# Skyline queries

## Information Systems M

Prof. Paolo Ciaccia

http://www-db.deis.unibo.it/courses/SI-M/

## Limits of scoring functions

- Although scoring functions are widely used to rank a set of objects, it is nowadays recognized that they have some major **problems**:
  - First, they have a limited expressive power, i.e., they can only capture those user preferences that "translates into numbers", which is not always the case (or, at least, doing so is not so natural!)

    "I prefer having white wine with fish and red wine with meat"
  - Second, deciding on the "best" scoring function to use and/or the specific weights can be hardly left to the final user, especially when there are several ranking attributes
- In this set of slides we will study an alternative to scoring functions, the so-called skyline queries, that have relevant practical applicability, and also represent a major step towards more general (i.e., powerful) preference models

# The concept of tuple domination

- A fundamental concept underlying the definition of skyline queries is that of

**Tuple domination**:

Given a relation R(A1,A2,…,Am,…), in which the Ai's are ranking attributes, assume without loss of generality that on each Ai lower values are better.

A tuple t dominates tuple t' with respect to **A** = {A1,A2,…,Am}, written $\mathbf{t} \succ_{\mathbf{A}} \mathbf{t'}$ or simply $\mathbf{t} \succ \mathbf{t'}$, iff:

$$\forall j = 1,\ldots,m: t.Aj \leq t'.Aj \wedge \exists j: t.Aj < t'.Aj$$

that is:

- t is no worse than t' on all the attributes, and
- strictly better than t' for at least one attribute

Notice that it can well be the case that neither $t \succ t'$ nor $t' \succ t$ hold

- The definition assumes that the "target point" is the origin **0** = (0,0,…,0), generalization to the case when the values of some attributes need to be maximized and to arbitrary target points is immediate



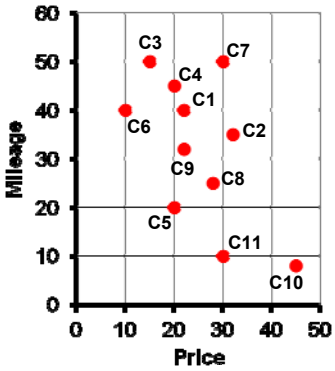
# Tuple domination: example (1)

- Both Points and Rebounds are to be maximized, thus:
  - Tracy McGrady dominates all players but Yao Ming and Shaquille O'Neal
  - Shaquille O'Neal dominates only Yao Ming and Steve Nash
  - Yao Ming dominates only Steve Nash
  - Steve Nash does not dominate anyone
  - …

| 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 | … |
| … | … | … | … |

# Tuple domination: example (2)

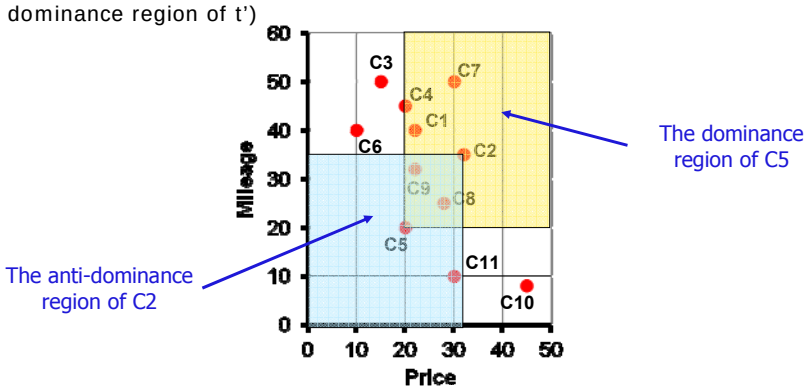
- Both attributes are to be minimized, thus:
  - Car C6 dominate C1 (same mileage, lower price), C3, C4, and C7
  - Car C5 dominates C1, C2, C4, C7, C8, and C9
  - Car C11 dominates …
  - …






# Dominance region

- The dominance region of a tuple t is the set of points in Dom(**A**) that are dominated by t
- Similarly, the anti-dominance (or "sudditance") region of t is the set of points in Dom(**A**) that dominate t
  - Clearly, t ≻ t' iff t' lies in the dominance region of t (and t in the anti-dominance region of t')



# Skyline queries

**Skyline of a relation** [BKS01]:

Given a relation R(A1,A2,…,Am,…), in which the Ai's are ranking attributes, the skyline of R with respect to **A** = {A1,A2,…,Am}, denoted $Sky_A(R)$ or simply Sky(R), is the set of undominated tuples in R:

$$Sky(R) = \{t \mid t \in R, \nexists\, t' \in R: t' \succ t\}$$

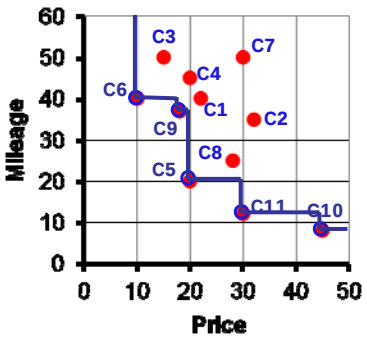
```
SELECT      *   -- Skyline query in PreferenceSQL [KK02]
FROM        R
PREFERRING LOW(A1) AND LOW(A2) AND ... AND LOW(Am)
```

