Multimedia Database Systems

Top-k and Skyline Computation

Outline of Presentation

- Introduction
- Top-*k* computation
  - Algorithm FA (Fagin's algorithm)
  - Algorithm TA (threshold algorithm)
  - Advanced topics
- Skyline computation
  - Introduction to R-trees
  - Algorithm BBS (branch-and-bound skyline)
  - Advanced topics
- Conclusions
- Bibliography

Introduction

In a database management system queries are usually expressed by using SQL. For example, to find the hotel names whose distance from the beach is at most 1km, we can write the following SQL statement:

```
SELECT hotels.name
FROM hotels
WHERE hotels.distance <= 1;
```

Introduction

In the previous query we specify exactly what we want, by stating that “hotels.distance <= 1”.

However, in many cases it is more convenient to let the system give the best possible answers it can get for us. This is even more helpful when multiple criteria are given by the user.

Introduction

Who is the best NBA player?

According to **points**: Tracy McGrady, score 2003
According to **rebounds**: Shaquille O'Neal, score 760
According to **points+rebounds**: Tracy McGrady, score 2487

Name	Points	Rebounds	Assists	Steals	……
Tracy McGrady	2003	484	448	135	……
Kobe Bryant	1819	392	398	86	……
Shaquille O'Neal	1669	760	200	36	……
Yao Ming	1465	669	61	34	……
Dwyane Wade	1854	397	520	121	……
Steve Nash	1165	249	861	74	……
……	……	……	……	……	……

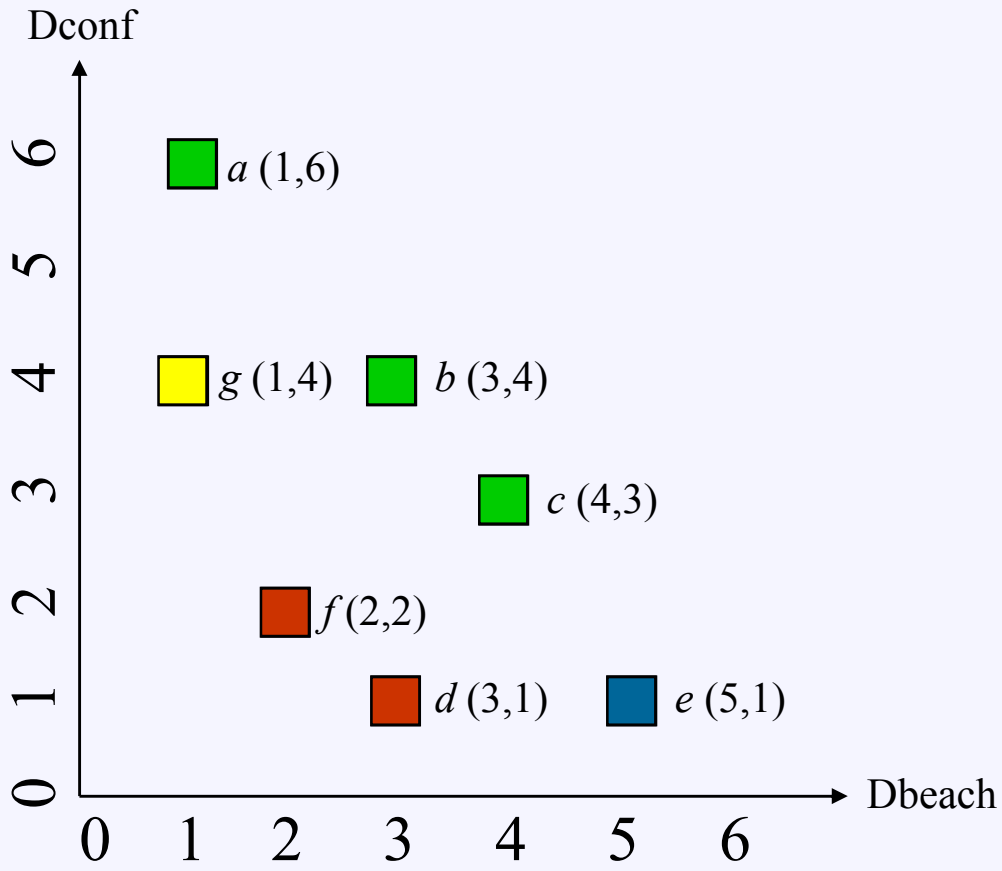
Introduction

Assume we are interested in hotels that are close to the beach AND close to a conference center. For a hotel x let Dbeach(x) denote the distance to the beach and Dconf(x) the distance to the conference center.

Assume further that we want the value Dbeach(x)+Dconf(x) to be the minimum possible.

Introduction

Object ranking based on Dbeach(x)+Dconf(x)



id	Dbeach	Dconf	Score
a	1	6	7
b	3	4	7
c	4	3	7
d	3	1	4
e	5	1	6
f	2	2	4
g	1	4	5

Best hotels: *d*, *f*
Second best: *g*
Third best: *e*
Next best: *a*, *b*, *c*

Introduction – Top-*k*

Top-*k* Query

Given a database D of n objects, a scoring function F (according to which we rank the objects in D) and the number of expected answers k, a Top-k query returns the k objects with the best score (rank) in D.

In our hotel example, the scoring function F(x) is simply the sum of Dbeach(x)+Dconf(x).

Introduction – Top-*k*

Monotonicity property

Assume we have two vectors X and Y

$X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_m)$

A scoring function F() is called monotone increasing if it preserves the order:

$$x_1 <= y_1, \ldots, x_m < y_m \rightarrow F(X) <= F(Y)$$

Examples: min, max, sum

Introduction – Top-*k*

Remarks

- The number of objects in the answer (k) is user-defined.
- The "best" score is either the lowest or the highest depending on user preferences.
- The ranking function F, may involve more than two attributes.

Introduction – Top-*k*

Top-*k* query in SQL

SELECT hotels.name
FROM hotels
ORDER BY (hotels.Dbeach+hotels.Dconf)
STOP AT 3

This is a Top-3 query.

Introduction – Top-*k*

In a Top-*k* query the ranking function F as well as the number of answers *k* must be provided by the user.

In many cases it is difficult to define a meaningful ranking function, especially when the attributes have different semantics (e.g., find the cheapest hotel closer to the beach).

Introduction – Skyline

To avoid the drawbacks of Top-*k* queries, Skyline queries have been proposed as an alternative to satisfy user preferences.

The Skyline query

- does not require a ranking function
- does not need the integer *k*

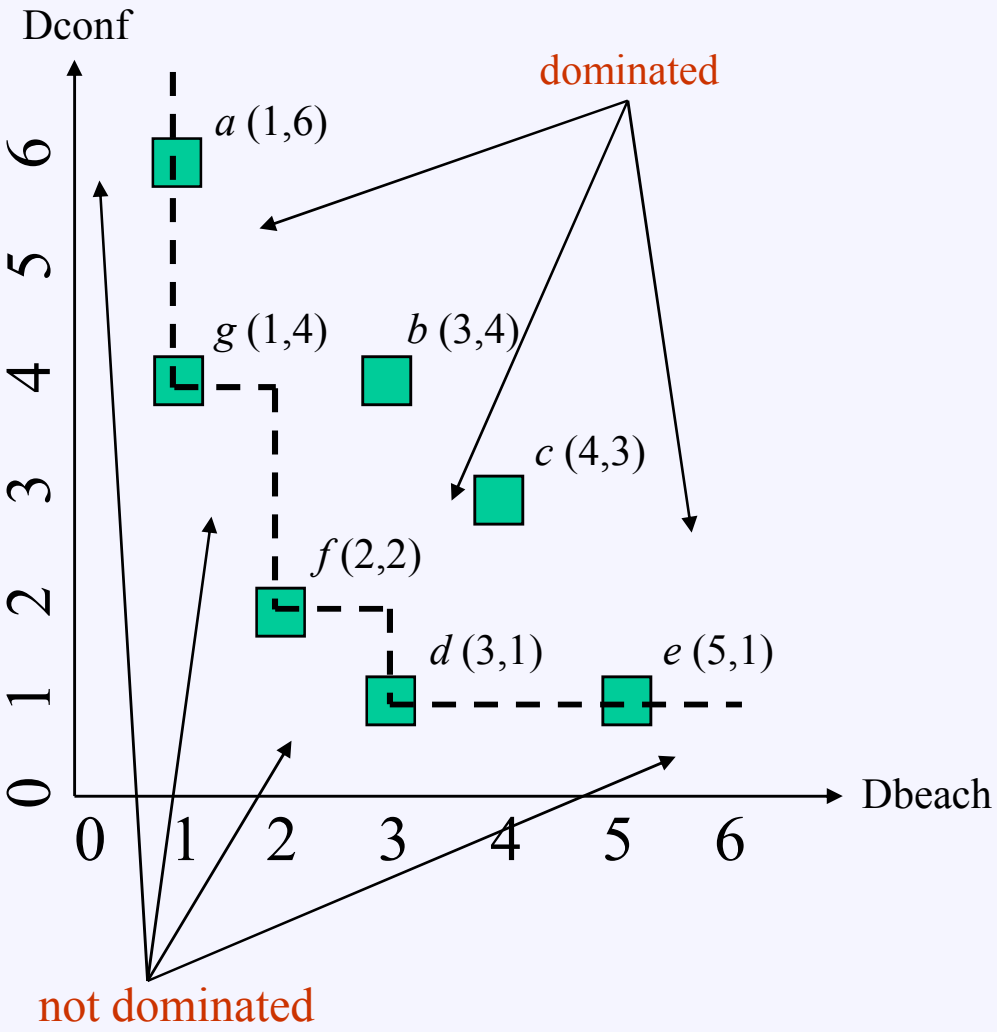
Introduction - Skyline

The Skyline of a set of objects (records) comprises all records that are not dominated by any other record.

A record *x* dominates another record *y* if *x* is as good as *y* in all attributes and strictly better in at least one attribute.

Again, in some cases we are interested in minimizing attribute values (e.g., price) and in other cases maximizing them (e.g., floor number)

Introduction - Skyline



Domination examples:

g dominates *b* because 1<3 and 4=4
f dominates *c* because 2<4 and 2<3
d dominates *e* because 3<5 and 1=1

The Skyline is the set: { *g*, *f*, *d* }

These objects are not dominated by any other object.

Introduction - applications

E-commerce

"I want to buy a PDA which is as cheap as possible, has large memory capacity and it is light-weighted"

For Top-*k* queries we should also provide the number *k* (how many PDAs we want in the answer) and the ranking function.

Introduction - applications

Multimedia Databases

"Give me the 3 images that have the highest resolution, they are red and depict flowers"

Introduction - applications

Web Information Retrieval

Let *M* be a meta search engine which uses yahoo and google. Both search engines return a set of results ranked by relevance.

yahoo

id	score
a	0.9
c	0.7
b	0.6

google

id	score
b	0.8
d	0.7
a	0.6

The challenge is to combine the results of all search engines in order to give a total ranking of the documents.

Introduction – naïve methods

It is possible to process Top-*k* and Skyline queries by using simple algorithmic techniques. However, although these techniques are easily implemented, they suffer from performance degradation due to their large complexities.

Introduction – naïve methods

Top-k processing

- Apply the ranking function F to all objects
- Sort the objects with respect to their score
- Return the *k* best objects

Disadvantages:

Sorting is an expensive operation requiring a complexity of O(n log$n$) for n elements. Usually, k is very small in comparison to the number of objects, so we pay too much!

Introduction – naïve methods

Skyline processing

- For each object, check if it is dominated by any other object
- Return the objects that are not dominated

Disadvantages:

Requires scanning the whole database for each object. Complexity O(n^2). This is not convenient in systems with large volumes of data.

Introduction - motivation

Since naïve methods do not perform well for large sets of objects, the challenge is to devise new algorithms in order to process Top-*k* and Skyline queries efficiently.

Goals:

- avoid sorting operations in Top-*k*
- avoid scanning the whole database in Skyline

Outline of Presentation

- Introduction
- Top-*k* computation
  - Algorithm FA (Fagin's algorithm)
  - Algorithm TA (threshold algorithm)
  - Advanced topics
- Skyline computation
  - Introduction to R-trees
  - Algorithm BBS (branch-and-bound skyline)
  - Advanced topics
- Conclusions
- Bibliography

Top-*k* Computation

Application: multimedia information retrieval

- We have an image database composed of n image objects $O_1, O_2, \ldots, O_n$.
- Each object O_i is described by m attributes $a_{i1}, a_{i2}, \ldots, a_{im}$.
- Therefore, each object is a vector in the m-th dimensional space.
- We assume that the total score of an image is the sum of the individual scores of all attributes.

