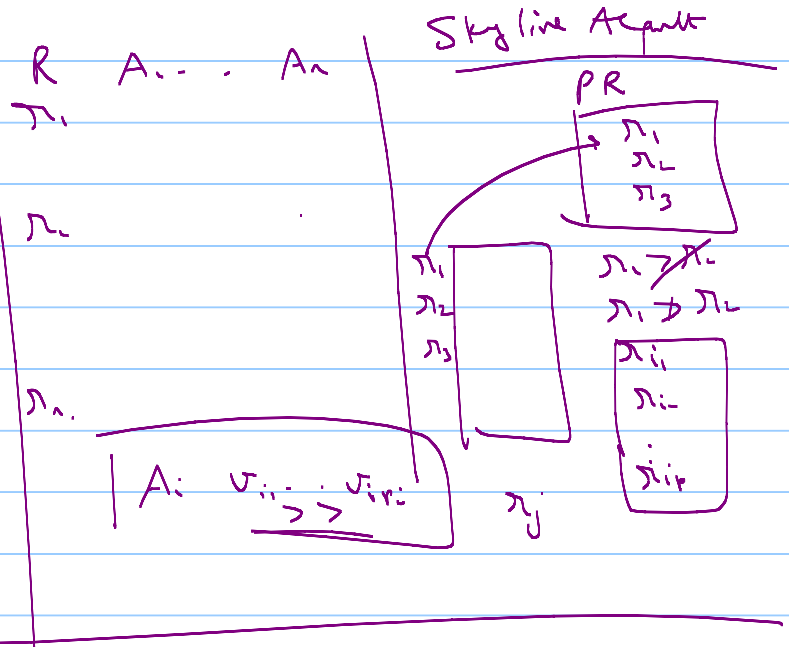
> 9

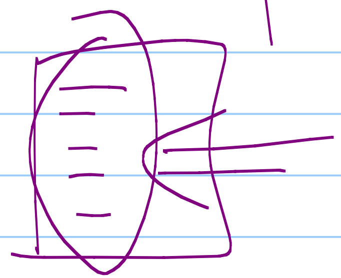
..

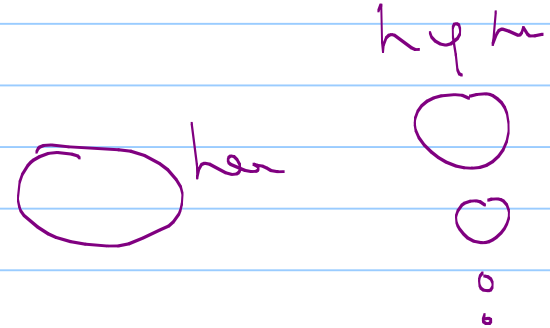
V

LK.









Cost Miles

Remain Sorted Skyline pt

	Cost	Miles
h3	50	3.0
h1	51	5.0
h2	52	4.0
h4	53	2.0

h1

h2

h3

Stan

$n > i$

New tuple

$>$

later

$t_j \in$ skyl

t_i t_i t_i

Query without Skyline Clause

- The following standard SQL query is equivalent to previous example but without using the Skyline operator

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

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

Using Skyline

2 and 3 Dimensional Skyline

- Two dimensional Skyline computed by sorting data

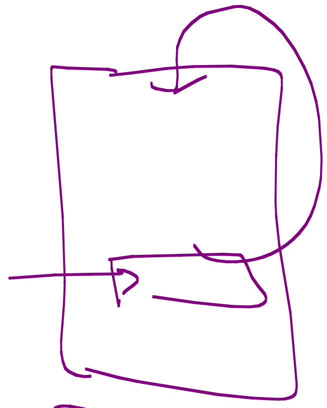
$$
\begin{array}{lll}
\langle h_1, & \$50, & 3.0\,\text{miles}\rangle \\
\langle h_2, & \boxed{\$51, \quad 5.0\,\text{miles}} & \rangle \\
\langle h_3, & \boxed{\$52, \quad 4.0\,\text{miles}} & \rangle \\
\langle h_4, & \$53, & 2.0\,\text{miles}\rangle
\end{array}
$$