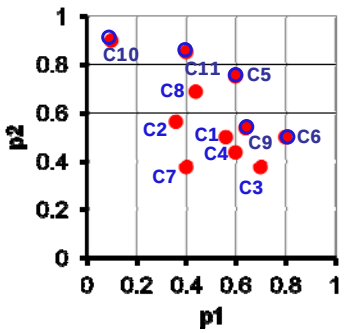
- Equivalently, $t \in Sky(R)$ iff no point in R lies in the anti-dominance region of t
- In computational geometry, skyline queries are also known as the "maximal vectors problem"; for multiple criteria optimization problems, their result is a set of so-called Pareto optimal solutions



# A skyline example

- In the attribute space…
  - The "skyline profile" shows the union of the dominance regions of skyline points
- In the score space…
  - No matter how we define scores, the skyline doesn't change!
  - I.e., the skyline is insensitive to any "stretching" of coordinates



## What's so special about skyline queries?

- Given a relation R, let d be a monotone distance function such that the 1-NN with respect to the origin, denoted $t_{NN,d}(R)$, is univocally defined
  - Thus, for 1-NN queries nondeterminism is not an isssue here
- Let **d** be the (infinite) set of all such distance functions
- We have the following result relating skyline and 1-NN queries, when both have the same target point:

$$\mathbf{Sky(R) = \cup_{d \in d} \{t_{NN,d}(R)\}}$$

- This is to say that:
  1) If t is the 1-NN for a suitable distance function d, then t is part of the skyline
  2) Conversely, if t is a skyline point, then there exists a distance function d that is minimized by t
- For this reason, skyline points are also sometimes called "potential NN's"
- Clearly, the same result holds for monotone scoring functions with no top-1 ties



## Proof

1) If t is the 1-NN for a suitable distance function d, then t is part of the skyline

- By negating the conclusion.
  Assume t is **not** part of the skyline, i.e., there exists a tuple t' that dominates t.
  For any monotone distance function d it therefore holds d(t',**0**) ≤ d(t,**0**).
  Since, by hypothesis, 1-NN ties are excluded, t can never be the 1-NN

2) If t is a skyline point, then there exists a distance function that is minimized by t

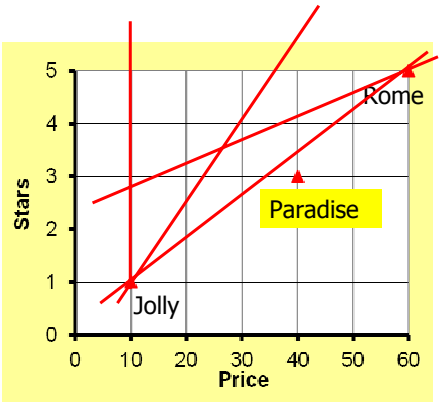
- The proof is constructive.
  Consider the weighted $\mathbf{L_{\infty,W}}$ distance with weights $w_i = 1/t.Ai$, i=1,…,m.
  It is $L_{\infty,W}(t,\mathbf{0}) = \max_i\{w_i*t.Ai\} = 1$.
  For any other point t' it is is $L_{\infty,W}(t',\mathbf{0}) = \max_i\{w_i*t'.Ai\} = \max_i\{t'.Ai/t.Ai\} > 1$, since t is a skyline point

# "Accessibility" of skyline points

**S = Ws * Stars – Wp * Price**

**Hotels**

| Name | Price | Stars |
|---|---|---|
| Jolly | 10 | 1 |
| Rome | 60 | 5 |
| Paradise | 40 | 3 |




- For no weights combination Paradise is the top-1 hotel
- Similar problems with:
  - Arbitrary values of k and/or
  - Almost all scoring functions



# Skylines do not admit any distance function

- The skyline of R does not correspond to any k-NN (or top-k) result, i.e:

  Given a schema R(A1,…,Am,…) there is no distance function d (equivalently, scoring function S) that, on all possible instances of R, yields in the first k positions the skyline points

  - Note that here we allow k to be variable, so as to match the actual number of skyline points on each instance of R

**Proof**: it is Sky(R') = {t1,t4}, thus it has to be: {S(t1), **S(t4)**} **> S(t2)**.

On the other hand, it is Sky(R") = {t2,t3}, thus: {**S(t2)**,S(t3)} **> S(t4)**, a contradiction

**R'**

| TID | p1 | p2 |
|---|---|---|
| t1 | 0.9 | 0.6 |
| t2 | 0.8 | 0.4 |
| t4 | 0.5 | 0.7 |

**R''**

| TID | p1 | p2 |
|---|---|---|
| t2 | 0.8 | 0.4 |
| t3 | 0.7 | 0.8 |
| t4 | 0.5 | 0.7 |

# Evaluation of skyline queries

- The issue of efficiently evaluating a skyline query has been largely investigated, and many algorithms introduced so far
- A basic reason is that the problem is "more difficult" than top-k queries, since it has a worst-case complexity of $\Theta(N^2)$ for a DB with N objects

- What we see are some algorithms that follow one of the two basic approaches:

Generic:

it computes the skyline without any auxiliary access method (indexes)

- Thus, the input relation can also be the output of some other operation (join, group by, etc.)