Top-*k* Computation

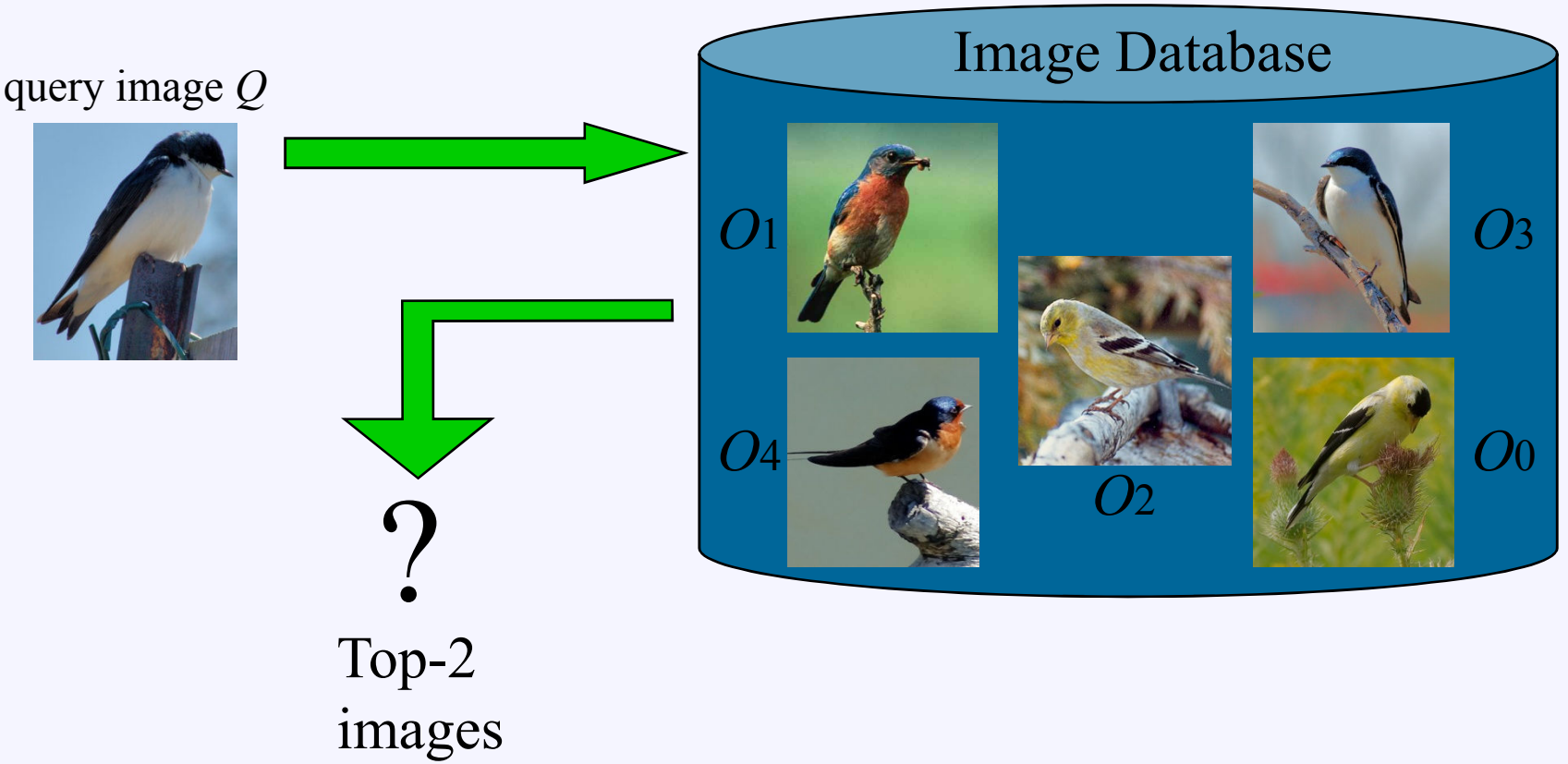
A user issues a Top-2 query:

"Given the query image *Q*, retrieve the 2 images from the database that best match the query"

(This is a typical query in content-based retrieval of images)

Top-*k* Computation

Assume that the database contains only 5 image objects O_0, O_1, O_2, O_3 and O_4.



Top-*k* Computation

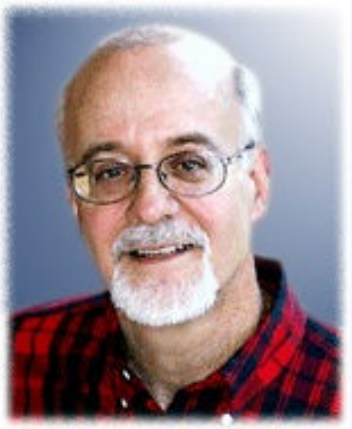
The database can be considered as an *n* x *m* score matrix, storing the score values of every object in every attribute.

a1	a2	a3	a4	a5
O_3, 99	O_1, 91	O_1, 92	O_3, 74	O_3, 67
O_1, 66	O_3, 90	O_3, 75	O_1, 56	O_4, 67
O_0, 63	O_0, 61	O_4, 70	O_0, 56	O_1, 58
O_2, 48	O_4, 07	O_2, 16	O_2, 28	O_2, 54
O_4, 44	O_2, 01	O_0, 01	O_4, 19	O_0, 35

Note that, for each attribute scores are sorted in descending order.

Top-*k* Computation – FA algorithm

Fagin's Algorithm (FA) is the first important contribution in the area.



The algorithm is based on two types of accesses:

Sorted access on attribute a_i: retrieves the next object in the sorted list of a_i

Random access on attribute a_i: gives the value of the i-th attribute for a specific object identifier.

Top-*k* Computation – FA algorithm

Outline of FA

Step 1:

- Read attributes from every sorted list using sorted access.
- Stop when *k* objects have been seen in common from all lists.

Step 2:

- Use random access to find missing scores.

Step 3:

- Compute the scores of the seen objects.
- Return the *k* highest scored objects.

Top-*k* Computation – FA algorithm

Step 1:

- Read attributes from every sorted list using sorted access
- Stop when *k* objects have been seen in common from all lists

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5
*O*3	99	90	75	74	67
*O*1	66	91	92	56	58
*O*4			70		67
*O*0	63	61		56	

No more sorted accesses are required, since we have determined *k*=2 objects contained in all lists (objects *O*1 and *O*3).

Top-*k* Computation – FA algorithm

Step 2:

- Use random access to find missing scores

a1	a2	a3	a4	a5
*O*3, 99	*O*1, 91	*O*1, 92	*O*3, 74	*O*3, 67
*O*1, 66	*O*3, 90	*O*3, 75	*O*1, 56	*O*4, 67
*O*0, 63	*O*0, 61	*O*4, 70	*O*0, 56	*O*1, 58
*O*2, 48	*O*4, 07	*O*2, 16	*O*2, 28	*O*2, 54
*O*4, 44	*O*2, 01	*O*0, 01	*O*4, 19	*O*0, 35

id	a1	a2	a3	a4	a5
*O*3	99	90	75	74	67
*O*1	66	91	92	56	58
*O*4	44	07	70	19	67
*O*0	63	61	01	56	35

All missing values for seen objects have been determined.
Therefore, no more random accesses are required.

Top-*k* Comput…orithm

Step 3:

…en obj…
…bjects…

	a4
	74
	56
	19
	56

…Total Score

- 405
- 363

} Top-2

- 207
- 216

Therefore, the two images th…image are:
O_3 with score 405 and O_1 w…

Top-*k* Computation – TA algorithm

Fagin and his colleagues performed some enhancements to FA, leading to algorithm TA (Threshold Algorithm).

The main contribution of this new algorithm is the incorporation of a threshold to determine when to stop scanning the sorted lists.

Top-*k* Computation – TA algorithm

Outline of TA

Step 1:

- Read attributes from every sorted list using sorted access.
- For each object seen *x*:
  - Use random access to find missing values.
  - Determine the score F(*x*) of object *x*.
  - If the object is among the top-*k* keep it in buffer.

Step 2:

- Determine threshold value *T* based on objects currently seen under sorted access.
  $T = a1(p) + a2(p) + \ldots + am(p)$ where p is the current sorted access position.
- If there are *k* objects with total scores >= *T* then STOP and report answers
  else $p = p + 1$ and GOTO Step1.

Top-*k* Computation – TA algorithm

Step 1:

- Read attributes from every sorted list using sorted access.
- For each object seen *x*:
  - Use random access to find missing values.
  - Determine the score F(*x*) of object *x*.
  - If the object is among the top-*k* keep it in buffer.

BUFFER:
($O3$, 405)
($O1$, 363)

p=1

a1	a2	a3	a4	a5
$O3$, 99	$O1$, 91	$O1$, 92	$O3$, 74	$O3$, 67
$O1$, 66	$O3$, 90	$O3$, 75	$O1$, 56	$O4$, 67
$O0$, 63	$O0$, 61	$O4$, 70	$O0$, 56	$O1$, 58
$O2$, 48	$O4$, 07	$O2$, 16	$O2$, 28	$O2$, 54
$O4$, 44	$O2$, 01	$O0$, 01	$O4$, 19	$O0$, 35

id	a1	a2	a3	a4	a5	F
$O3$	99	90	75	74	67	**405**
$O1$	66	91	92	56	58	**363**

Top-*k* Computation – TA algorithm

Step 2:

- Determine threshold value T based on objects currently seen under sorted access. $T = a1(p) + a2(p) + \ldots + am(p)$ where p is the current sorted access position.
- If there are k objects with total scores >= T then STOP and report answers else $p = p + 1$ and GOTO Step1.

BUFFER:
($O3$, 405)
($O1$, 363)

p=1

a1	a2	a3	a4	a5
$O3$, 99	$O1$, 91	$O1$, 92	$O3$, 74	$O3$, 67
$O1$, 66	$O3$, 90	$O3$, 75	$O1$, 56	$O4$, 67
$O0$, 63	$O0$, 61	$O4$, 70	$O0$, 56	$O1$, 58
$O2$, 48	$O4$, 07	$O2$, 16	$O2$, 28	$O2$, 54
$O4$, 44	$O2$, 01	$O0$, 01	$O4$, 19	$O0$, 35

id	a1	a2	a3	a4	a5	F
$O3$	99	90	75	74	67	405
$O1$	66	91	92	56	58	363

$T = 99+91+92+74+67 = 423$

There are NO k objects with a score >= T, GOTO Step1 …

Top-*k* Computation – TA algorithm

Step 1 (second execution):

- Read attributes from every sorted list using sorted access.
- For each object seen *x*:
  - Use random access to find missing values.
  - Determine the score F(*x*) of object *x*.
  - If the object is among the top-*k* keep it in buffer.

BUFFER:
(O_3, 405)
(O_1, 363)

p=2

a1	a2	a3	a4	a5
O_3, 99	O_1, 91	O_1, 92	O_3, 74	O_3, 67
O_1, 66	O_3, 90	O_3, 75	O_1, 56	O_4, 67
O_0, 63	O_0, 61	O_4, 70	O_0, 56	O_1, 58
O_2, 48	O_4, 07	O_2, 16	O_2, 28	O_2, 54
O_4, 44	O_2, 01	O_0, 01	O_4, 19	O_0, 35

id	a1	a2	a3	a4	a5	F
O_3	99	90	75	74	67	**405**
O_1	66	91	92	56	58	**363**
O_4	44	07	70	19	67	**207**

Top-*k* Computation – TA algorithm

Step 2 (second execution):

- Determine threshold value T based on objects currently seen under sorted access. $T = a1(p) + a2(p) + \ldots + am(p)$ where p is the current sorted access position.
- If there are k objects with total scores >= T then STOP and report answers else $p = p + 1$ and GOTO Step1.

BUFFER:
($O3$, 405)
($O1$, 363)

p=2 ⇒

a1	a2	a3	a4	a5
$O3$, 99	$O1$, 91	$O1$, 92	$O3$, 74	$O3$, 67
$O1$, 66	$O3$, 90	$O3$, 75	$O1$, 56	$O4$, 67
$O0$, 63	$O0$, 61	$O4$, 70	$O0$, 56	$O1$, 58
$O2$, 48	$O4$, 07	$O2$, 16	$O2$, 28	$O2$, 54
$O4$, 44	$O2$, 01	$O0$, 01	$O4$, 19	$O0$, 35

id	a1	a2	a3	a4	a5	F
$O3$	99	90	75	74	67	**405**
$O1$	66	91	92	56	58	**363**
$O4$	44	07	70	19	67	**207**

$T = 66+90+75+56+67 = 354$

Both objects in the buffer have scores higher than T. STOP and report answers.