Skyline

- For more than 2 dimension, sorting does not work

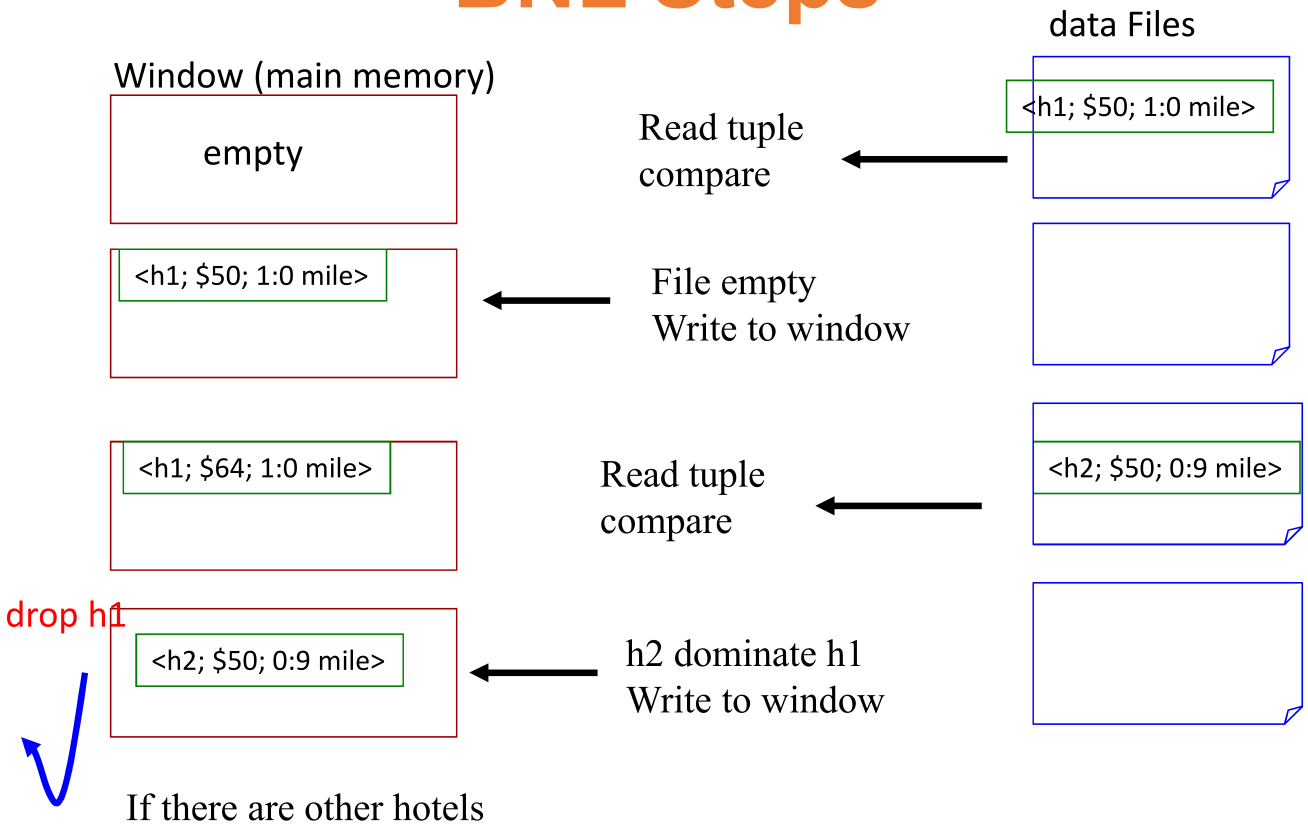
$$
\begin{array}{lllll}
\langle h_1, & \$50, & 3.0\,\text{miles}, & ***\rangle \\
\langle h_2, & \$51, & 5.0\,\text{miles}, & ****\rangle \\
\langle h_3, & \$52, & 4.0\,\text{miles}, & ***\rangle \\
\langle h_4, & \$53, & 2.0\,\text{miles}, & ***\rangle
\end{array}
$$

BNL Algorithm



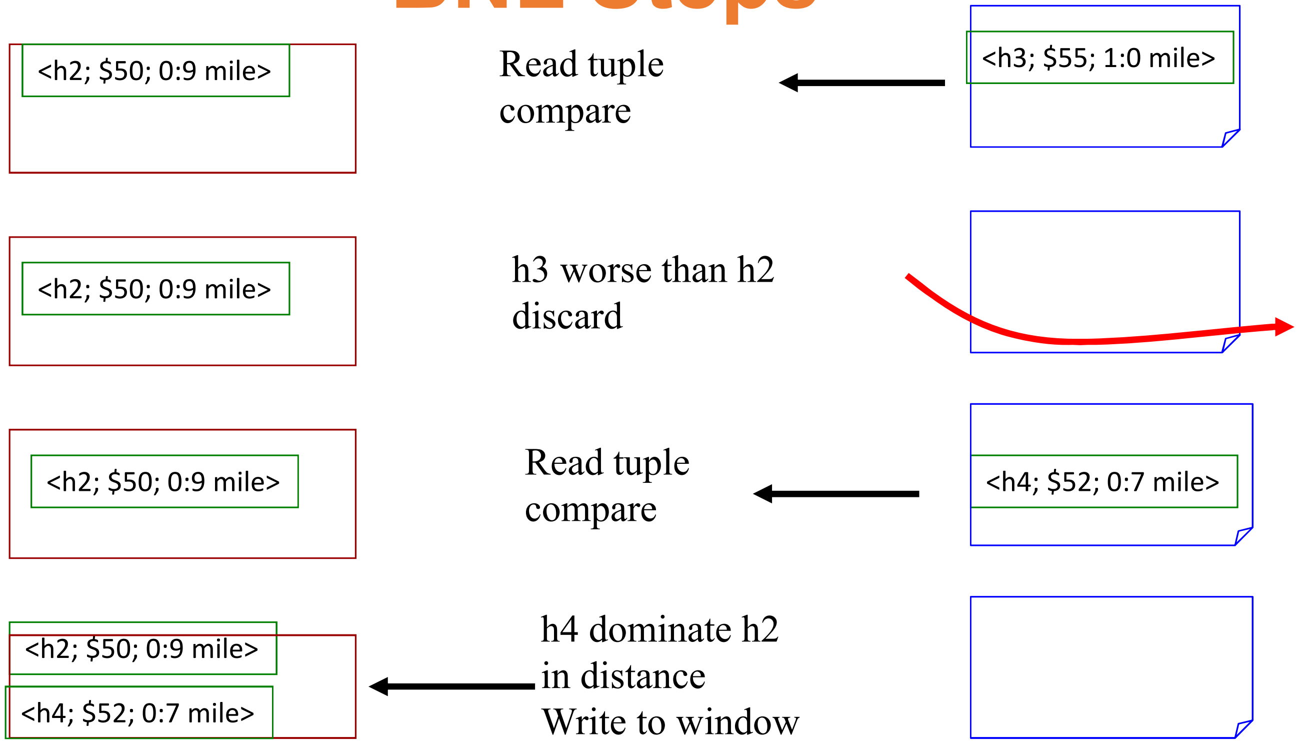
- Block Nested Loop
  - Compare each tuple with one another
  - Window in main memory contain best tuple
  - Write to temp file (if window has no space)
  - Authors Improvement – self organizing list

