Department of Informatics
Aristotle University of Thessaloniki

Skyline Query Processing

A. Papadopoulos
Associate Professor

Outline of Presentation

- Introduction
- Skyline computation
  - Introduction to R-trees
  - Algorithm BBS (branch-and-bound skyline)
  - Advanced topics
- Conclusions
- Bibliography

Introduction

Skyline queries have been proposed as an alternative to satisfy user preferences.

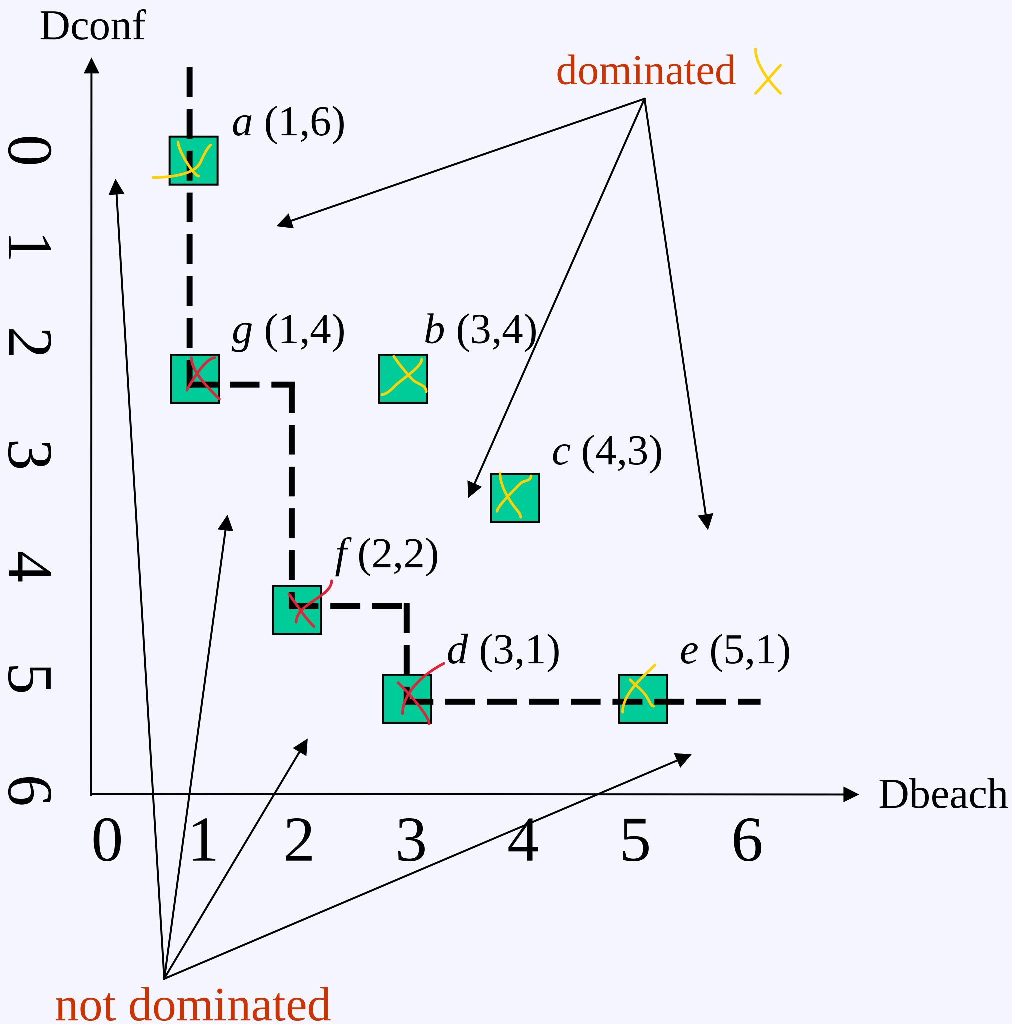
The Skyline query

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

Introduction

- 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



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 google

id	score	i	score
a	0.9	*b*	0.8
c	0.7	*d*	0.7
b	0.6	*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