Top-*k* Computation - FA vs TA

- TA sees less objects than FA
  - TA stops at least as early as FA
    - When we have seen *k* objects in common in FA, their scores are higher or equal than the threshold in TA.

- TA may perform more random accesses than FA
  - In TA, (*m*-1) random accesses for each object.
  - In FA, random accesses are done at the end, only for missing scores.

- TA requires only bounded buffer space (*k*) at the expense of more random seeks.
- FA makes use of unbounded buffers.

Top-*k* Computation – other methods

Fagin et al proposed two significant variations:

- The NRA algorithm (No Random Access): the method uses only sorted accesses and never use random accesses.
- The CA algorithm (Combined Algorithm): this method is a combination of TA and NRA and yields better performance.

Top-*k* Computation - advanced topics I

Distributed Top-*k* computation

Data are frequently distributed across a number of machines. The challenge in such an environment is to determine the Top-k answers trying to minimize the network traffic and the latency.

Specialized algorithms have been proposed that work efficiently in a distributed environment.

Top-*k* Computation - advanced topics II

Complex Top-*k* queries

In some cases the Top-*k* ranking function should be evaluated only on records that satisfy a join condition. The challenge is to provide the Top-*k* joining records without scanning the whole database.

Top-*k* Computation - advanced topics III

Top-k queries on probabilistic data

In several applications there is uncertainty in the data. For example, values may be missing or we are not sure about an existing value. A challenging research direction is to investigate algorithms for Top-*k* computation in such a case.

Outline of Presentation

- Introduction
- Top-*k* computation
  - Algorithm FA (Fagin's algorithm)
  - Algorithm TA (threshold algorithm)
  - Advanced topics
- Skyline computation
  - Introduction to R-trees
  - Algorithm BBS (branch-and-bound skyline)
  - Advanced topics
- Conclusions
- Bibliography

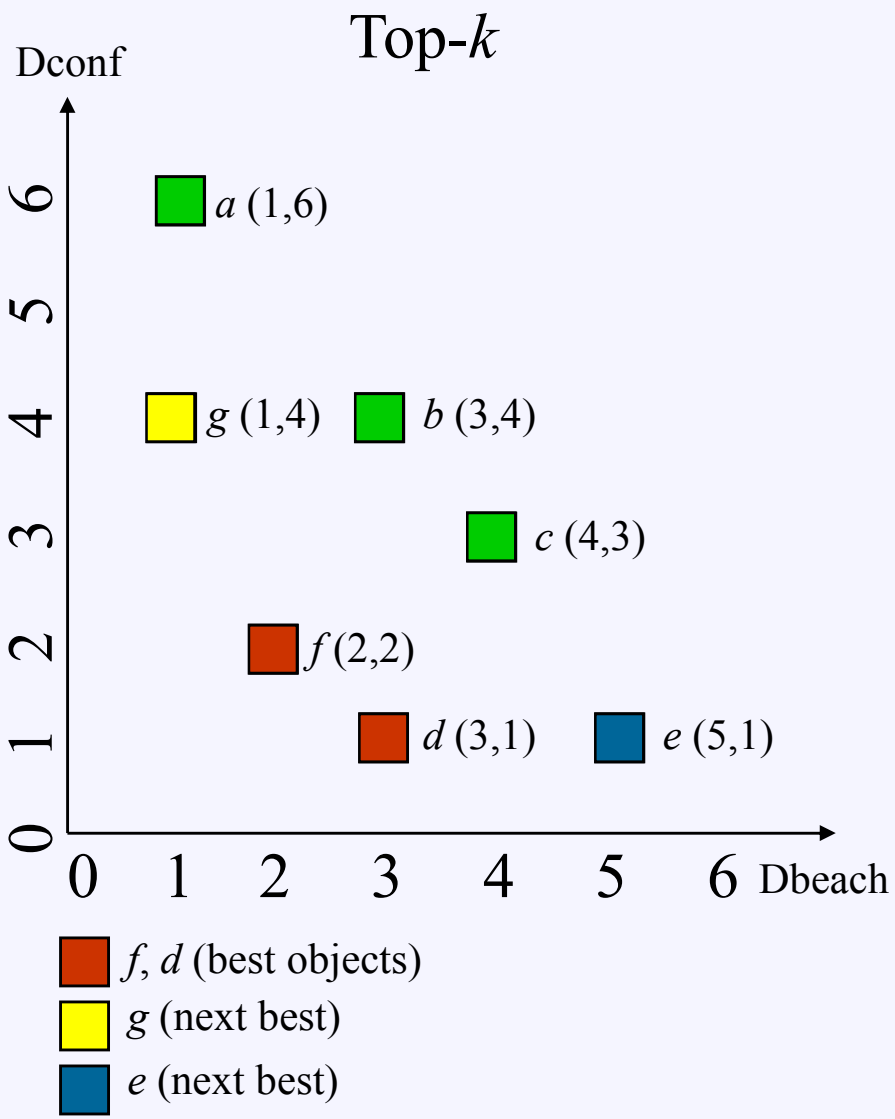
Skyline Computation

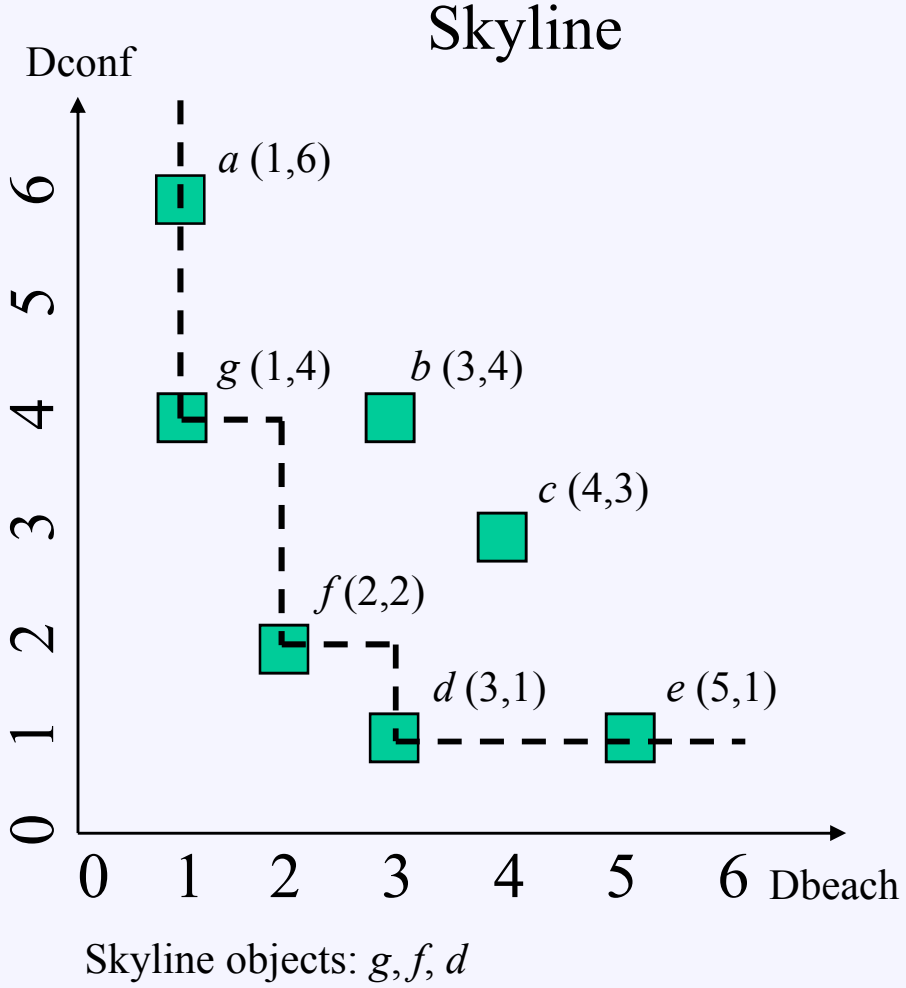
Remember that:

Top-*k* query processing requires a user-defined ranking function F and an integer *k* to declare the number of best objects in the answer.

On the other hand, Skyline query processing does NOT require any of these.

Skyline Computation





Skyline Computation

Some techniques:

- Nested Block Loop (NBL): perform a nested loop over all blocks of the data.
- Divide and Conquer (DC): partition the space in subspaces, solve the problem in the subspaces and then synthesize the solution in the whole space.
- Nearest-Neighbor based (NN): uses an R-tree index and performs a sequence of nearest-neighbor queries until all Skyline objects have been found.

Introduction to R-trees

- Many real-life applications require the organization and management of multidimensional data (e.g., each image is represented as a point in the 5-dimensional space).
- To enable efficient query processing, data should be organized by means of an indexing scheme which is used to speed-up processing.
- The index helps in reducing the number of inspected objects significantly, avoiding the sequential scan of the whole database.
- Indexing schemes for multidimensional data work in a similar manner to access methods for simple numeric data (e.g., B-trees and Hashing).

Introduction to R-trees

One of the most important contributions in the area of multidimensional indexing is due to Antonin Guttman which invented the R-tree.

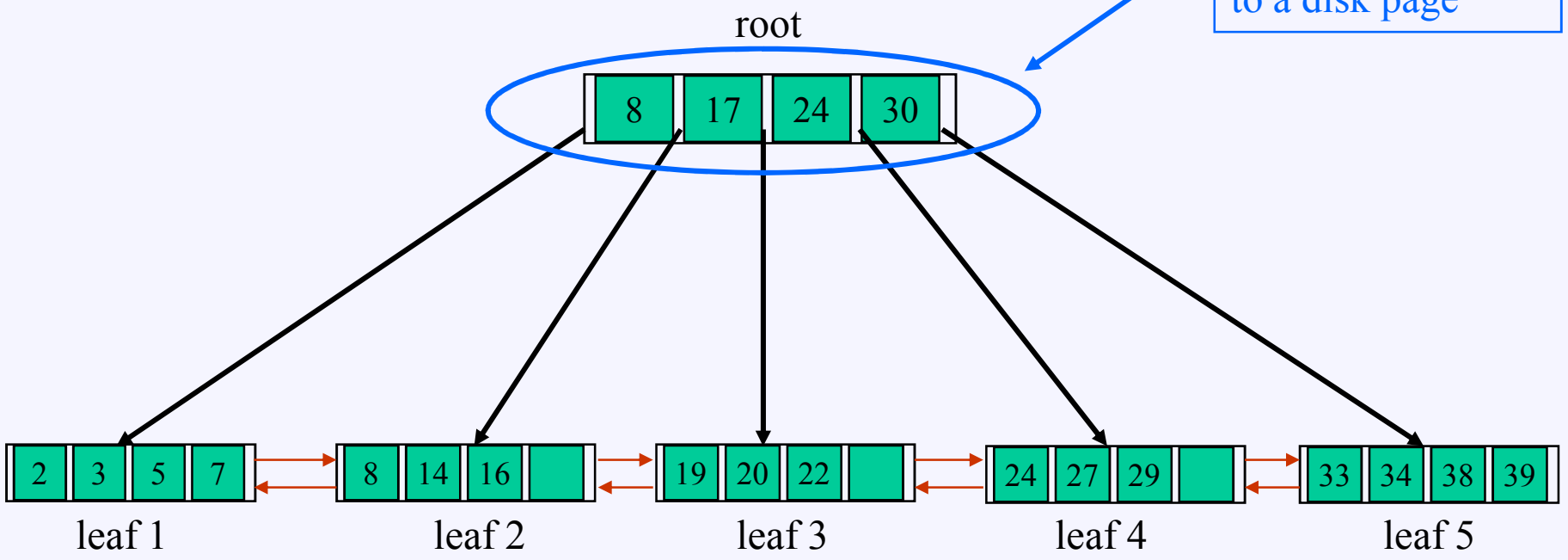
His work:
"R-trees: a dynamic index structure for spatial searching",
ACM SIGMOD Conference 1984

has received more than 2,900 citations
(source google scholar)

Introduction to R-trees

The R-tree can be viewed as an extension of the B+-tree to handle multiple dimensions. Recall that, a B+-tree is used to organize numeric data in one dimension only.

B+ tree example with 6 nodes:



Introduction to R-trees

R-trees have been extensively used in spatial databases to organize points and rectangles. They show excellent performance in processing interesting queries such as:

Range query: return the points that are contained in a specified region.

K-nearest-neighbor: given a point *p* and an integer *k* return the *k* objects closer to *p*.