BNL Steps

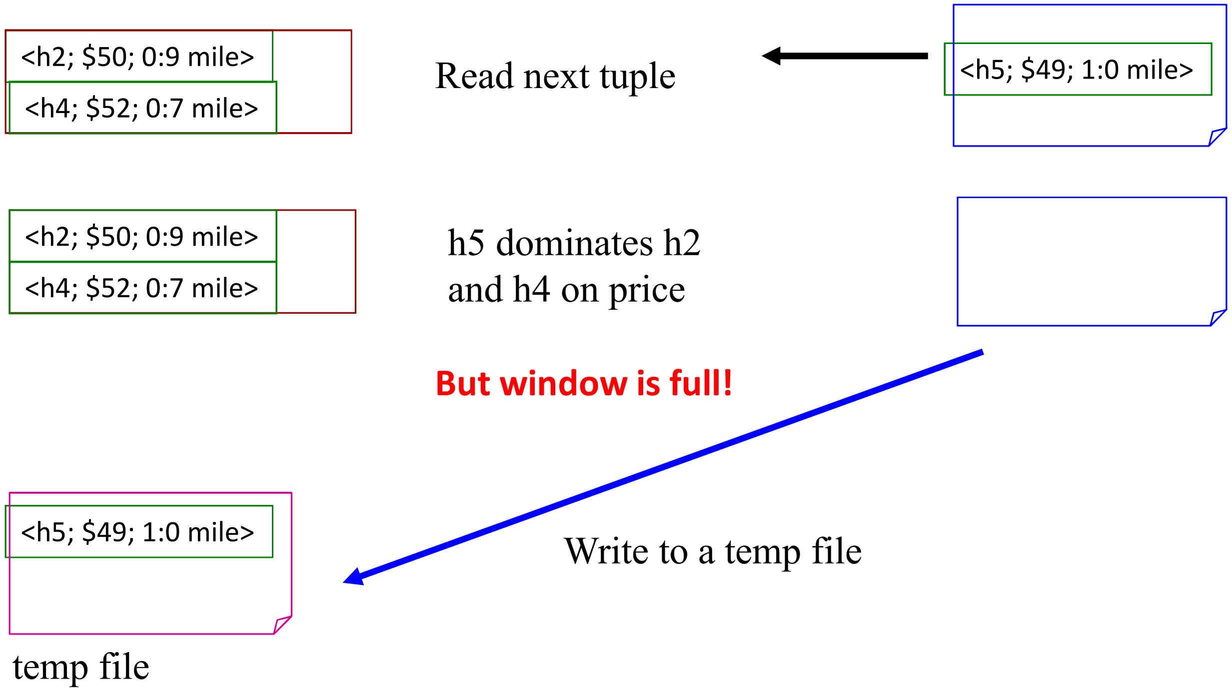


If there are other hotels worse than h2 → drop

BNL Steps

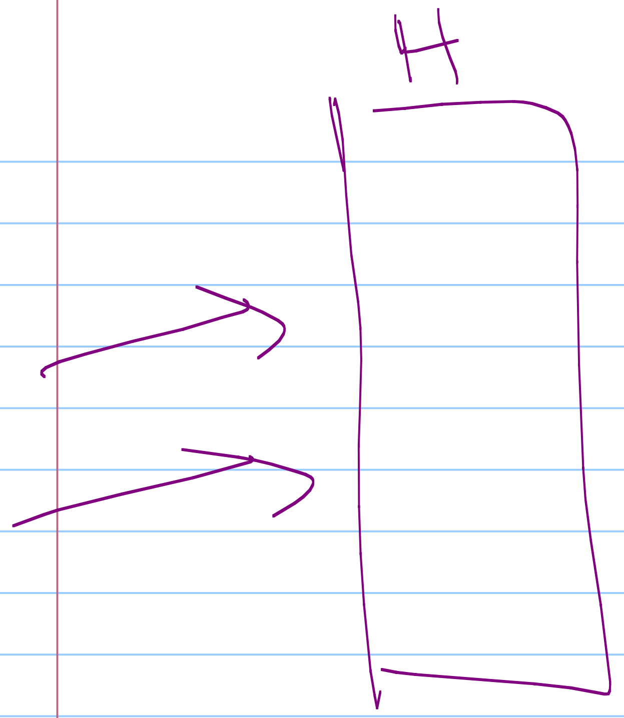

<h2; $50; 0:9 mile>
Read tuple
compare
<h3; $55; 1:0 mile>
<h2; $50; 0:9 mile>
h3 worse than h2
discard
<h2; $50; 0:9 mile>
Read tuple
compare
<h4; $52; 0:7 mile>
<h2; $50; 0:9 mile>
<h4; $52; 0:7 mile>
h4 dominate h2
in distance
Write to window