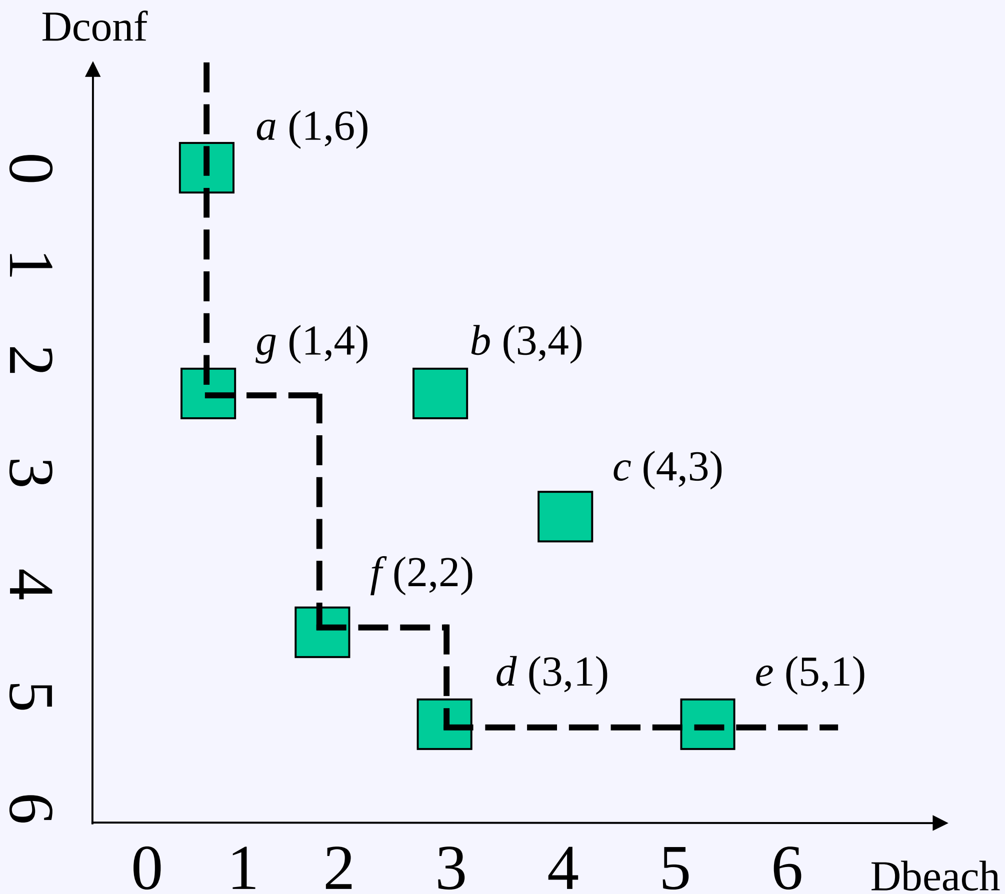
Skyline processing

- 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.

Skyline Computation



Skyline objects: *g*, *f*, *d*

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

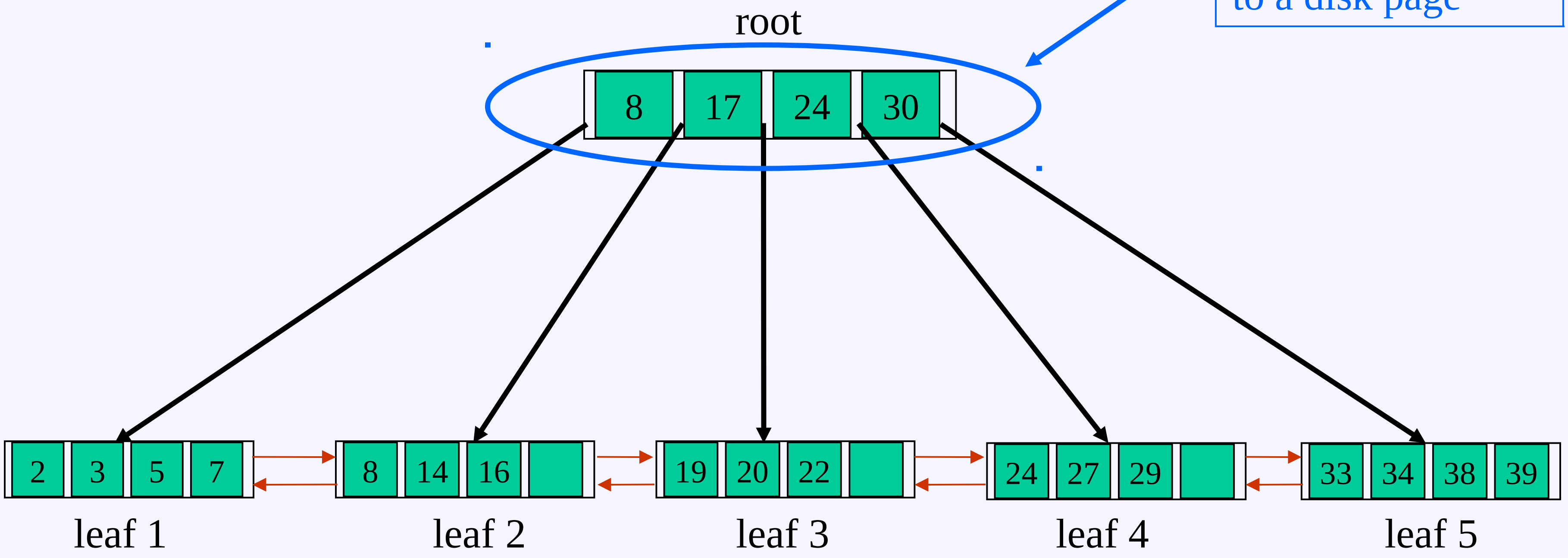
has received more than 2,900 citations
(source google scholar)