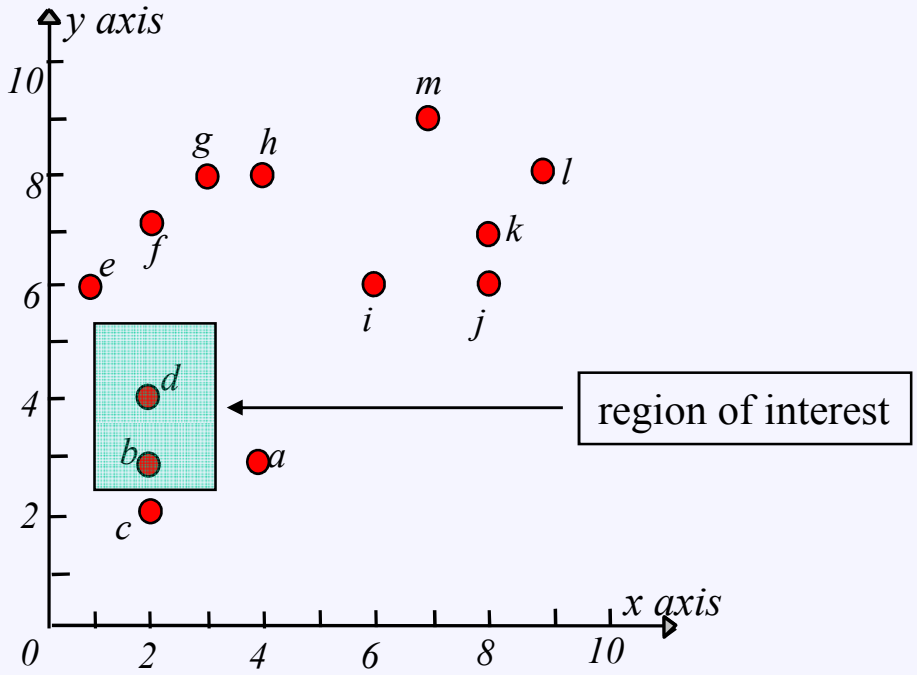
Introduction to R-trees

range query example:
which cities are within distance *R* from Amsterdam

k-NN query example:
Find the 3 cities closer to Utrecht ($k = 3$)

Introduction to R-trees

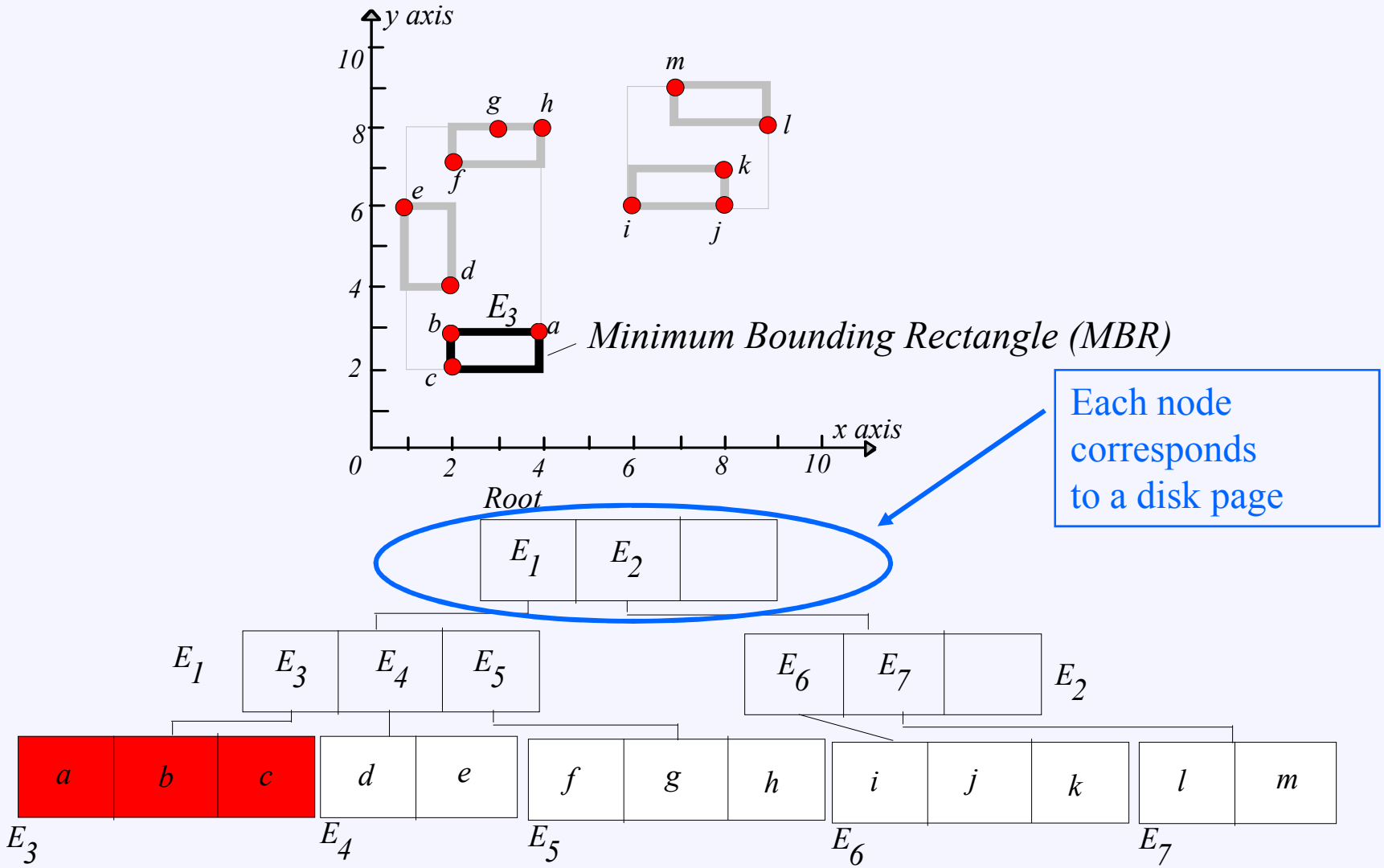
Example:
13 points in
2 dimensions



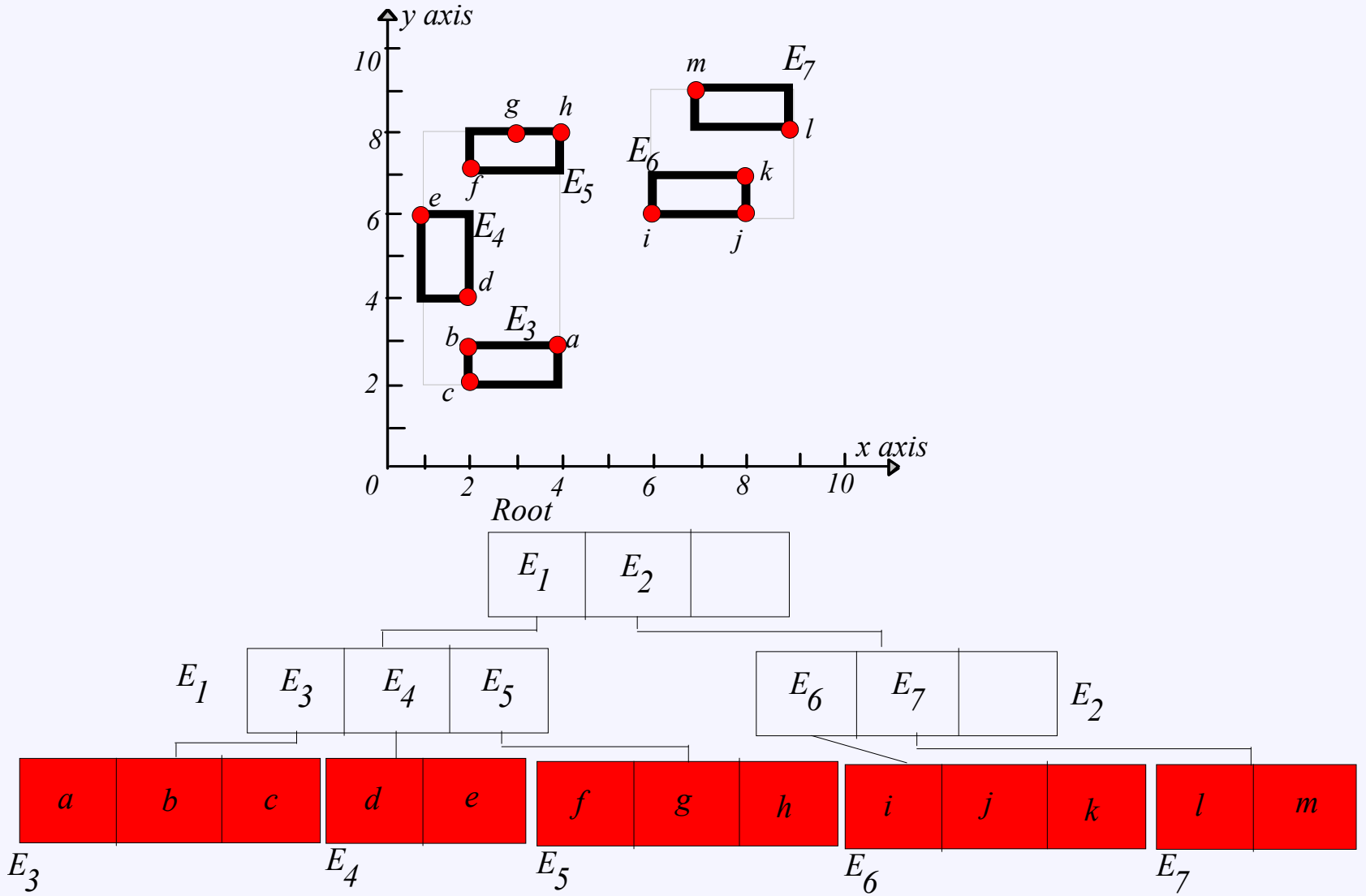
Range query example: *"find the objects in a given region"*.
E.g. find all hotels in Utrecht.

No index: scan through all objects. NOT EFFICIENT!