Index-based:

it is assumed that an index is available



# The naïve Nested-Loops (NL) algorithm

- The simplest (and very inefficient!) way to compute the skyline of R is to compare each tuple with all the others

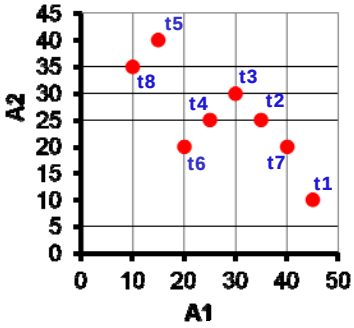
**ALGORITHM NL (nested-loops)**
**Input**: a dataset R, a set of attributes **A** inducing $\succ$
**Output**: Sky(R), the skyline of R with respect to **A**

```
Sky(R) := ∅;
for all tuples t in R:
    undominated := true;
    for all tuples t' in R:
            if t' ≻ t then: {undominated := false; break}
    if undominated then: Sky(R) := Sky(R) ∪ {t};
return Sky(R);
end.
```

# NL: an example

- The origin is the target (Low(A1) and Low(A2))

| R |
|---|
| t1 |
| t2 |
| t3 |
| t4 |
| t5 |
| t6 |
| t7 |
| t8 |




| Sky |
|---|
| t1 |
| t6 |
| t8 |

| TID | No. of comparisons |
|---|---|
| t1 | 7 |
| t2 | 3 |
| t3 | 3 |
| t4 | 5 |
| t5 | 7 |
| t6 | 7 |
| t7 | 6 |
| t8 | 7 |
| **Total** | **45** |

If t $\in$Sky(R), it will always be compared with all other tuples



# The Block-Nested-Loops (BNL) algorithm

- The BNL algorithm [BKS01] improves over NL by immediately discarding all tuples that are dominated by at least one other tuple
- Thus, it also avoids comparing twice the same pair of tuples (as NL does!)
- BNL allocates a buffer (*window*) W in main memory, whose size is a design parameter, and sequentially reads the data file
- Every new tuple t that is read from the data file is compared with only those tuples that are currently in W

The BNL algorithm has been proposed in [BKS01] for skyline queries, however its applicability is far more general!

Donald Kossmann

# The logic of the BNL algorithm

- When reading a new tuple t, three cases are possible:

1) If some tuple t' in W dominates t, then t is immediately discarded
2) If t dominates some tuple t' in W, all such tuples are removed from W and t is inserted into W
3) If none of the above two cases holds, then t is inserted into W. However, if no space in W is left, then t is written to a temporary file F

- When all tuples have been processed, if F is empty the algorithm stops, otherwise a new iteration is started by taking F as the new input stream

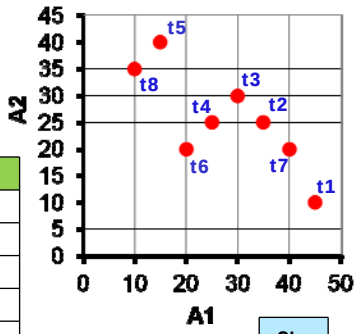
- The tuples that were inserted in W when F was empty can be immediately output, since they have been compared with all other tuples
- The others in W can be output **during** the next iteration; a tuple t can be output when a tuple t' is found in F that followed t in the sequential order
  - For this, a timestamp (counter) is attached to each tuple



# BNL: an example

- Assume |W| = 2 and the origin as the target

| R |
|---|
| t1 |
| t2 |
| t3 |
| t4 |
| t5 |
| t6 |
| t7 |
| t8 |




| Sky |
|---|
| t1 |
| t6 |
| t8 |

**1st**

| W |
|---|
| t1 |
| t6 |

| F |
|---|
| t3 |
| t5 |
| t8 |
| … |

**2nd iteration**

| W |
|---|
| t6 |
| t8 |

| TID | No. of comparisons |
|---|---|
| t1 | 7 |
| t2 | 2 |
| t3 | 1 |
| t4 | 2 |
| t5 | 2 |
| t6 | 2 |
| t7 | 0 |
| t8 | 0 |
| **Total** | **16** |

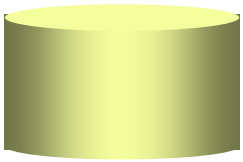
For each tuple t only comparisons with tuples following t in R are counted

# BNL: another example

| Restaurant | Price | Rating |
|---|---|---|
| FreshFish | 70 | 2 |
| OceanView | 30 | 3 |
| VealHere | 50 | 7 |
| Sunset | 40 | 6 |
| Country | 48 | 5 |
| SteakHouse | 60 | 3 |

W

| Restaurant | ... |
|---|---|
| | |
| | |



F

**Low(Price) and High(Rating)**



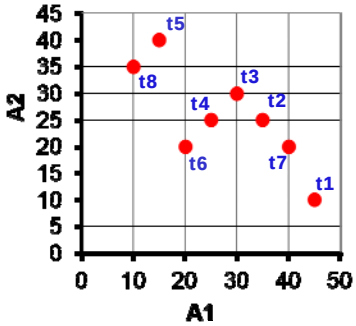
# BNL: some comments

- Experimental results in [BKS01] show that BNL is CPU-bound and that its performance deteriorates if W grows
  - Since with larger W BNL executes more comparisons
- On the other hand, BNL has a relatively low I/O cost
- Performance is also negatively affected by the number of skyline points
- The skyline cardinality depends on the number of attributes and on their correlation
  - Negatively (or anti-)correlated attributes, like Price and Mileage, lead to larger skylines
- [BKS01] also introduces some variants of BNL, among which BNL-sol, that manages W as a self-organizing list
  - The idea is to first compare incoming objects with those in W (called "*killer*" objects) that have been found to dominate several other objects