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:

Each node corresponds to a disk page



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*.

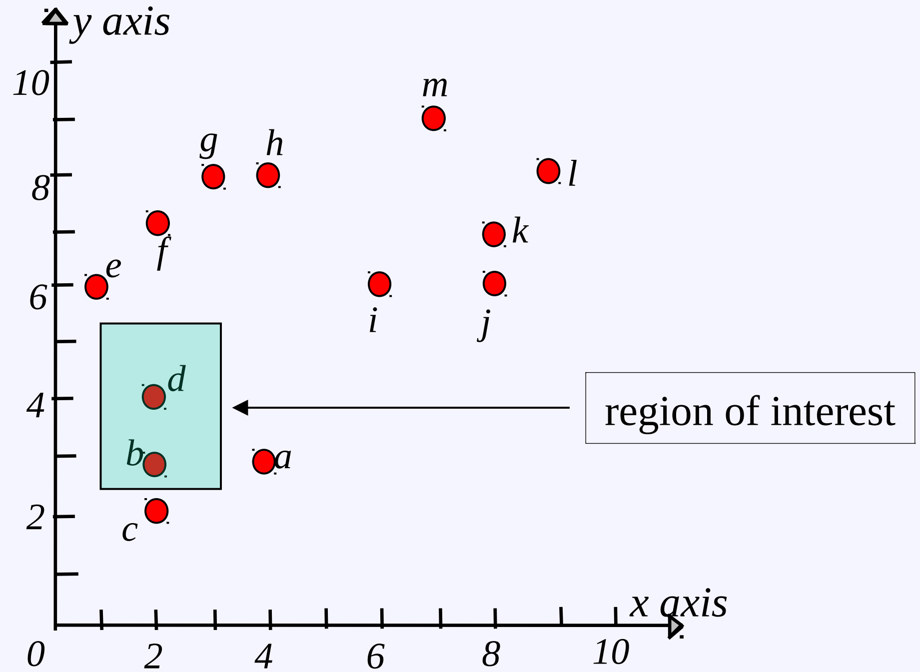
Introduction to R-trees

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