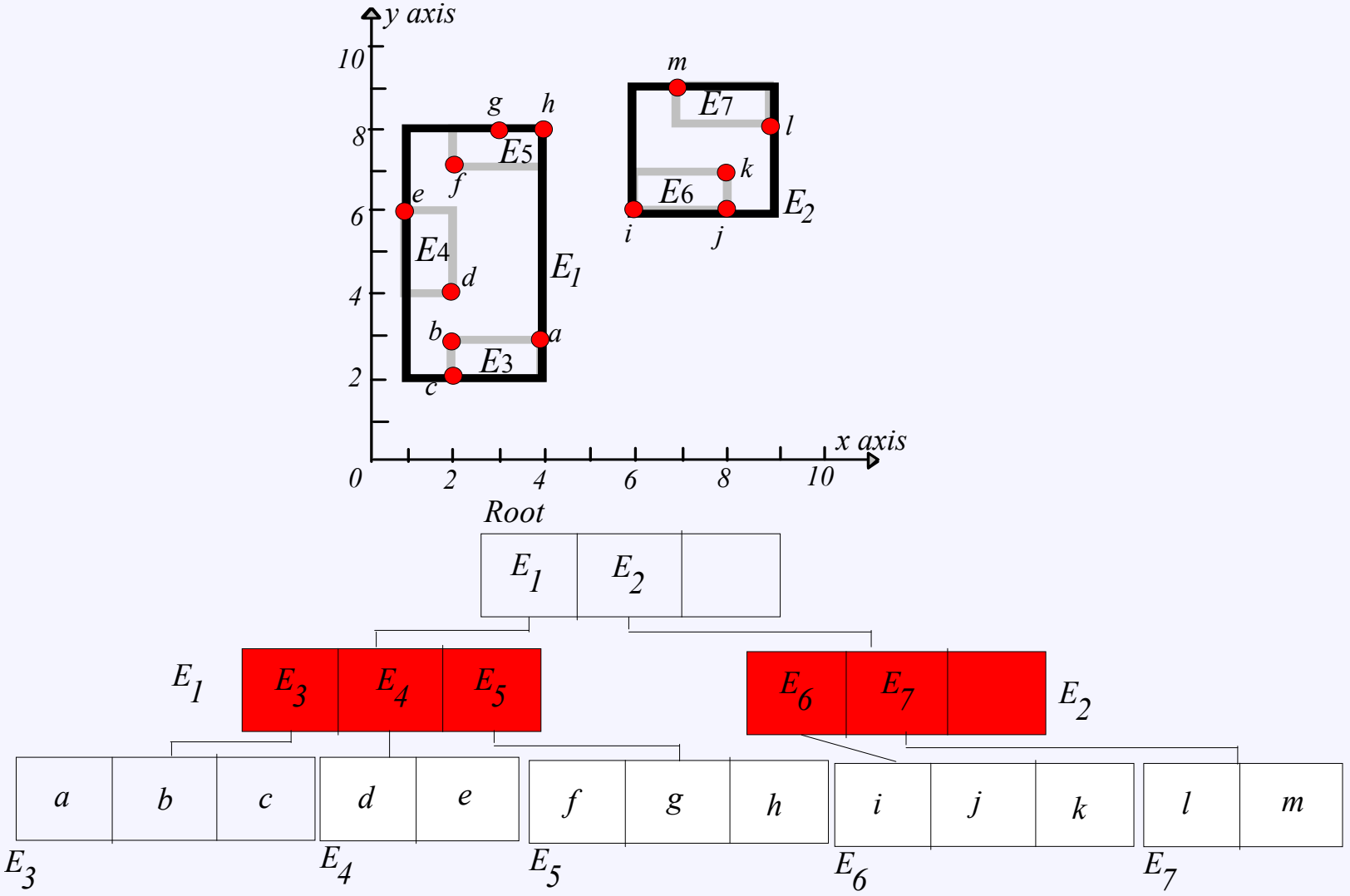
Introduction to R-trees – structure



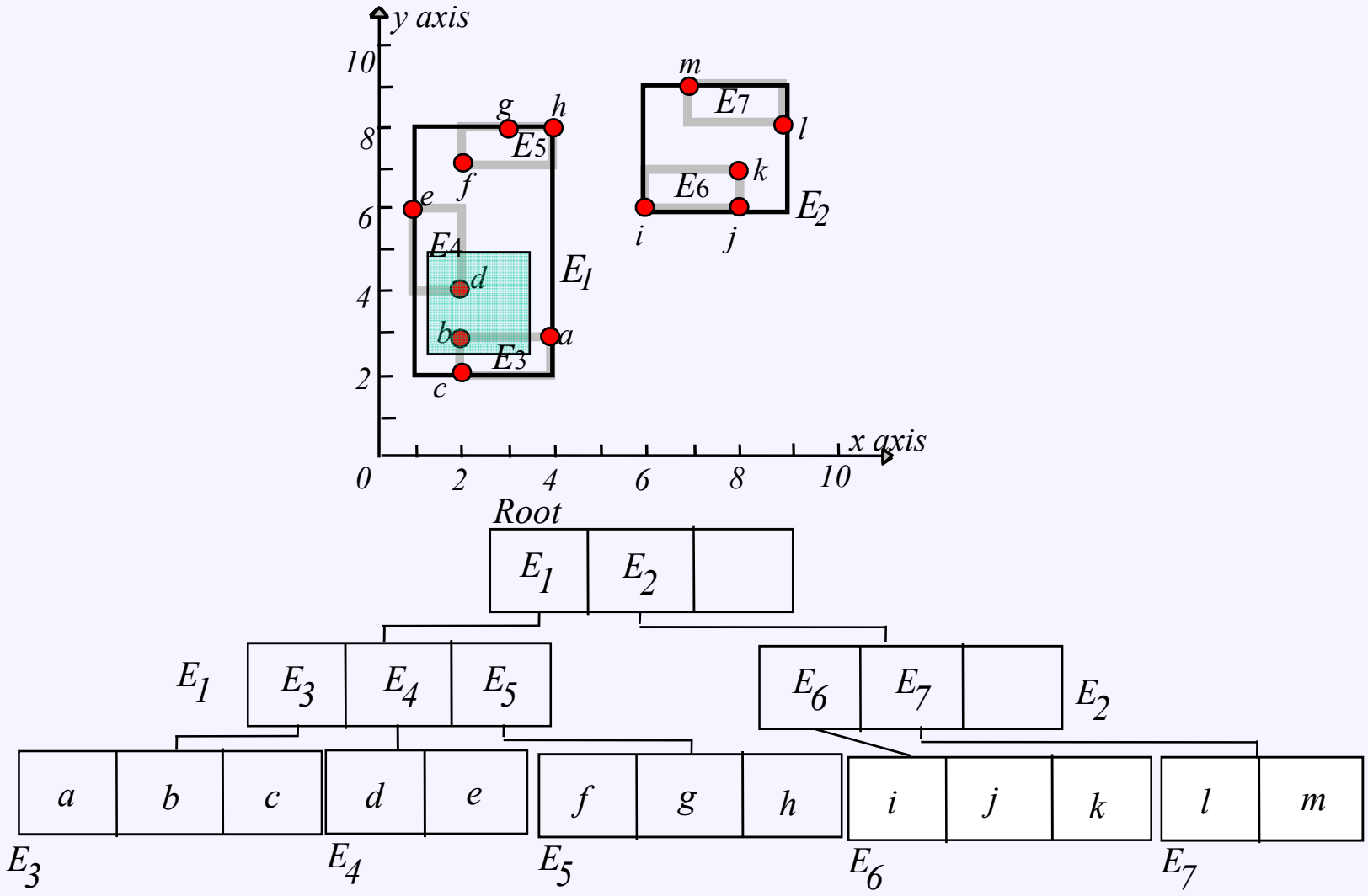
Introduction to R-trees – structure



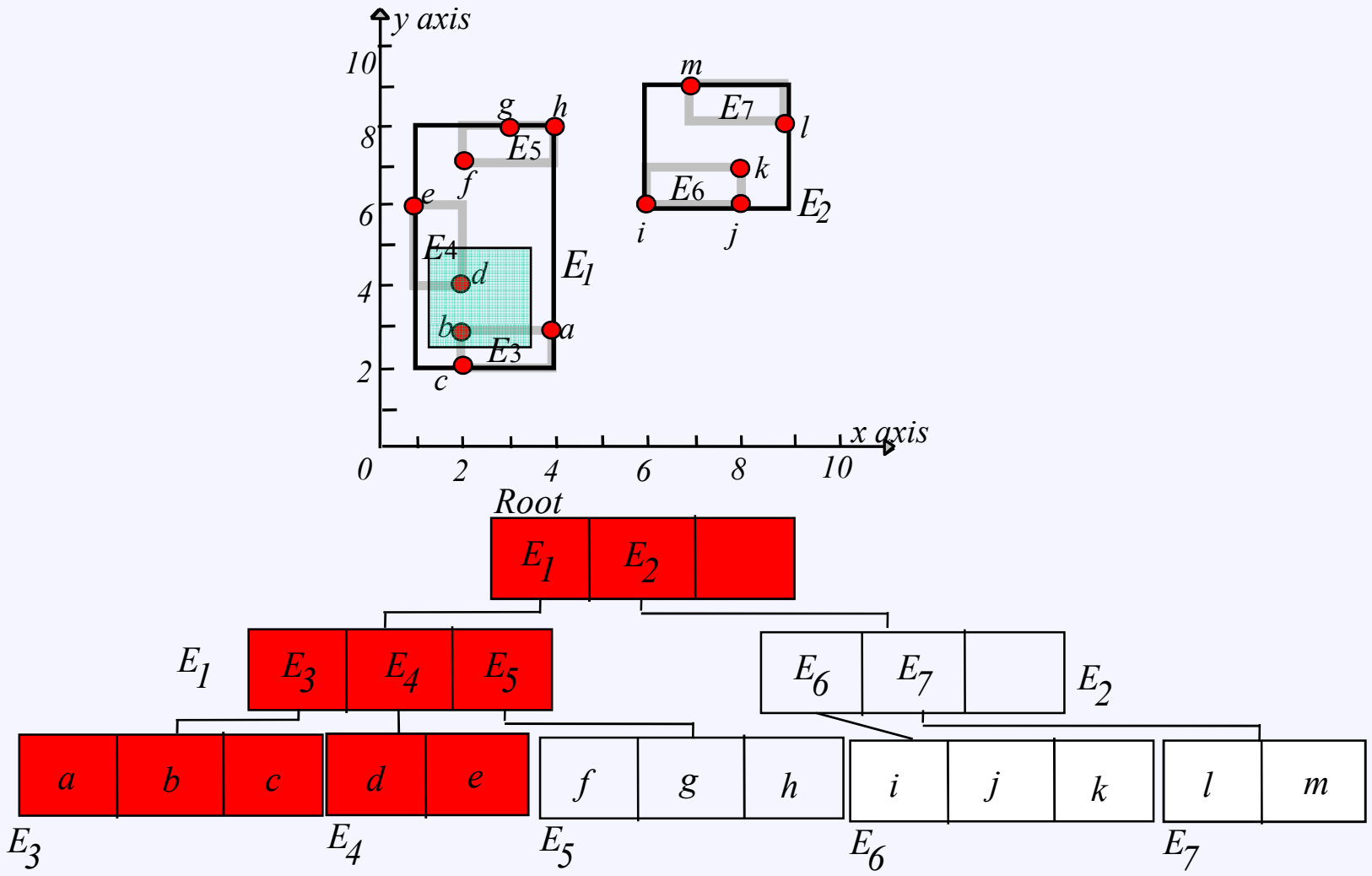
Introduction to R-trees – structure



Introduction to R-trees – range query



Introduction to R-trees – range query



BBS Algorithm – Basic Properties

Any Branch-and-Bound method requires two decisions:

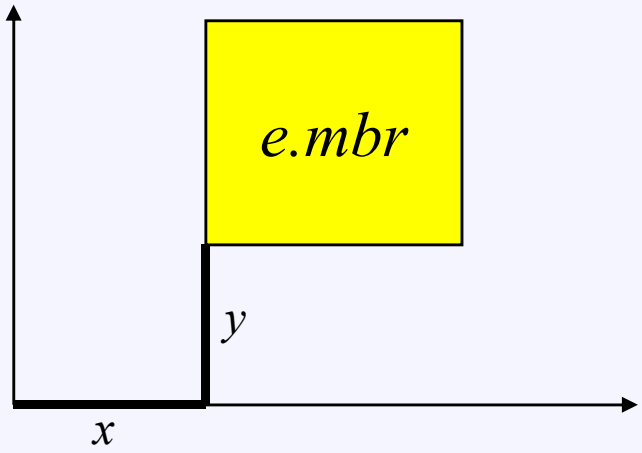
1. How to branch: which part of the space needs to be investigated next?

2. How to bound: which parts of the search space can be safely eliminated.

BBS Algorithm – basic properties

The algorithm uses a priority queue, where R-tree entries are prioritized by the mindist value. The mindist value of an entry *e,* is the cityblock (L_1) distance of its MBR's (*e.mbr*) lower-left corner to the origin.

For example:

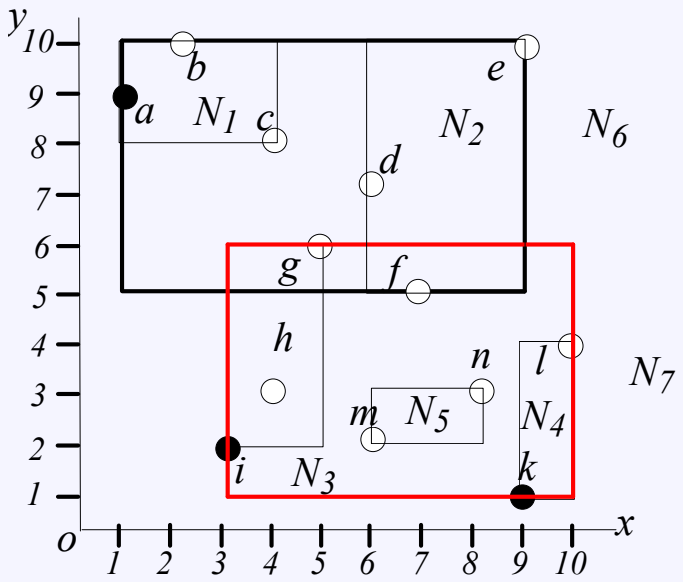


mindist(*e.mbr*) = x + y

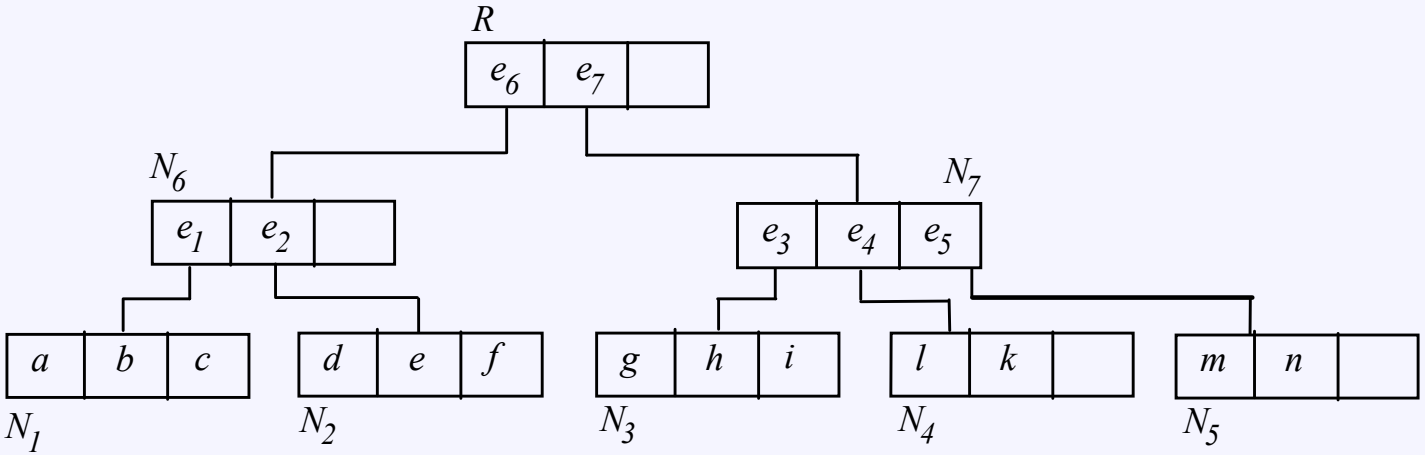
BBS Algorithm – basic properties

- The algorithm in every step chooses the best R-tree entry to check, according to the mindist measure. Upon visiting a node, the mindist of its entries is calculated and entries are inserted into the priority queue.
- The algorithm keeps the discovered skyline points in the set *S*.
- If the top of the queue is a data point, it is tested if it is dominated by any point in *S*. If yes it is rejected, otherwise it is inserted into *S*.