... and another algorithm (D&C) based on a "divide-and-conquer" approach

# BNL: setting |W| = 1

- Now |W| = 1, which yields the minimum number of comparisons for a given input order (equal to those of |W| =2 in this example)




| 1st | 2nd | 3rd | 3rd (end) |
|---|---|---|---|
| W | W | W | W |
| t1 | t6 | t6 | t8 |
| F | F | F | F |
| t2 | t3 | t5 | t5 |
| t3 | t5 | t8 | |
| t4 | t8 | | |
| t5 | | | |
| t6 | | | |
| t7 | | | |
| t8 | | | |

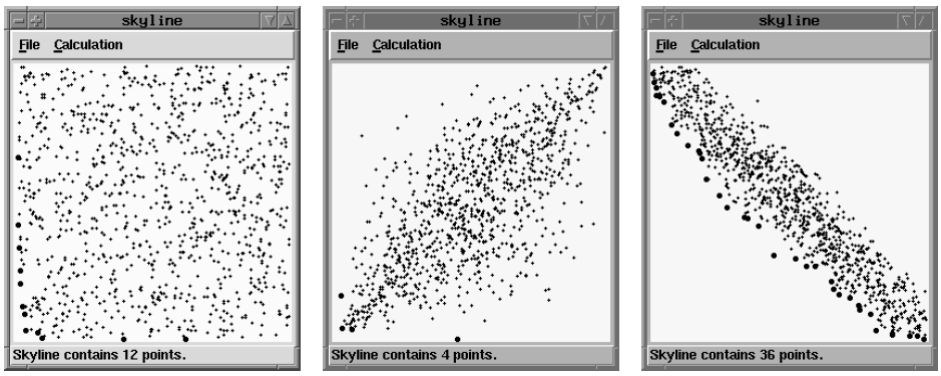
t6 can be output during the 3rd iteration, just after reading t8

| TID | No. of comp. |
|---|---|
| t1 | 7 |
| t2 | 2 |
| t3 | 1 |
| t4 | 2 |
| t5 | 2 |
| t6 | 2 |
| t7 | 0 |
| t8 | 0 |
| **Total** | **16** |



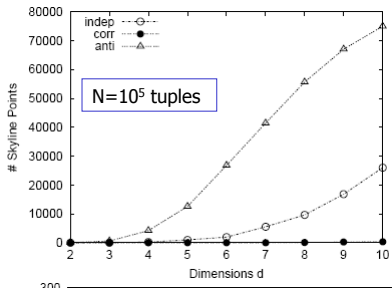
# BNL: datasets and experiments (1) [BKS01]

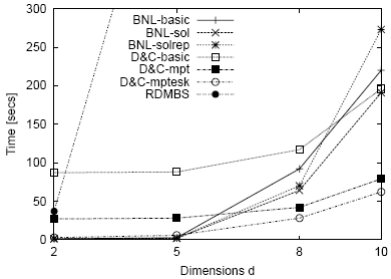
- Synthetic data (uniform independent, correlated and anti-correlated)




- In the figure: 1000 points (skyline points are in bold)

# BNL: datasets and experiments (2) [BKS01]




- RDBMS: the NL algorithm implemented as a correlated subquery:
  "t is part of the skyline if NOT EXISTS(…)"

**In this figure:**
Independent datasets

dimensionality $\in$ [2,10]
window = 1Mbyte

cardinality N=$10^5$ tuples

Sun Ultra, 333MHz CPU
128Mbytes RAM



# SFS: Sort-Filter-Skyline [CGG+03]

- SFS aims to reduce the overall number of comparisons
- To this end, it first performs a topological sort of the input data, which respects the skyline preference criteria

**Topological sort**:
Given $\succ$, a topological sort of R is a complete (no ties) ordering < of the tuples in R such that:

$$t \succ t' \Rightarrow t < t'$$

i.e., if t dominates t', then t precedes t' in the complete ordering

- Here the key observation is:

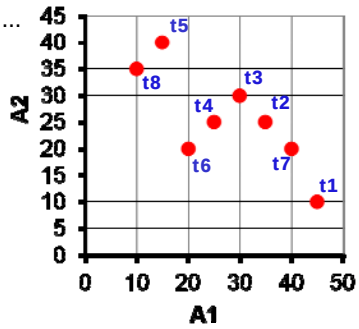
If the input is topologically sorted,
then a new read tuple cannot dominate
any previously read tuple! ($t > t' \Rightarrow t \not\succ t'$)