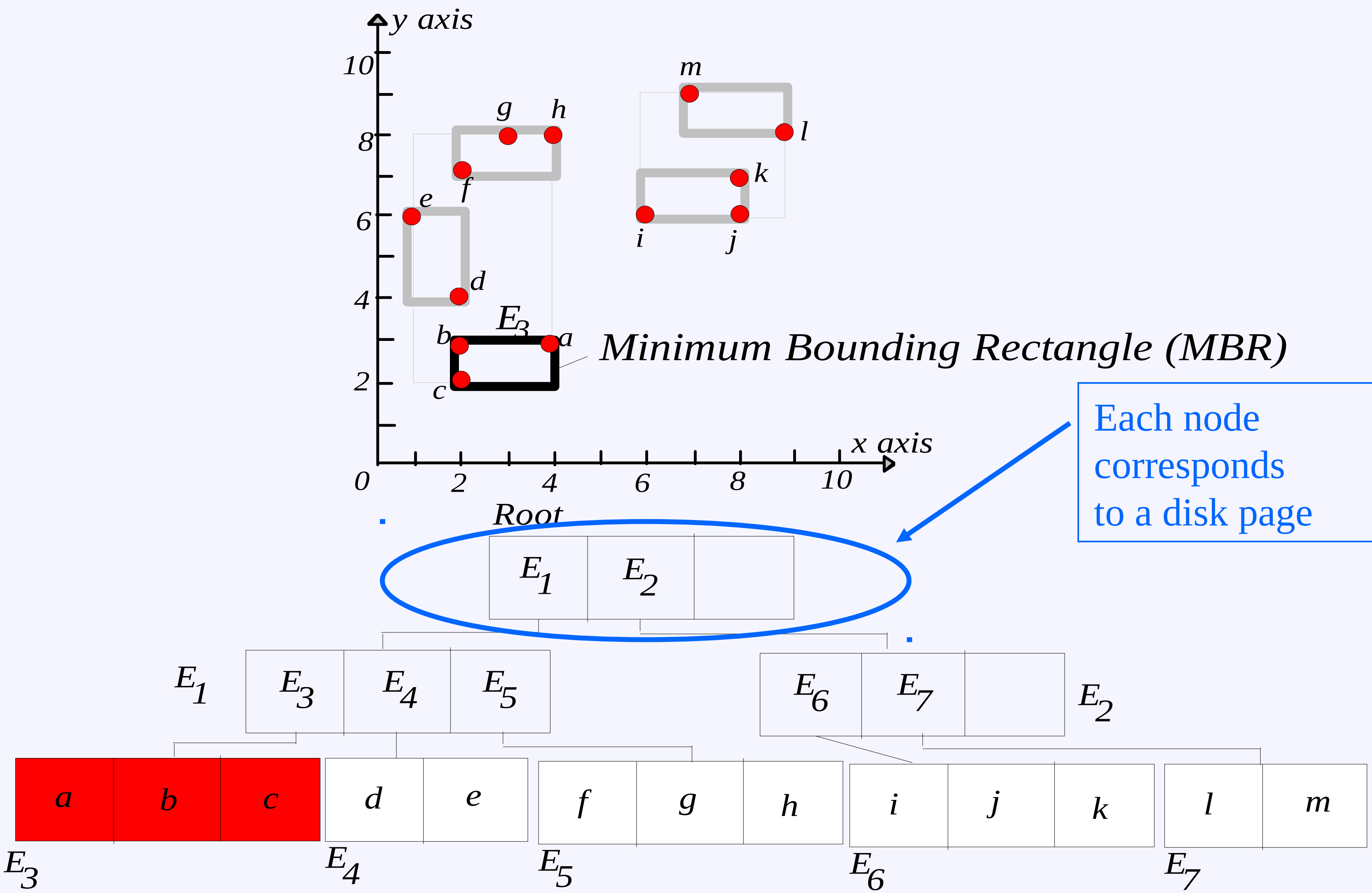
Example:
13 points in
2 dimensions



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

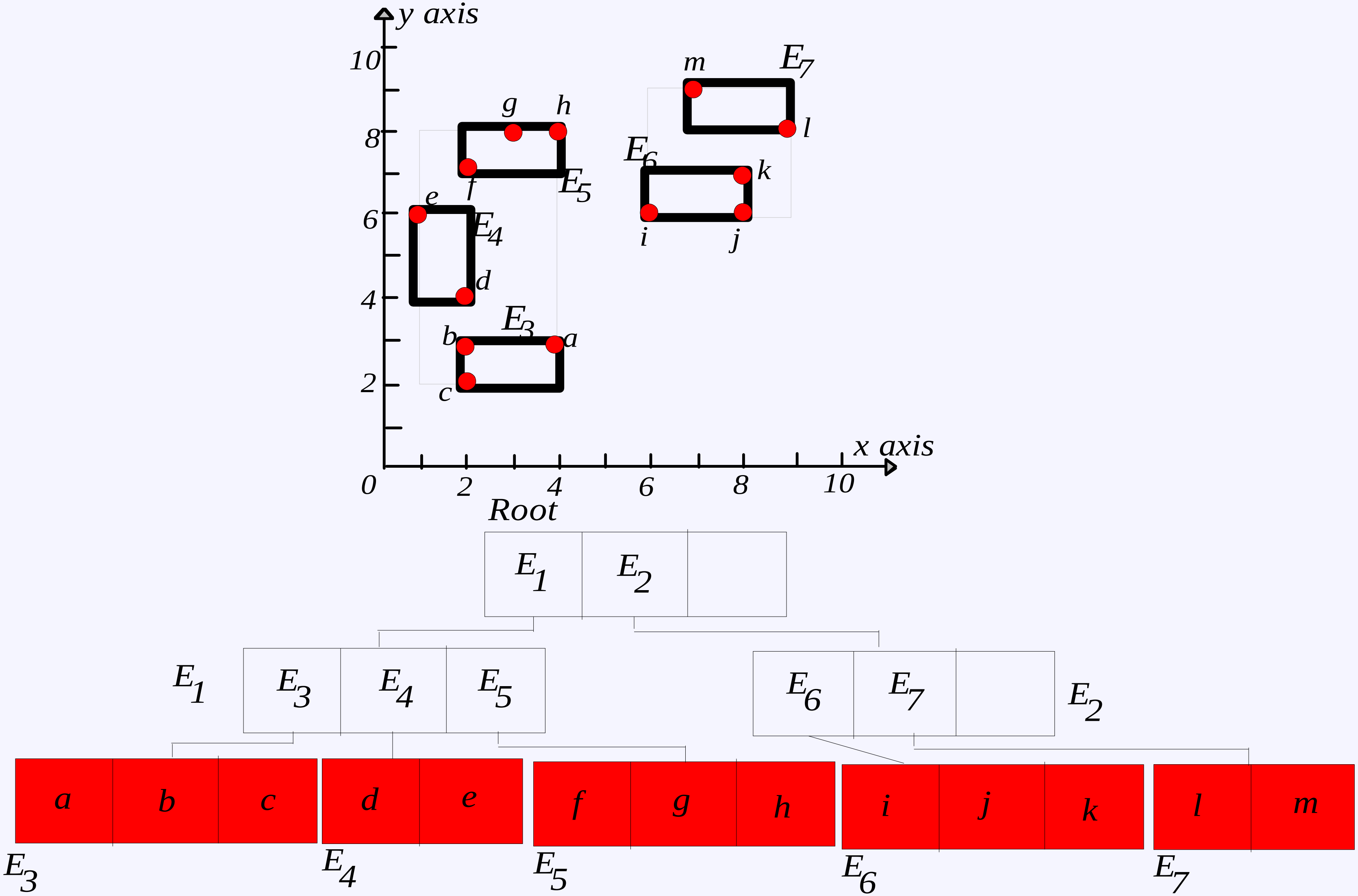
No index: scan through all objects. NOT EFFICIENT!