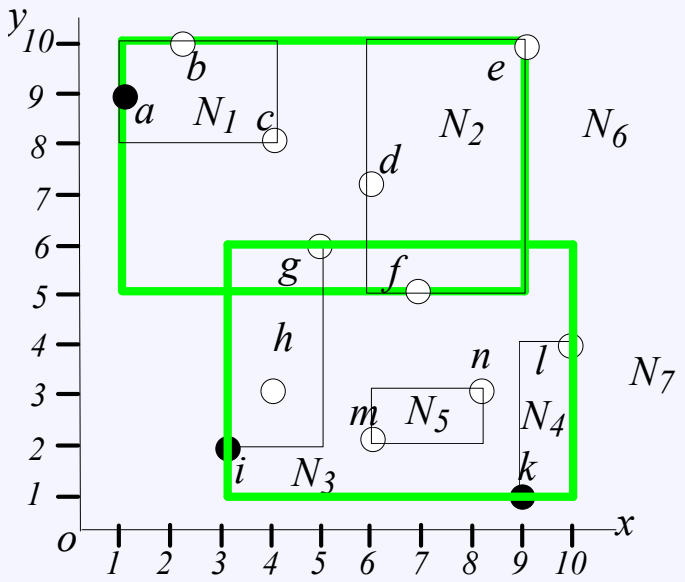
BBS Algorithm - example



- Assume all points are indexed in an R-tree.
- mindist(MBR) = the L_1 distance between its lower-left corner and the origin.

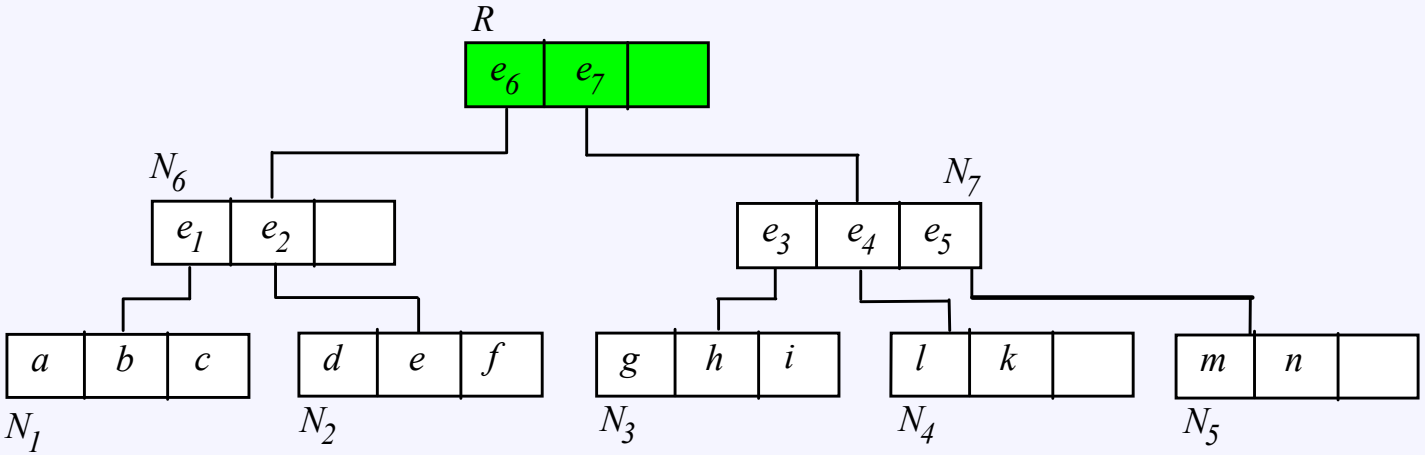


BBS Algorithm - example

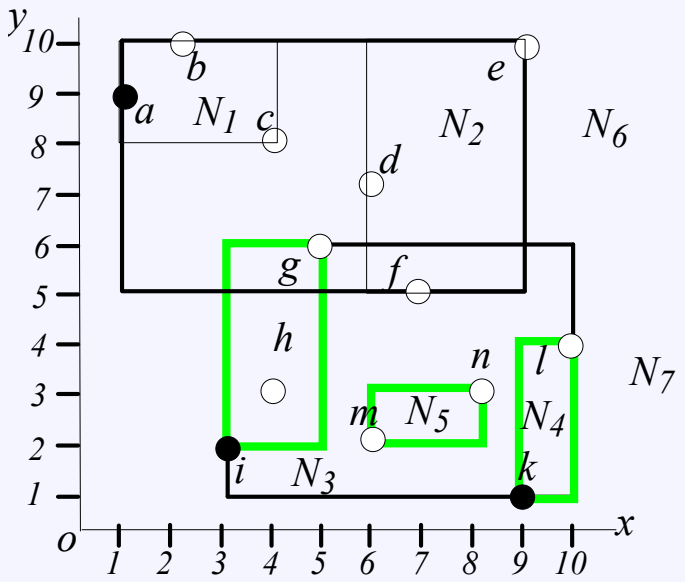


- Each heap entry keeps the mindist of the MBR.

action	**heap contents**	***S***
access root	$<e_7,4><e_6,6>$	$\varnothing$

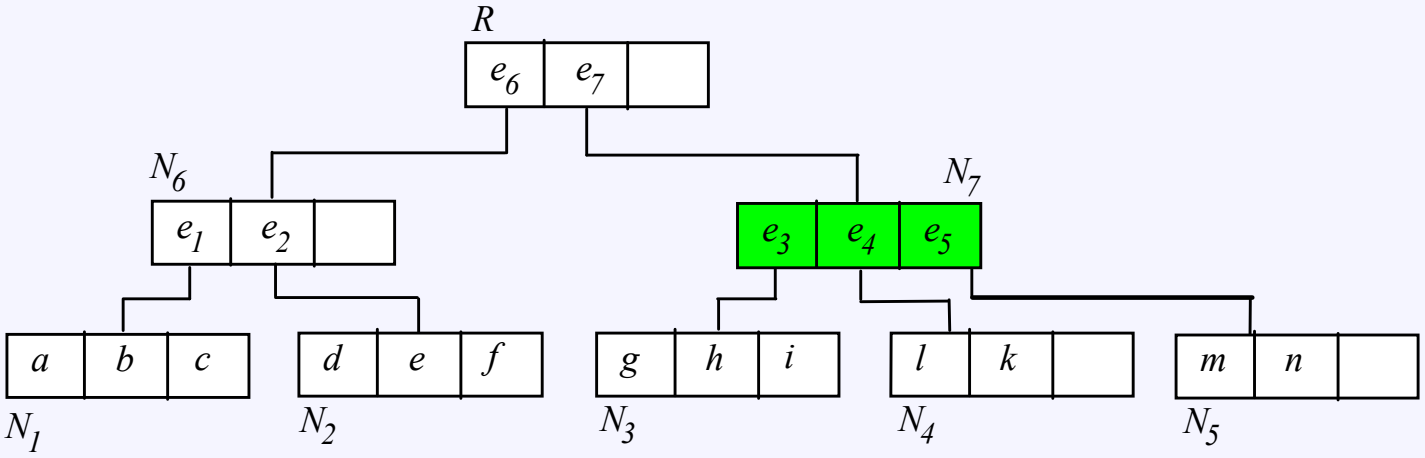


BBS Algorithm - example

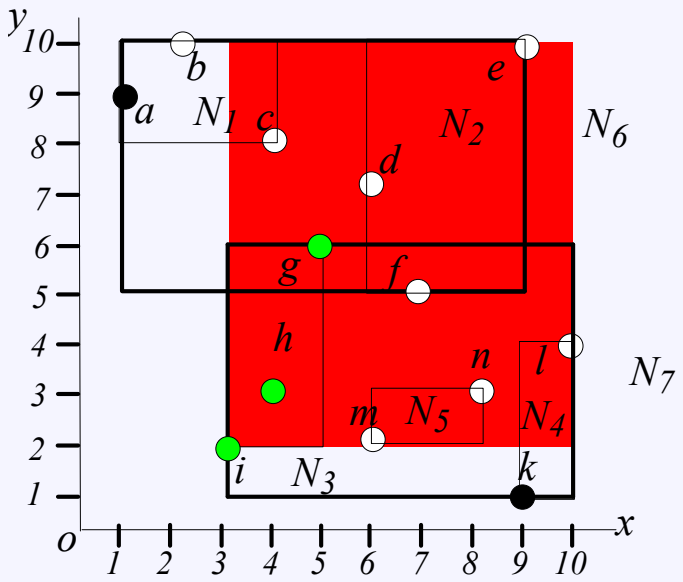


- Process entries in ascending order of their mindists.

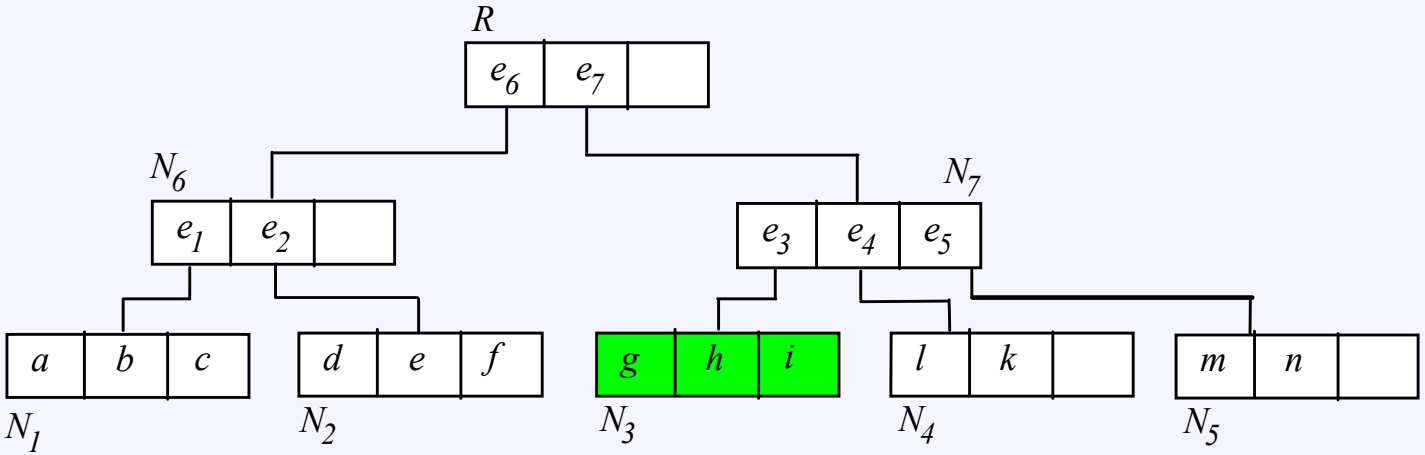
action	heap contents	*S*
access root	<e_7,4><e_6,6>	∅
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	∅