# Topological sort: example

- For the data in the figure, possible results of a topological sort are:

| | | | | sum | | product |
|---|---|---|---|---|---|---|
| t6 | t8 | t1 | t6 | 40 | t8 | 350 |
| t4 | t5 | t6 | t8 | 45 | t6 | 400 |
| t2 | t6 | t4 | t4 | 50 | t1 | 450 |
| t3 | t1 | t7 | t5 | 55 | t5 | 600 |
| t1 | t4 | t8 | t1 | 55 | t4 | 625 |
| t8 | t7 | t3 | t2 | 60 | t7 | 800 |
| t7 | t3 | t2 | t3 | 60 | t2 | 875 |
| t5 | t2 | t5 | t7 | 60 | t3 | 900 |

…

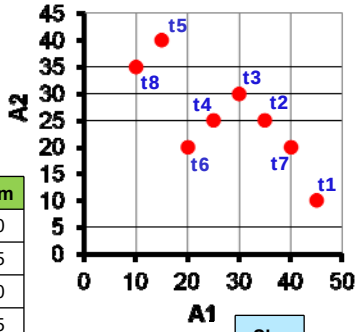



- In practice, a topological sort is obtained by ordering data using a monotone distance (scoring) function compatible with the skyline criteria



# SFS: an example

- Assume |W| = 2
  and the origin as the target




| | sum |
|---|---|
| t6 | 40 |
| t8 | 45 |
| t4 | 50 |
| t5 | 55 |
| t1 | 55 |
| t2 | 60 |
| t3 | 60 |
| t7 | 60 |

| Sky |
|---|
| t6 |
| t8 |
| t1 |

**1st**

| W |
|---|
| t6 |
| t8 |

| F |
|---|
| t1 |
| … |

**2nd iteration**

| W |
|---|
| t1 |
| |

| TID | No. of comp. |
|---|---|
| t6 | 7 |
| t8 | 3 |
| t4 | 0 |
| t5 | 0 |
| t1 | 0 |
| t2 | 0 |
| t3 | 0 |
| t7 | 0 |
| **Total** | **10** |

For each tuple t only comparisons with tuples following t in the sorted input are counted

# SFS: further properties

- At the end of each iteration all the tuples in W can be output
  - since no tuple in W can be discarded by a subsequent tuple
- The number of iterations is therefore the minimum one: $\lceil |Sky(R)|/|W| \rceil$
  - In contrast, BNL has no such guarantee
- SFS can return a tuple as soon as it is inserted in the window
  - Therefore, in W one can just store the skyline attribute values, which leads to save (much) space
- Two non-skyline tuples will never be compared
  - Since in W only skyline tuples are present
- Managing the window data structure is now much easier
  - Since only insertions are to be supported
  - No deletion of specific tuples, thus no need to manage empty slots



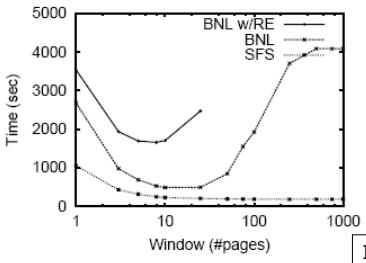
# Experimental results (from [CGG+03])

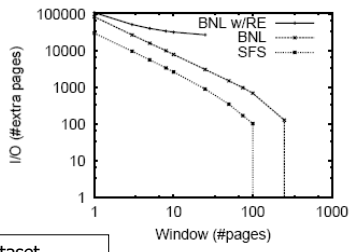
- Data sorted using the "entropy" distance function:

$$d(t,\mathbf{0}) = -\sum_{i=1,m} \ln(2 - t.Ai)$$

$$= -\ln(\exp(\sum_{i=1,m} \ln(2 - t.Ai))) = -\ln(\prod_{i=1,m}(2-t.Ai))$$

which yields the same ordering as $2^m - \prod_{i=1,m}(2-t.Ai)$ ( $\in [0,2^m - 1]$)



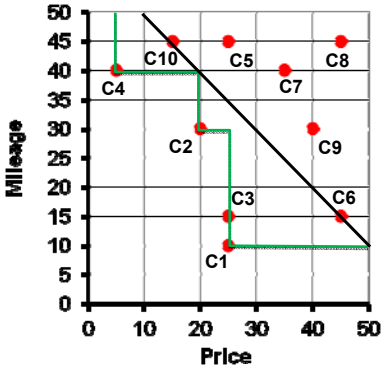




BNL w/RE: input sorted using the "reverse" entropy

Independent dataset
cardinality $N=10^6$ tuples
dimensionality = 7
window = # 4Kbyte pages
AMD Athlon, 900MHz CPU
384Mbytes RAM

# SaLSa [BCP06] [BCP08]

- SaLSa (Sort and Limit Skyline algorithm) extends the ideas of SFS by observing that, when data are topologically sorted, it is possible to avoid reading all the input tuples




Data sorted using sum: t.Price + t.Mileage

After reading C6 (or C10), whose sum is 60, we know that no further skyline point exists

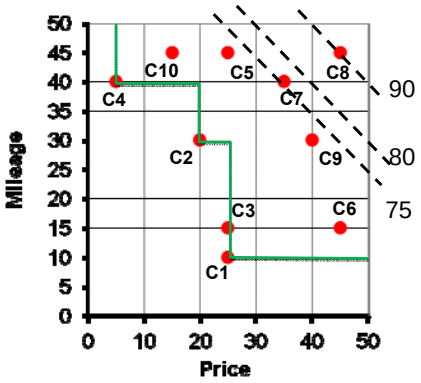
... however using **all** the current points in Sky(R) to this purpose is costly: The problem is NP-hard [BCP08]

And?