Introduction to R-trees – structure

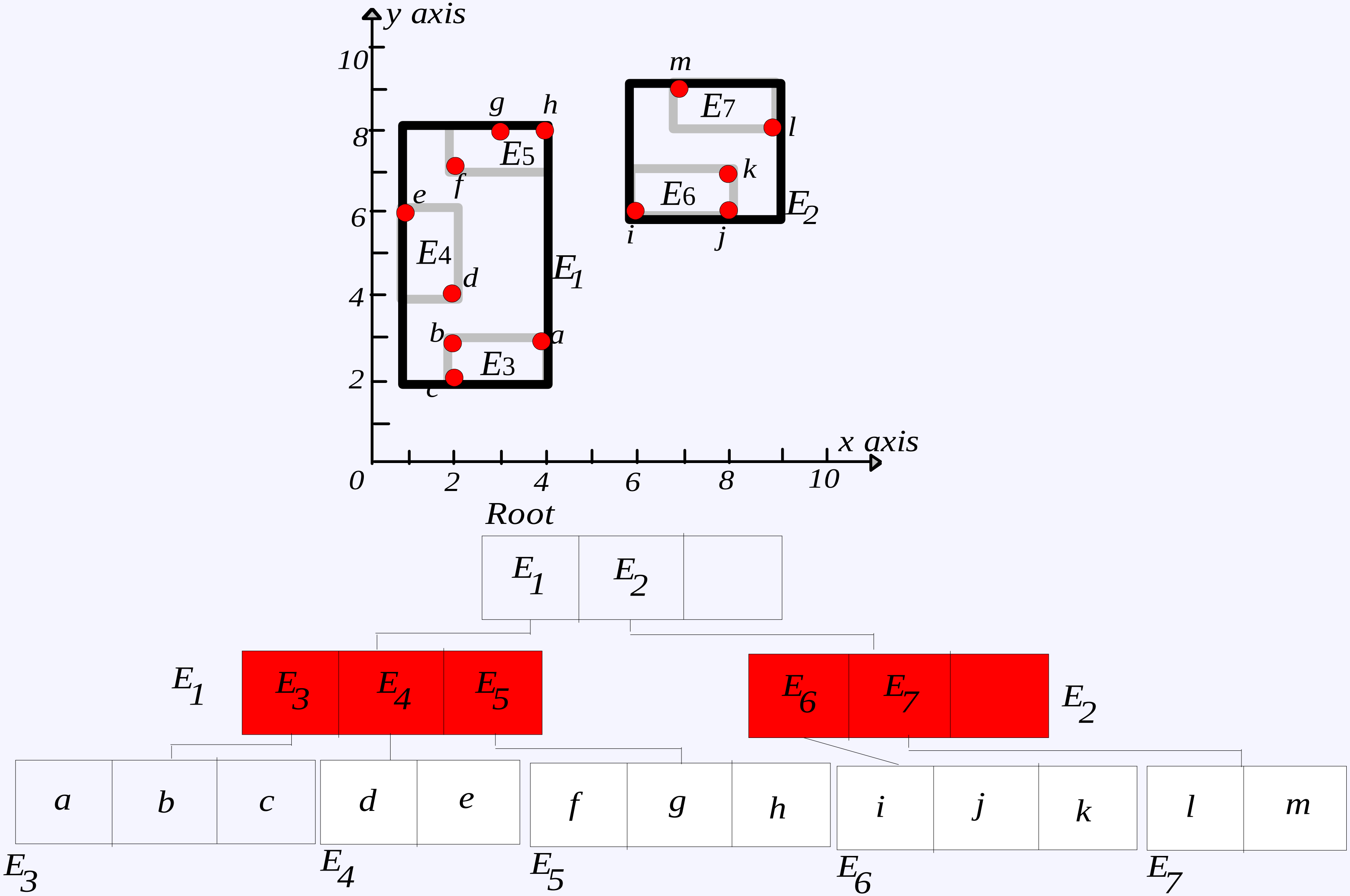


Each node corresponds to a disk page

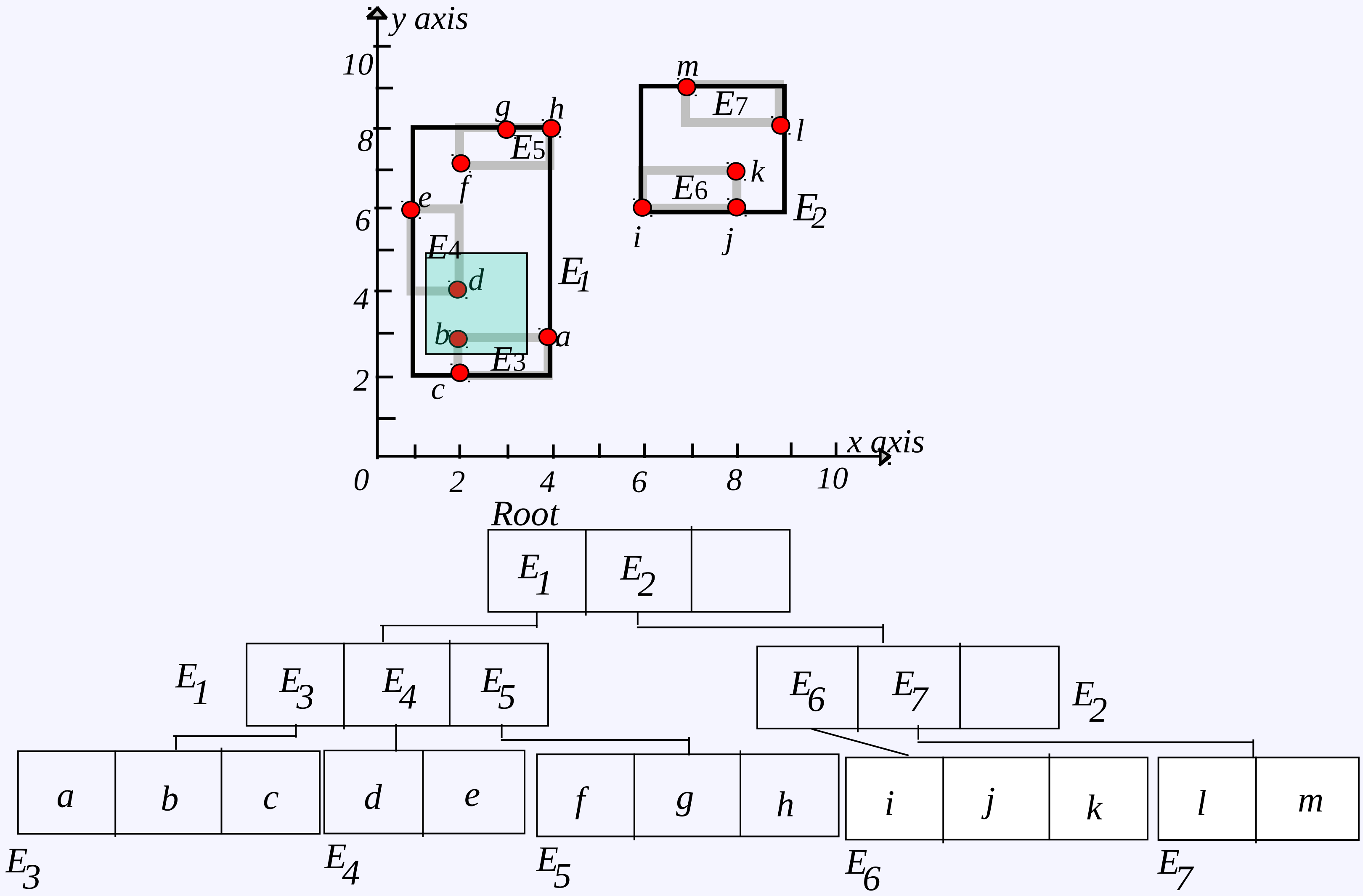