BNL Steps

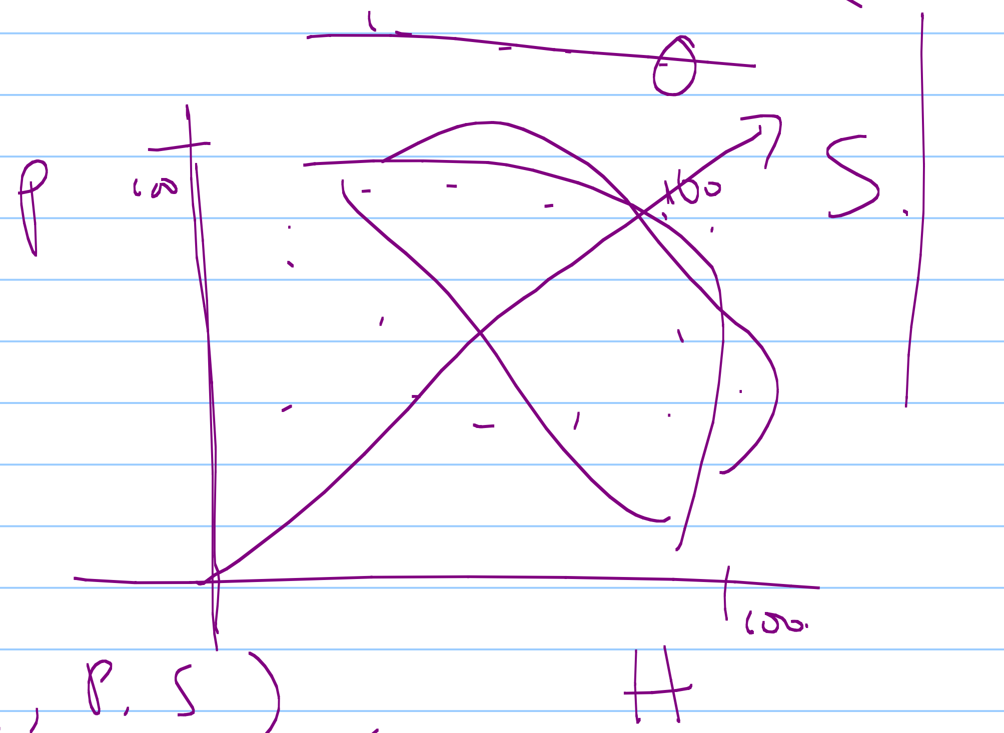


<h2; $50; 0:9 mile>
<h4; $52; 0:7 mile>
Read next tuple
<h5; $49; 1:0 mile>
<h2; $50; 0:9 mile>
<h4; $52; 0:7 mile>
h5 dominates h2
and h4 on price
But window is full!
<h5; $49; 1:0 mile>
Write to a temp file
temp file

$\boxed{0 - 100}$



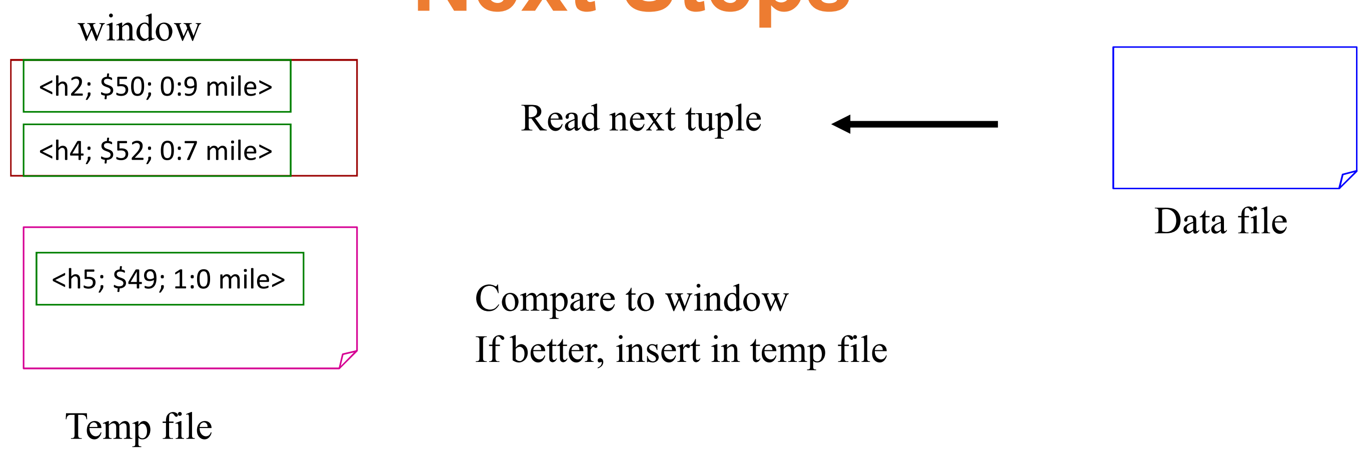
(E)