# The "stop-point"

- SaLSa makes use of a **single** skyline tuple, the so-called stop-point , $t_{stop}$, to determine when execution can be halted
- In this case it is sufficient to check that what is still to be read lies in the dominance region of $t_{stop}$




| $t_{stop}$ | halt when sum ≥ |
|---|---|
| C1 | 75 |
| C2 | 80 |
| C4 | 90 |

# Choosing the stop-point

- For symmetric distance (scoring) functions, and assuming that on all coordinates the ranges are the same ([0,1], [0,50], etc.) it is possible to prove that the optimal choice for the stop-point is given by the rule:

$$t_{stop} = \text{argmin}_{t \in SKY(R)} \{\max_i\{t.Ai\}\}$$

  that is, the tuple for which the maximum coordinate value is minimum

- Note that this holds for **any** symmetric distance function

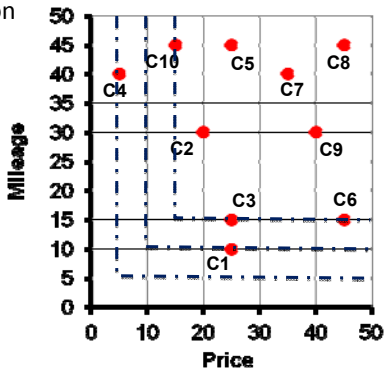
| $t_{stop}$ | Price | Mileage | halt when sum ≥ |
|---|---|---|---|
| C1 | 25 | 10 | 75 |
| C2 | 20 | 30 | 80 |
| C4 | 5 | 40 | 90 |



# Optimally ordering the points

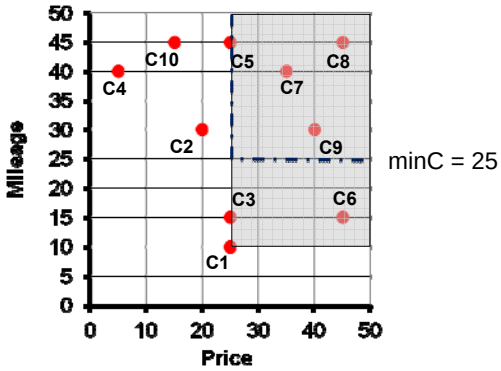
- Among the many alternatives to sort the input data, SaLSa uses a provably optimal criterion, i.e., on each instance, ordering data using another (symmetric) function cannot discard more points
- The optimal criterion is called **minC** (minimum coordinate), that is, for each tuple t the value of $\min_i\{t.Ai\}$ is used
- In case of ties, the secondary criterion "sum" is used

| | minC | sum |
|---|---|---|
| C4 | 5 | |
| C1 | 10 | |
| C3 | 15 | 40 |
| C6 | 15 | 60 |
| C10 | 15 | 60 |
| C2 | 20 | |
| C5 | 25 | |
| C9 | 30 | |
| C7 | 35 | |
| C8 | 45 | |



# Stopping with minC

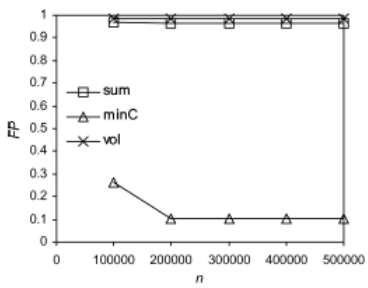
- The stop-point is C1, for which it is **$\max_i\{C1.Ai\} = 25$**
- Thus, as soon as it is minC ≥ 25 SaLSa can be halted
- The general stop condition is therefore: **$minC \geq \max_i\{t_{stop}.Ai\}$**



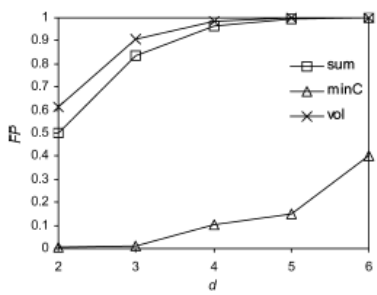



# Experimental results (from [BCP08]) (1)

- FP = fraction of fetched points, independent datasets (vol = SFS)



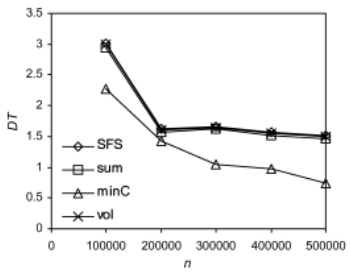

cardinality N ∈ [$10^5$,5*$10^5$] tuples
dimensionality = 4



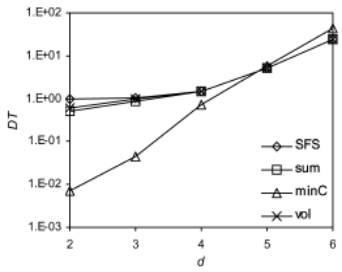

cardinality N=5*$10^5$ tuples
dimensionality ∈ [2,6]