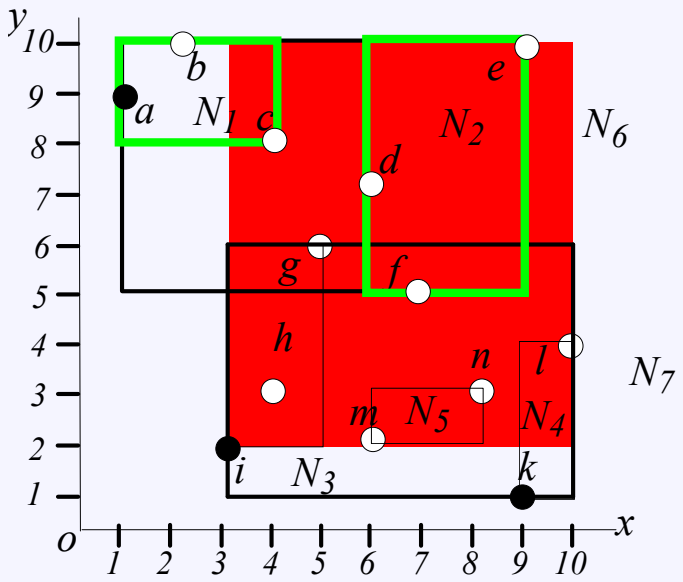
BBS Algorithm - example



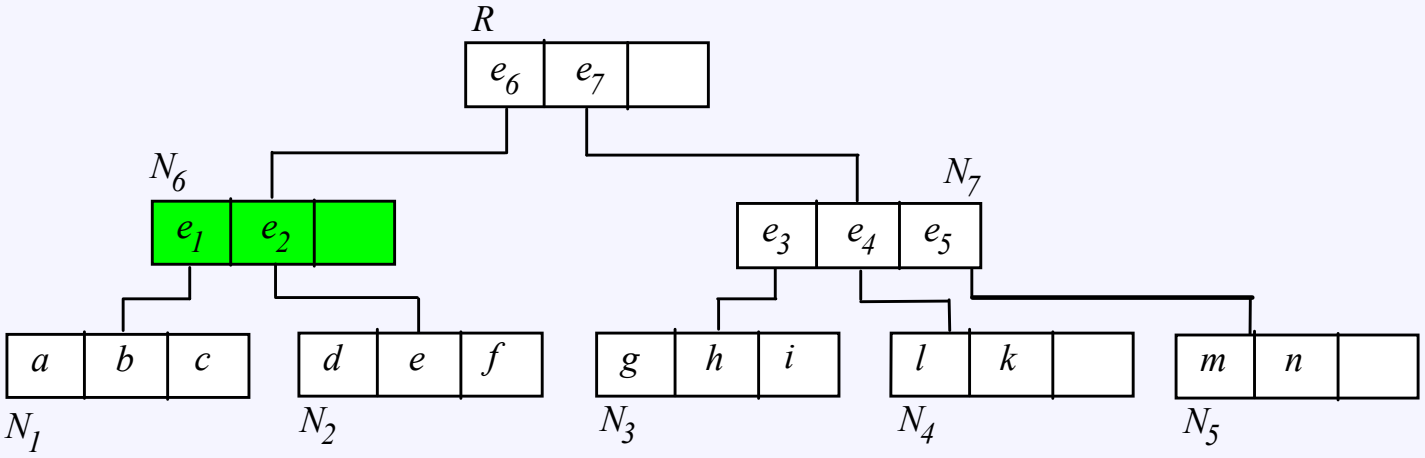
action	heap contents	S
access root	<e_7,4><e_6,6>	∅
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	∅
expand e_3	**<*i*,5>**<e_6,6><e_5,8><e_4,10>	{*i*}



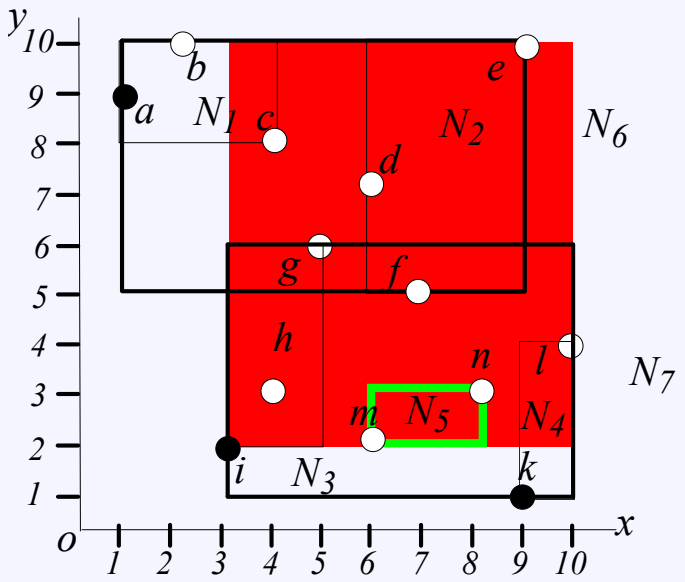
BBS Algorithm - example



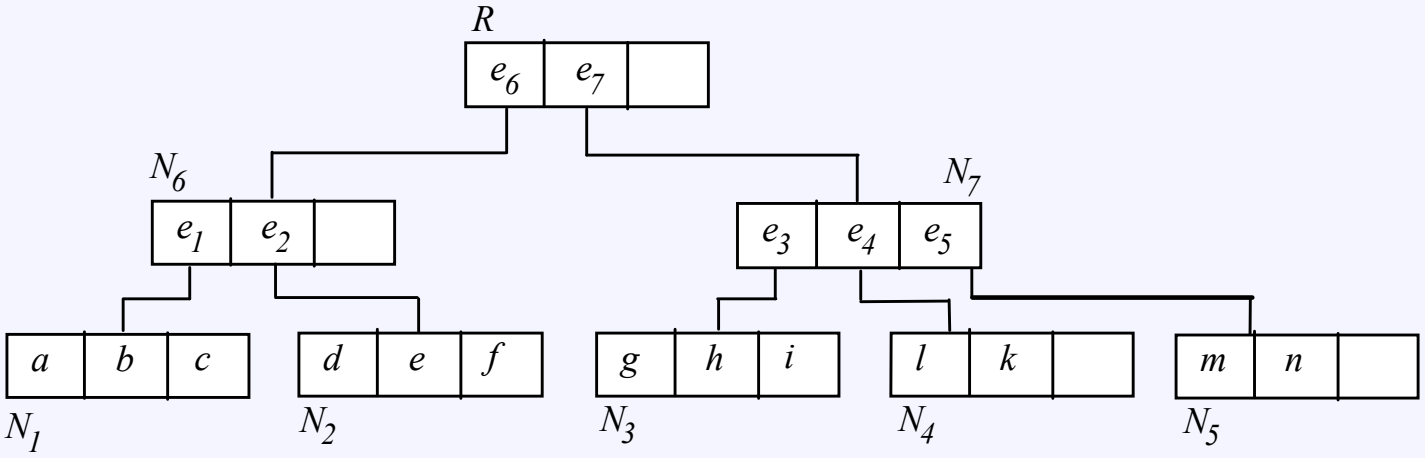
action	heap contents	S
access root	<e_7,4><e_6,6>	∅
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	∅
expand e_3	**<*i*,5>**<e_6,6><e_5,8> <e_4,10>	{*i*}
expand e_6	<e_5,8><e_1,9><e_4,10>	{*i*}



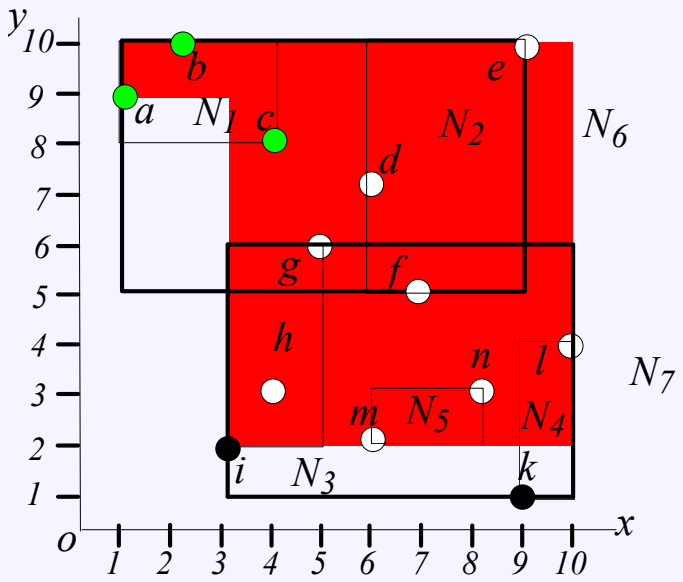
BBS Algorithm - example



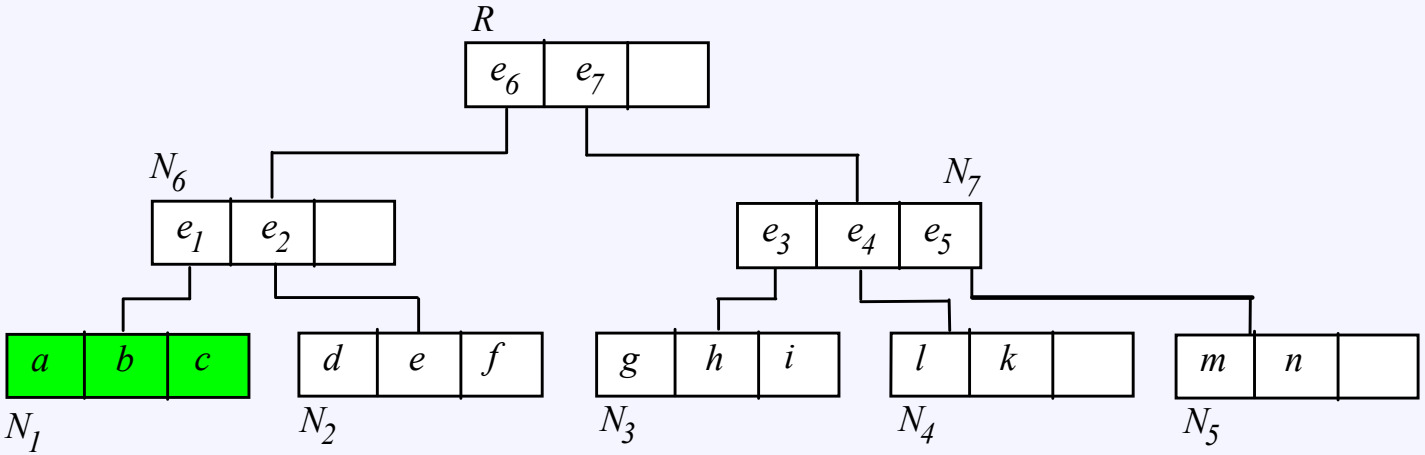
action	heap contents	*S*
access root	<e_7,4><e_6,6>	∅
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	∅
expand e_3	**<*i*,5>**<e_6,6><e_5,8> <e_4,10>	{*i*}
expand e_6	<e_5,8><e_1,9><e_4,10>	{*i*}
remove e_5	<e_1,9><e_4,10>	{*i*}