H, H, P, (H, P, S)

(H, P, S, E)

Next Steps



- End of Iteration – compare tuples in window with tuples in file
- If tuples is not dominated then part of skyline
- BNL works particularly well if the Skyline is small

Variants of BNL

- Speed-up by having window as self-organizing list
- Every point found dominating is moved to the beginning of window
- Example Hotel h5 under consideration eliminates hotel h3 from window. Move h5 to the beginning of window.
- Since h5 will be compared first by next tuple, it can reduces number of comparisons if h5 has the best value

Variants of BNL

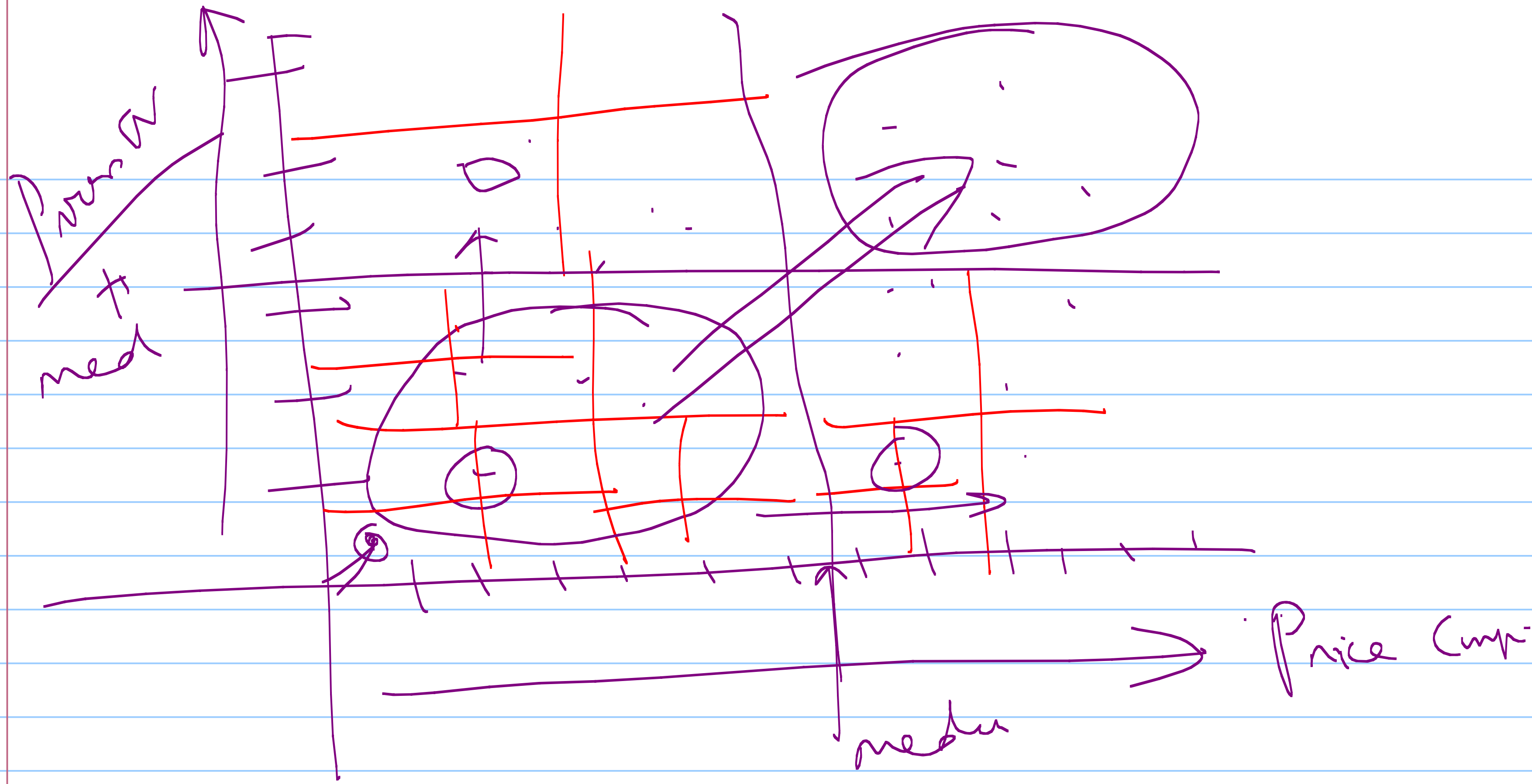
- Replace tuples in window: Keep dominant set of tuple.