# Experimental results (from [BCP08]) (2)

- DT = no. of comparisons (dominance tests), normalized to the cardinality of the dataset




cardinality N ∈ [$10^5$,5*$10^5$] tuples
dimensionality = 4




cardinality N=5*$10^6$ tuples
dimensionality ∈ [2,6]



# Experimental results (from [BCP08]) (3)

- Mixed dataset = half points are anti-correlated, others are dominated

Table II. Elapsed Time on Synthetic Datasets ($n = 500K$, $d = 4$. Times are in Seconds)

| | Uniform | | | Mixed | | |
|---|---|---|---|---|---|---|
| | sum | minC | vol | sum | minC | vol |
| Sorting | 1.57 | 1.79 | 1.64 | 2.15 | 2.40 | 2.44 |
| Fetching | 3.85 | 0.42 | 3.94 | 3.82 | 1.91 | 3.90 |
| Filtering | 2.11 | 0.10 | 2.21 | 5.02 | 2.57 | 6.26 |
| Total | 7.53 | 2.31 | 7.79 | 10.99 | 6.88 | 12.60 |

cardinality N = $10^5$ tuples
dimensionality = 4
Data stored and sorted in IBM DB2
Pentium IV, 3.4GHz CPU
512Mbytes RAM

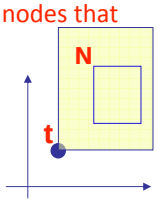
Table III. Elapsed Time on Real Datasets ($d = 4$)

| | NBA | | | Color | | | Household | | | EEG | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | sum | minC | vol | sum | minC | vol | sum | minC | vol | sum | minC | vol |
| Sorting | 0.02 | 0.03 | 0.02 | 0.07 | 0.11 | 0.07 | 0.34 | 0.40 | 0.36 | 39.97 | 48.65 | 40.33 |
| Fetching | 0.08 | 0.03 | 0.09 | 0.54 | 0.08 | 0.55 | 1.02 | 0.01 | 1.02 | 22.59 | 0.01 | 22.59 |
| Filtering | 0.03 | 0.01 | 0.04 | 0.35 | 0.06 | 0.34 | 0.52 | 0.01 | 0.52 | 23.39 | 0.01 | 24.46 |
| Total | 0.13 | 0.07 | 0.15 | 0.96 | 0.25 | 0.96 | 1.88 | 0.42 | 1.90 | 85.95 | 48.67 | 87.38 |

# Computing the skyline with R-trees

- If we have an index over the ranking attributes, we can use it to avoid scanning the whole DB
- The BBS (Branch and Bound Skyline) algorithm [PTF+03] is reminiscent of kNNOptimal, in that it accesses index nodes by increasing values of MinDist (in the following the query/target point coincides with the origin) and of next-NN, in that the queue PQ keeps both tuples and nodes
  - For computational economy, [PTF+03] evaluates distances using $L_1$ (Manhattan distance)
- The basic objective of the algorithm is to avoid accessing index nodes that cannot contain any skyline object
- To this end it exploits the following simple observation:

  If the region Reg(N) of node N completely lies in the dominance region of a tuple t, then N cannot contain any skyline point ("t dominates N")




- It also exploits the (now well-known) fact that **if $L_1(t',0) \geq L_1(t,0)$ then $t' \not\prec t$**
- PQ also stores key(N), i.e., the MBR of N, in order to check if N is dominated by some tuple t



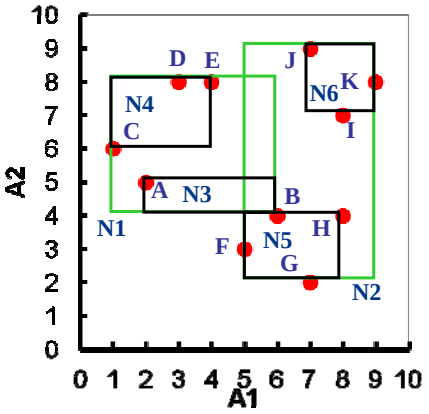
# The BBS algorithm

**Input**: index tree with root node RN
**Output**: Sky, the skyline of the indexed data

```
Initialize PQ with [ptr(RN),Dom(R),0];    // starts from the root node
Sky := ∅;                                 // the Skyline is initially empty
while PQ ≠ ∅:                             // until the queue is not empty…
    [ptr(Elem), key(Elem), dMIN(0,Reg(Elem))] := DEQUEUE(PQ);
    If no point in Sky dominates Elem then:
        if Elem is a tuple t then: Sky := Sky ∪ {t}
            else: { Read(Elem);   // …node Elem might contain skyline points
                if Elem is a leaf then: { for each tuple t in Elem:
                    if no tuple in Sky dominates t then:
                        ENQUEUE(PQ,[ptr(t), key(t), L1(0,key(t))]) }
                    else: { for each child node Nc of Elem:
                    if no point in Sky dominates Nc then:
                        ENQUEUE(PQ,[ptr(Nc), key(Nc), dMIN(0,Reg(Nc))])
    }};
return Sky;
end.
```