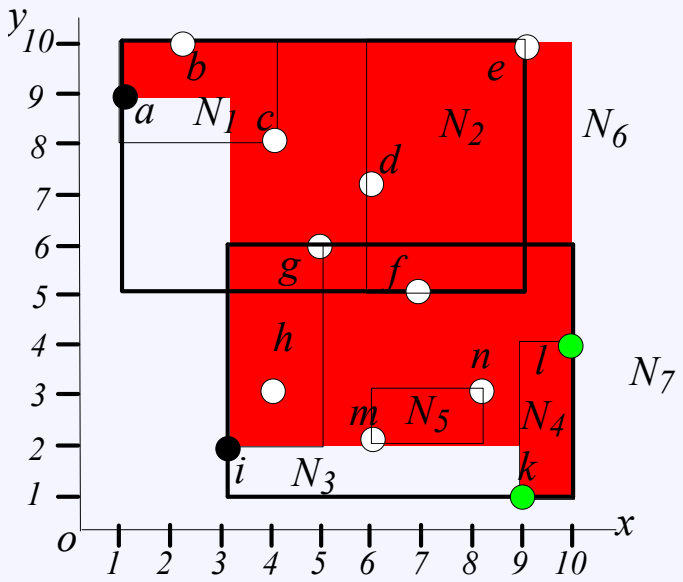
BBS Algorithm - example



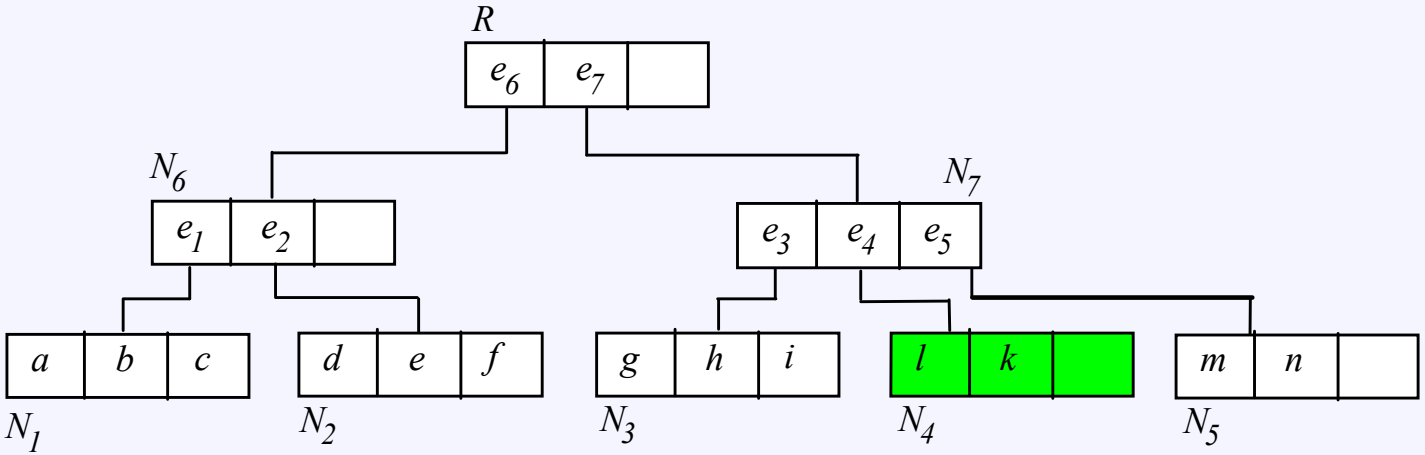
action	heap contents	S
access root	$<e_7,4><e_6,6>$	$\varnothing$
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	$\varnothing$
expand e_3	**<i,5>**$<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$
remove e_5	$<e_1,9><e_4,10>$	$\{i\}$
expand e_1	**<a,10>**$<e_4,10>$	$\{i,a\}$



BBS Algorithm - example

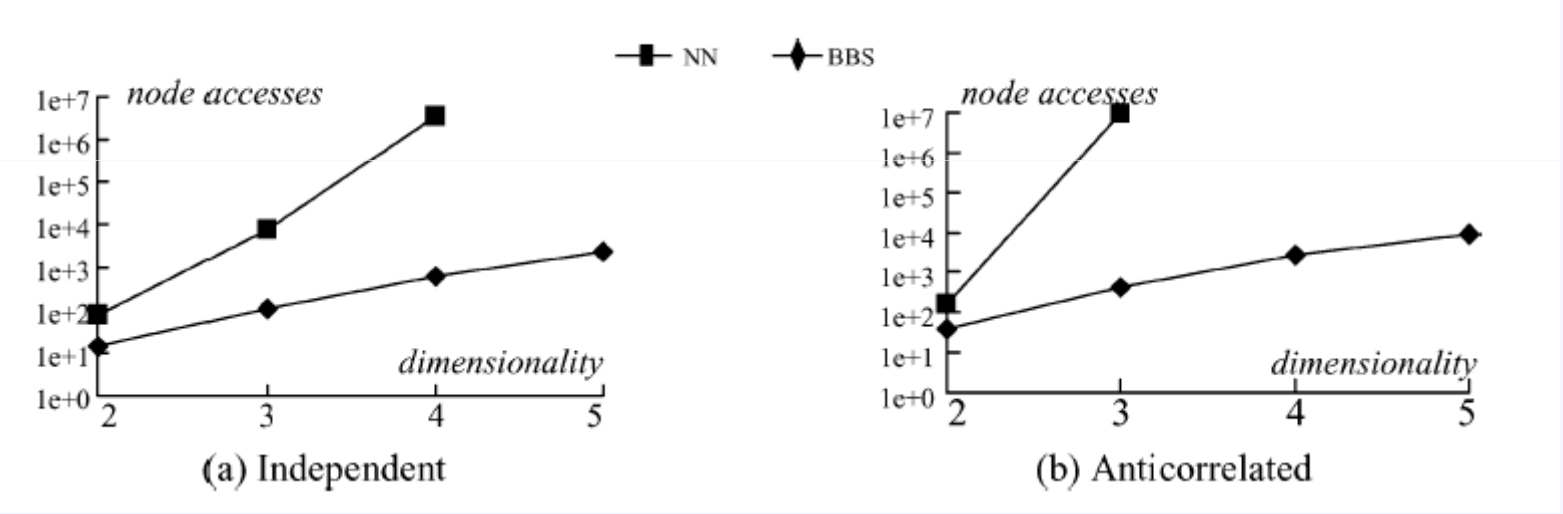


action	heap contents	S
access root	$<e_7,4><e_6,6>$	$\varnothing$
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	$\varnothing$
expand e_3	**<*i*,5>**$<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$
remove e_5	$<e_1,9><e_4,10>$	$\{i\}$
expand e_1	**<*a*,10>**$<e_4,10>$	$\{i,a\}$
expand e_4	**<*k*,10>**	$\{i,a,k\}$



BBS Algorithm - performance

BBS performs better than previously proposed Skyline algorithms, regarding CPU time and I/O time.



Number of R-tree node accesses vs dimensionality

(source: Papadias et al TODS 2005)

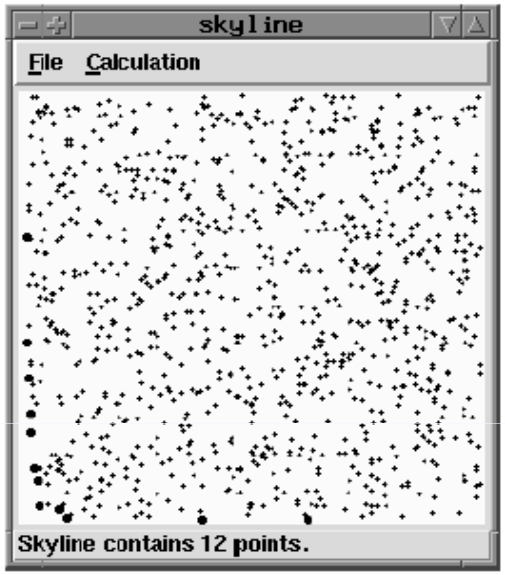
Skyline Computation - advanced topics I

Skylines in subspaces

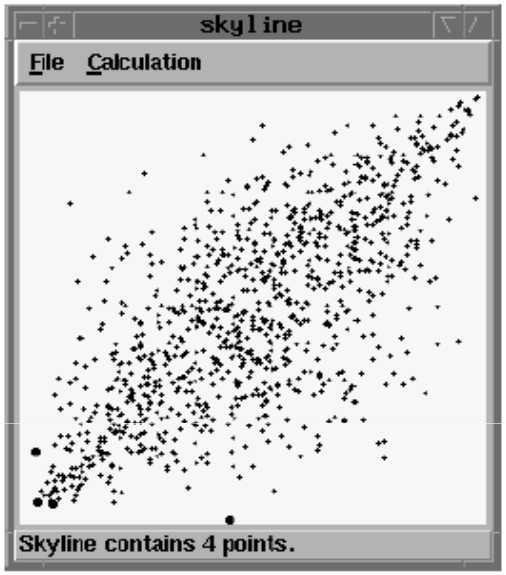
When the number of attributes (dimensions) increases, the number of points contained in the Skyline increases substantially. This happens because the probability that a point dominates another decreases.

Solution: find the Skyline on a subset of the attributes instead of using the whole set of attributes.

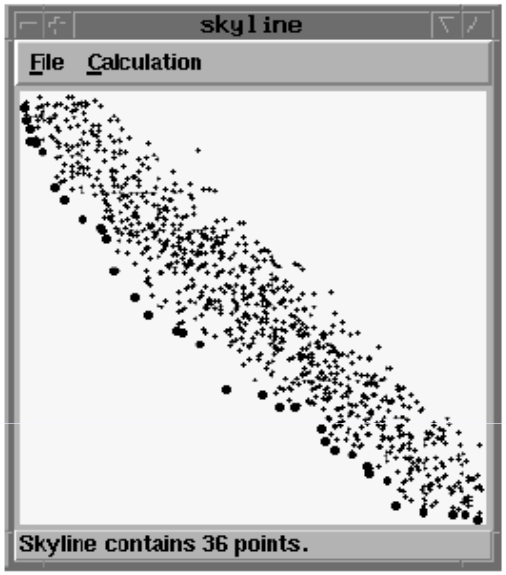
Skyline Computation - advanced topics I



independent



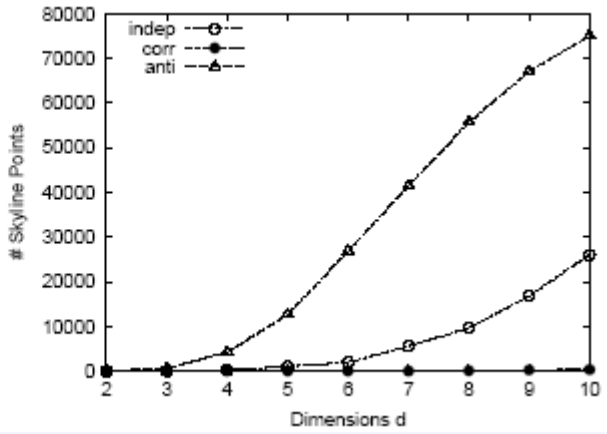
correlated



anticorrelated

100,000 points

d	*corr*	*indep*	*anti*
2	1	12	49
3	3	69	632
4	11	267	4239
5	17	1032	12615
6	21	1986	26843
7	43	5560	41484
8	121	9662	55691
9	243	16847	67101
10	378	26047	75028



(source: Borzonyi et al ICDE 2001)

Skyline Computation - advanced topics II

Distributed Skylines

In several applications, data are distributed across different sites (e.g., web applications, P2P). A number of research contributions deal with efficient processing of Skyline queries in such an environment.

Skyline Computation - advanced topics III

Most important Skyline objects

The number of Skyline points may be large in some cases. The challenge is to rank the Skyline points according to a score. For example, each Skyline point may be ranked according to the number of points it dominates. The highly-ranked points are presented to the user.

Conclusions

- ✓ Preference queries are very important in database systems.
- ✓ Preferences are expressed by minimization or maximization criteria on the attributes (dimensions).
- ✓ Important queries: Top-*k* and Skylines
- ✓ Top-*k* query: requires a scoring function F() and an integer *k* and returns the *k* best objects according to F().
- ✓ Skyline query: requires the preferences regarding minimization or maximization and returns the dominant objects (not dominated by others).

Conclusions

- ✓For Top-k we discussed part of Fagin's work (FA, TA, NRA and CA algorithms). These methods require that attributes are sorted in decreasing score order.
- ✓For Skylines we discussed the Branch-and-Bound Skyline (BBS) algorithm which requires an R-tree index to operate.
- ✓Both Top-k and Skylines offer a convenient way to select the best objects from a database, when multiple criteria are considered.

Conclusions

Current Trends

- Find efficient indexing schemes to speed-up the processing of Top-*k* and Skyline queries.
- Algorithms to process approximate answers (less accurate but faster).
- Preference queries in distributed environments.

Bibliography

1. S. Borzsonyi, D. Kossmann, K. Stocker. "The Skyline Operator". *Proceedings of the International Conference on Data Engineering*, pp.421-430, 2001.

2. R. Fagin, Amnon Lotem, Moni Naor. "Optimal Aggregation Algorithms for Middleware". *J. Comput. Syst. Sci.* 66(4), pp. 614-656, 2003.

3. R. Fagin. "Combining Fuzzy Information from Multiple Systems". *Proceedings of the 15th ACM Symposium on principles of database systems*, pp. 216-226, Montreal Canada, 1996.

4. R. Fagin. "Fuzzy Queries in Multimedia Database Systems". *Proceedings of the 17th ACM Symposium on principles of database systems*, pp. 1-10, Seattle USA, 1998.

5. A. Guttman. "R-trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of the ACM SIGMOD Conference*, 1984.

6. D. Papadias, Y. Tao, G. Fu, B. Seeger. "Progressive Skyline Computation in Database Systems", *ACM Transactions on Database Systems*, Vol.30, No.1, pp.41-82, 2005.