<h1; $50; 1:0 mile>
<h2; $59; 0:9 mile>

h3 incomparable
Write to temp file

<h3; $60; 0:1 mile>

- h3 and h1 can eliminate more than (h1 and h2)
- Switch h3 to window and h2 to temp file



min./max

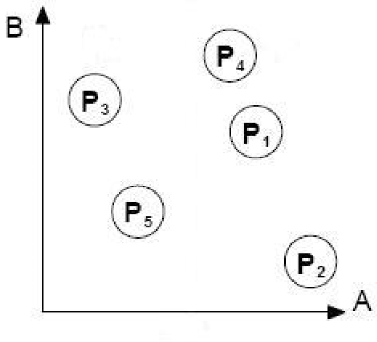
?

D & C Algorithms

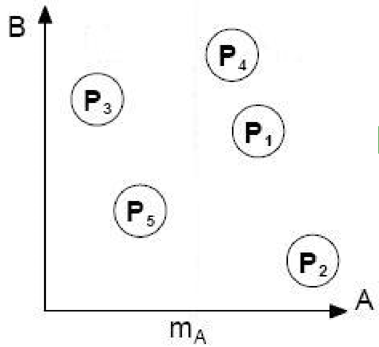
- Divide and Conquer
  - Get median value
  - Divide tuples into 2 partition
  - Compute skyline of each partition
  - Merge partition
  - Authors Improvement: M-Way & Early Skyline

D & C Algorithm

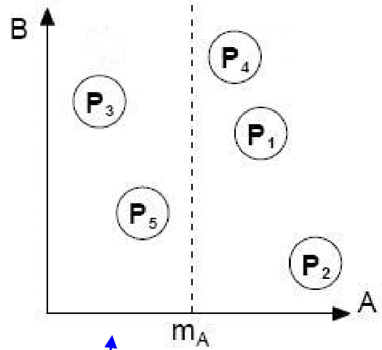
1. Original data



2. Get Median (mA) for all points

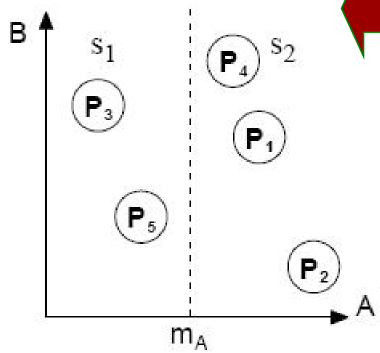


3. Divides dataset into 2 parts



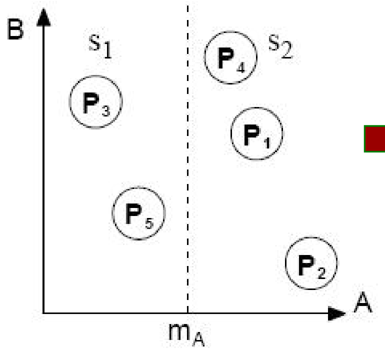
Values less than median

4. Compute Skyline S1 and S2

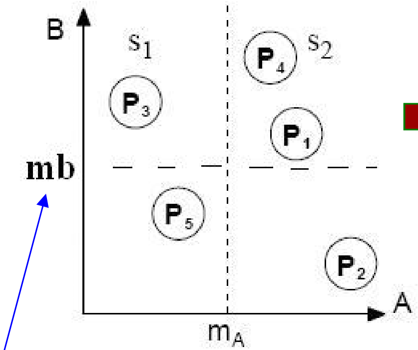


Next Steps

5. Eliminates points in S2 dominated by S1

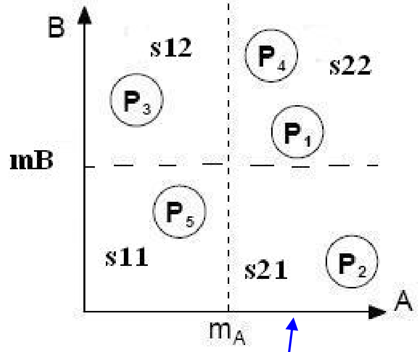


6. Get Median (mB) for S1



mb is median for dimension B

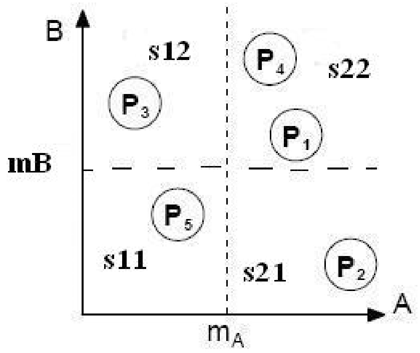
7. S1 and S2 divided into S11, S12, S21, S22



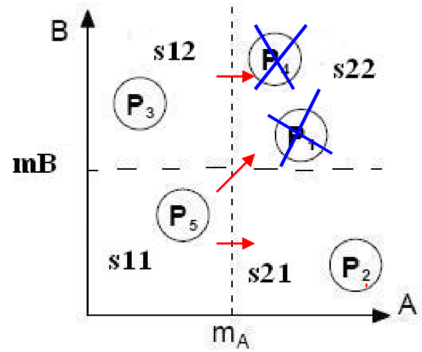
S21 smaller value in dimension B

Next Steps

7. S21 not dominated



8. Further partition and merge



Merge
S11 and S21
S11 and S22
S12 and S22

Do not merge
S12 and S21

S1x better than S2x in dimension A
Sx1 better than Sx2 in dimension B

The final skyline of AB is {P3; P5; P2}

Extension to D&C (M-Way)