# BBS: An example (1/2)

- distance: L1

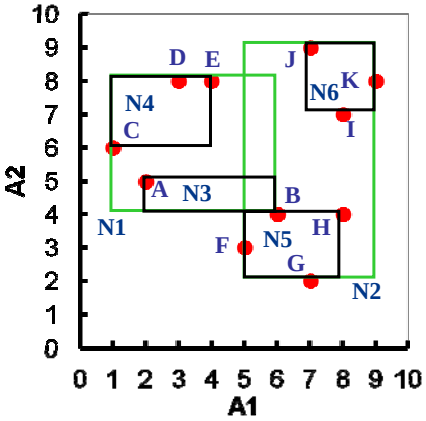



| Elem | $d_{MIN}$ |
|---|---|
| A | 7 |
| B | 10 |
| C | 7 |
| D | 11 |
| E | 12 |
| F | 8 |
| G | 9 |
| H | 12 |
| I | 15 |
| J | 16 |
| K | 17 |
| N1 | 5 |
| N2 | 7 |
| N3 | 6 |
| N4 | 7 |
| N5 | 7 |
| N6 | 14 |



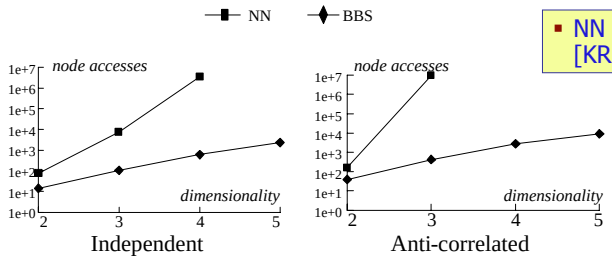
# BBS: An example (2/2)

| Action | PQ | | | |
|---|---|---|---|---|
| Read(RN) | (N1,5) | (N2,7) | | |
| Read(N1) | (N3,6) | (N4,7) | (N2,7) | |
| Read(N3) | (A,7) | (N4,7) | (N2,7) | (B,10) |
| Return(A) | (N4,7) | (N2,7) | (B,10) | |
| Read(N4) | (C,7) | (N2,7) | (B,10) | |
| Return(C) | (N2,7) | (B,10) | | |
| Read(N2) | (N5,7) | (B,10) | | |
| Read(N5) | (F,8) | (G,9) | (B,10) | |
| Return(F) | (G,9) | (B,10) | | |
| Return(G) | (B,10) | | | |

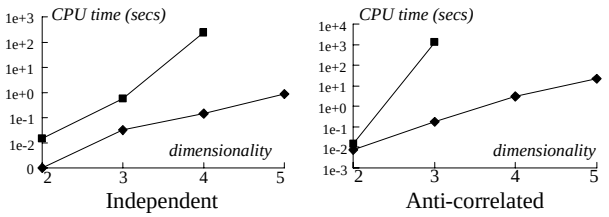



- The example clearly shows why a tuple currently undominated, such as B, which is stored in N3, needs to be inserted into the queue

# Experimental results (from [PTF+03])




Node accesses vs. $d$ ($N$=1M)




CPU-time vs. $d$ ($N$=1M)

- NN is an algorithm from [KRR02], also based on R-trees

**Experimental setup**
Independent (uniform) and anti-correlated datasets

dimensionality $\in$ [2,5]
cardinality N=1M tuples

Node size = 4Kbytes
(C = 204 when d=2;
C = 94 when d=5)

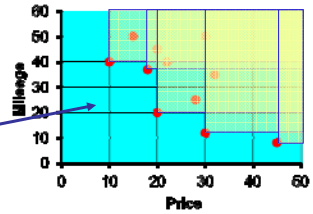
Pentium 4, 2.4GHz CPU
512Mbytes RAM



# Correctness and Optimality of BBS

- The correctness of BBS is easy to prove, since the algorithm only discards nodes that are found to be dominated by some point in the Skyline
- As SFS and SaLSa, when a tuple t is inserted into Sky, then t is guaranteed to be part of the final result
  - This is a direct consequence of accessing nodes by increasing values of MinDist and of inserting a tuple into Sky only when it becomes the first element of PQ
- Optimality of BBS (which we do not formally prove) means:
  BBS only reads those nodes that intersect the "Skyline search region"; this is the complement of the union of the dominance regions of skyline points

The Skyline search region



# Variants of skyline queries

- [PTF+03] introduces some variants of basic skyline queries:

1. **Ranked skyline queries**
   ranking within the skyline with a scoring function
2. **Constrained skyline queries**
   limiting the search region
3. **K-dominating queries**
   the k tuples that dominate the largest number of other tuples

Dimitris Papadias

- Many other skyline-related problems have been proposed/studied so far, e.g.:
  - Reverse skyline queries: given a query point q, which are the tuples t such that q is in the skyline computed with respect to t (when t is the target)?
  - Representative skyline points: which are the k "most representative" points in the skyline?



# Recap

- Skyline queries represent a valid alternative to top-k queries, since they do not require any choice of scoring functions and weights
- The skyline of a relation R, Sky(R), contains all and only the undominated tuples in R, i.e., those tuples representing "interesting alternatives" to consider
- Computing Sky(R) can rely on both sequential and index-based algorithms
- The BNL algorithm works by allocating a main-memory window, and then comparing incoming tuples with those in the window. Several iterations are usually needed
- SFS pre-sorts data yielding a topological sort that introduces several benefits compared to BNL
- SaLSa adds a stop condition, that avoids reading all the data
- BBS is a provably I/O-optimal algorithm for computing Sky(R) using an R-tree