1. If all data does not fit memory: terrible performance
2. Improve by dividing M-Partition that fits memory
3. Not take median but quantiles (smaller value)

$m \times n$

4. Merge pair-wise (m-merge)
5. Sub-partition is merged (refer to figure) and occupy memory

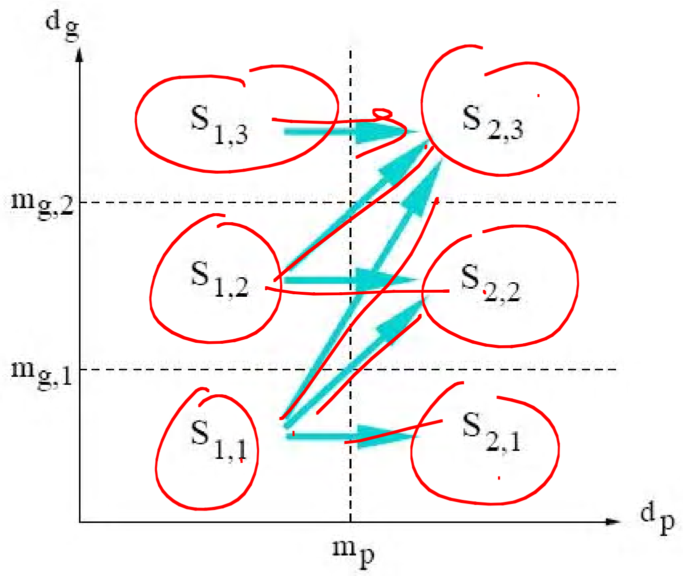


Figure 7: 3-way Merge

Extension (Early Skyline)

- Available main memory is limited

- Algorithm as follows
  - Load a large block of tuples, as many tuples as fit into the available main memory buffers
  - Apply the basic divide-and-conquer algorithm to this block of tuples in order to immediately eliminate tuples which are dominated by others
  - Step 2 is 'Early Skyline' (same as sorting in previous slide)
  - Partition the remaining tuples into m partitions

- Early Skyline incurs additional CPU cost, but it also saves I/O because less tuples need to be written and reread in the partitioning steps

- Good approach if result of Skyline small

	A_1	A_2		A_n
m_1				
m_2				
m_{100}				

$$(m_{i_1}, m_{i_2}, m_{i_3}) \Uparrow$$

Inputs to all variable phrases

Skyline BNL | DC |

Skylines

Subspace

$\nexists \quad \nexists \quad \exists$

Algo

$$\frac{p \in \text{Skyline}(i,j)}{p \in \text{Skyline}(i,j,k)}$$

Experiments and Result

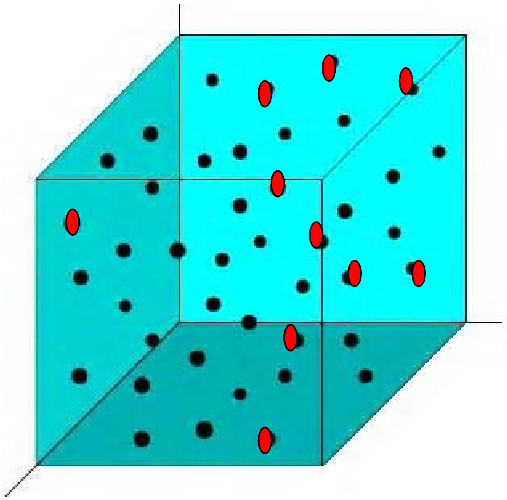
- The BNL algorithm outperforms other algo – window large
- Early Skyline very effective for the D&C algorithm
  - Small Partitions : algorithm completed quickly
- Other D&C variants (without Early Skyline) show very poor performance
  - Due to high I/O demands
- The BNL variants are good if the size of the Skyline is small
  - Number of dimensions increase D&C algorithm performs better
- Larger Memory: Performance of D&C algorithms improve but BNL worse
  - BNL algorithms are CPU bound

Introduction

- The maximal vector problem : Find vectors that is not dominated by any of the vectors from the set
- A vector dominates another if
  - Each of its components has an equal or higher value than the other vector's corresponding component
  - And it has a higher value on at least one of the corresponding components
- **Does this sound familiar??** Actually, this is the Skyline
- The maximal vector problem resurfaced with the introduction of skyline queries
- Instead of vectors or points, find the maximals over tuple

The Maximal Vector Problem

- Tuples = vectors (or points) in k-dimension space
- E.g., Hotel : Rating-stars, distance, price → <x, y, z>
- Input Set: n vectors, k dimensions



Output Set: m maximal vectors or SKYLINE

Algorithms Analysis

- Large data set: Do not fit main memory
- Compatible with a query optimizer
- At worse we want linear run-time
- Sorting is too inefficient
- How to limit the number of comparisons?
- Scan based or D&C algo?

Cost Model

- Simple approach: compare each point against every other point to determine whether it is dominated
  - This is O(n^2), for any fixed dimensionality k
  - Dominating point found: processing for that point can be curtailed
  - Average-case running time significantly better

- Best-case scenario, for each non-maximal point, we would find a dominating point for it immediately
  - Each non-maximal point would be eliminated in O(1) steps
  - Each maximal point expensive to verify since it need to be compared against each of the other maximal points to show it is not dominated
  - If there are not too many maximals, this will not be too expensive

Existing Generic Algorithms

Divide-and-Conquer Algorithms

- DD&C: double divide and conquer [Kung 1975 (JACM)]
- LD&C: linear divide and conquer[Bentley 1978 (JACM)]
- FLET: fast linear expected time [Bentley 1990 (SODA)]
- SD&C: single divide and conquer [Börzsönyi 2001 (ICDE)]

Scan-based (Relational “Skyline”) Algorithms

- BNL: *block nested loops* [Börzsönyi 2001 (ICDE)]
- SFS: *sort filter skyline* [Chomicki 2003 (ICDE)]
- **LESS**: *linear elimination sort for skyline* [Godfrey 2005 (VLDB)]

Performance of existing Algorithms

algorithm		ext.	best-case		average-case		worst-case	
DD&C	[14]	no	$\mathcal{O}(kn\lg n)$	§2.2	$\Omega(kn\lg n + (k-1)^{k-3}n)$	Thm.12	$\mathcal{O}(n\lg^{k-2} n)$	[14]
LD&C	[4]	no	$\mathcal{O}(kn)$	§2.2	$\mathcal{O}(n)$, $\Omega((k-1)^{k-2}n)$	[4], Thm. 11	$\mathcal{O}(n\lg^{k-1} n)$	[4]
FLET	[3]	no	$\mathcal{O}(kn)$	§2.2	$\mathcal{O}(kn)$	[3]	$\mathcal{O}(n\lg^{k-2} n)$	[3]
SD&C	[5]	–	$\mathcal{O}(kn)$	Thm. 2	$\Omega(\sqrt{k}2^{2k}n)$	Thm. 10	$\mathcal{O}(kn^2)$	Thm. 3
BNL	[5]	yes	$\mathcal{O}(kn)$	Thm. 4	–		$\mathcal{O}(kn^2)$	Thm. 5
SFS	[8]	yes	$\mathcal{O}(n\lg n + kn)$	Thm. 6	$\mathcal{O}(n\lg n + kn)$	Thm. 8	$\mathcal{O}(kn^2)$	Thm. 9
LESS	–	yes	$\mathcal{O}(kn)$	Thm. 14	$\mathcal{O}(kn)$	Thm. 13	$\mathcal{O}(kn^2)$	Thm. 15

Index based Algorithm

- So far we consider only generic algorithms

- Interest in index based algorithms for Skyline
  - Evaluate Skyline without need to scan entire datasets
  - Produce Skyline progressively, to return answer ASAP

- Bitmaps explored for Skyline evaluation
  - Number of value along dimensions small

- Limitation for index-based algorithm
  - Performance of index does not scale with the dimensions

D&C: Comparisons per Vector

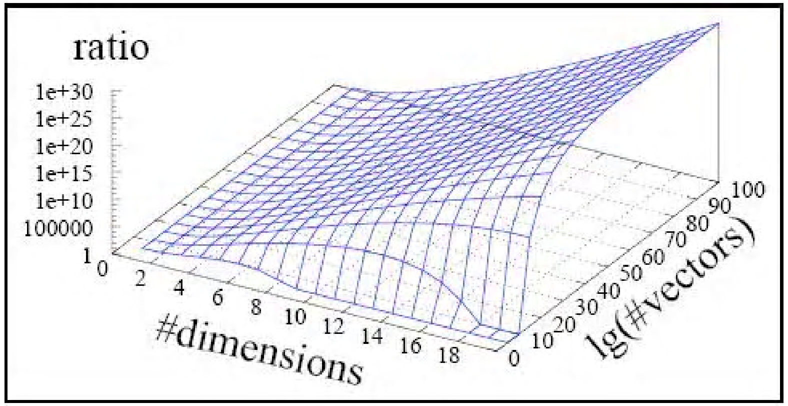
D&C algorithm's average-case in terms of n and k

$$
\begin{aligned}
T(n) &= 2T(n/2) \\
&\quad + \widehat{m}_{k,n} \lg_2^{k-2} \widehat{m}_{k,n} \\
&\quad \vdots \\
&\approx (k-1)^{k-2} n
\end{aligned}
$$

k	$(k-1)^{k-2}$
5	64
7	7,776
9	2,097,152

Claim in previous work: D&C more appropriate for large datasets with larger dimensions (k) say, for k > 7 than BNL

Analysis shows the opposite: D&C will perform increasingly worse for larger k and with larger n



Conclusion

- Divide and Conquer based algorithms are flawed. The dimensionality k results in very large "multiplicative constants" over their O(n) average-case performance

- The scan-based skyline algorithms, while naive, are much better behaved in practice

- Author introduced a new algorithm, LESS, which improves significantly over the existing skyline algorithms. It's average-case performance is O(kn).

- This is linear in the number of data points for fixed dimensionality k, and scales linearly as k is increased