Introduction to R-trees – structure



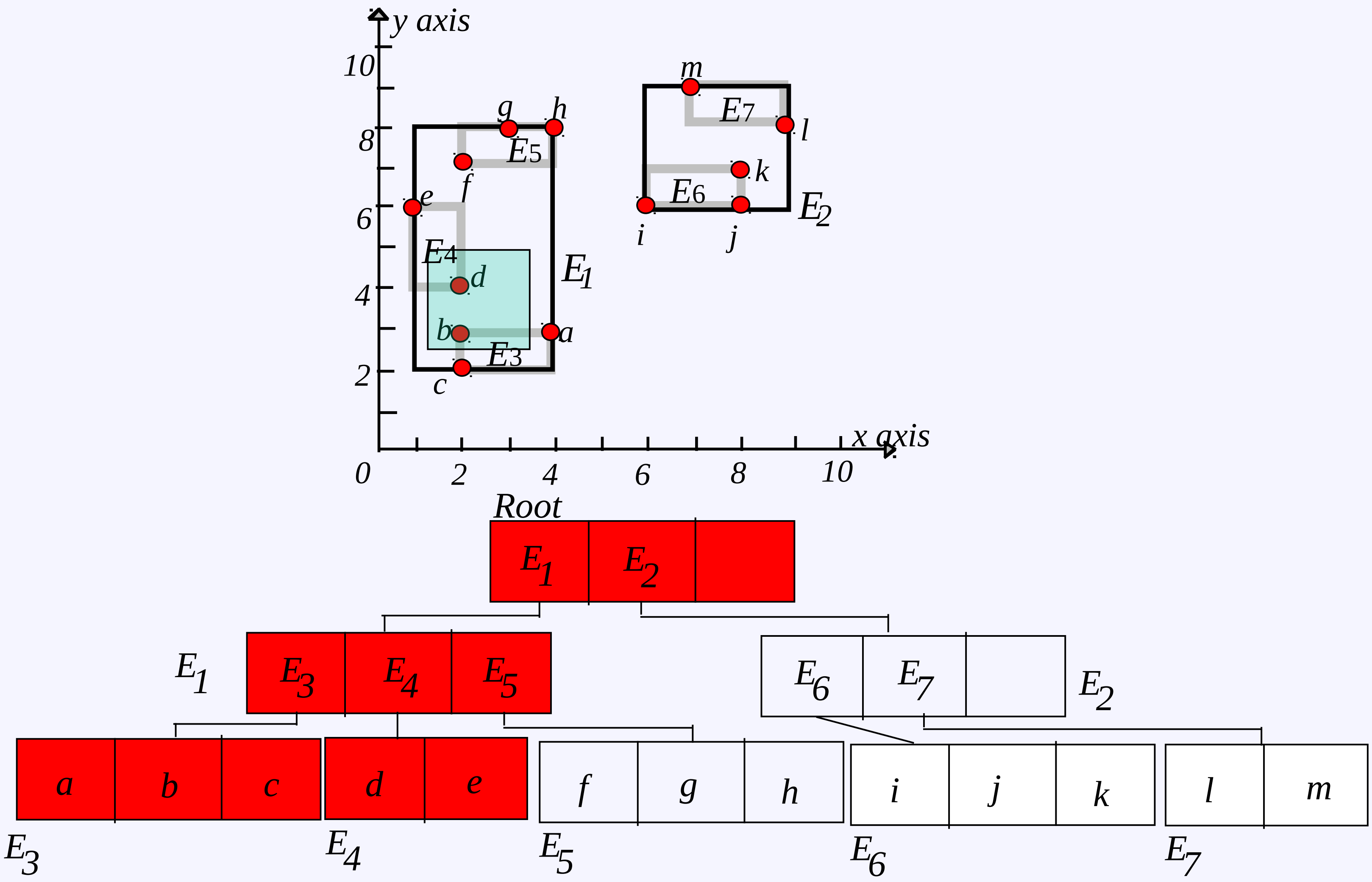
Introduction to R-trees – structure



Introduction to R-trees – range query



Introduction to R-trees – range query



BBS Algorithm – Basic Properties

Any Branch-and-Bound method requires two decisions:

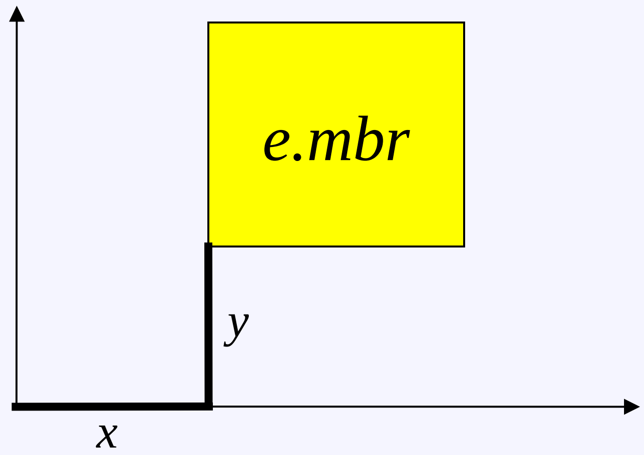
1. How to branch: which part of the space needs to be investigated next?

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:

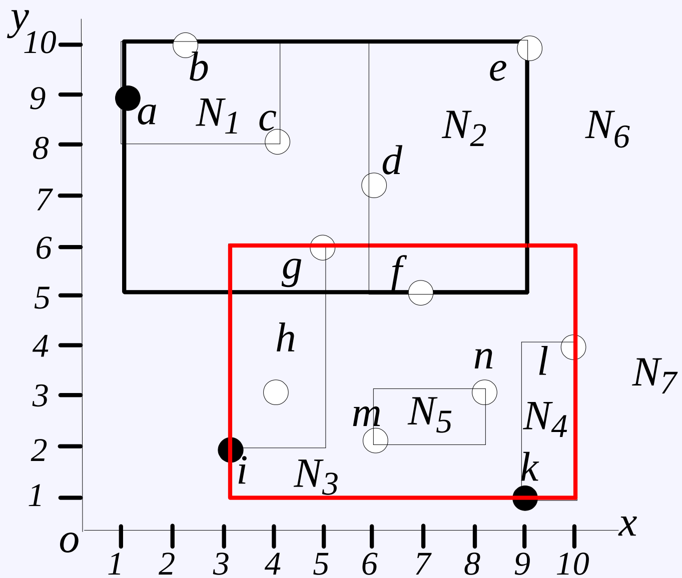


mindist(*e.mbr*) = x + y

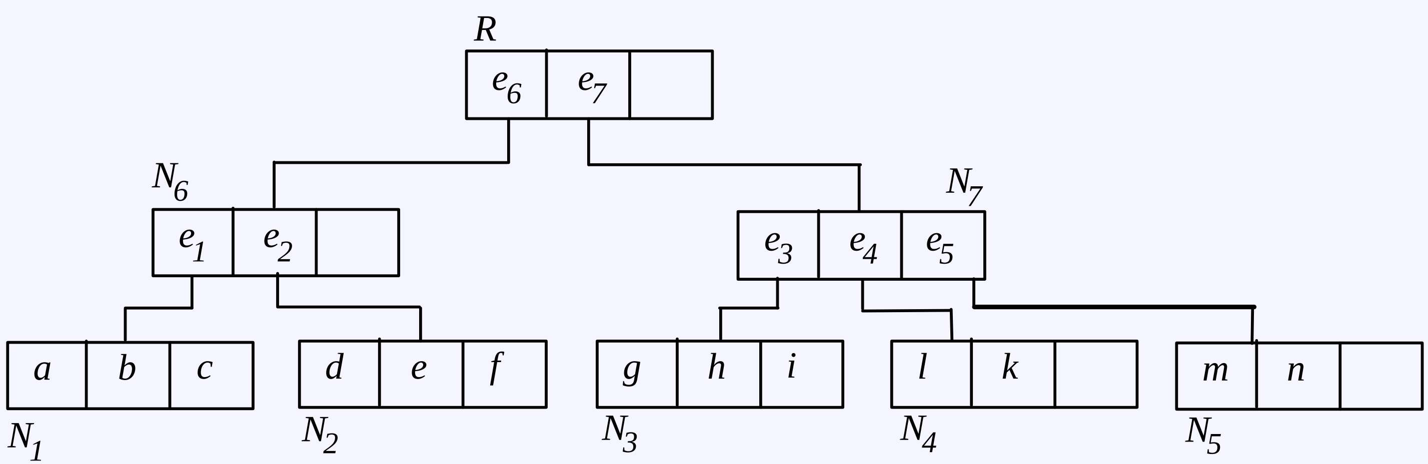
BBS Algorithm – basic properties

- 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*.

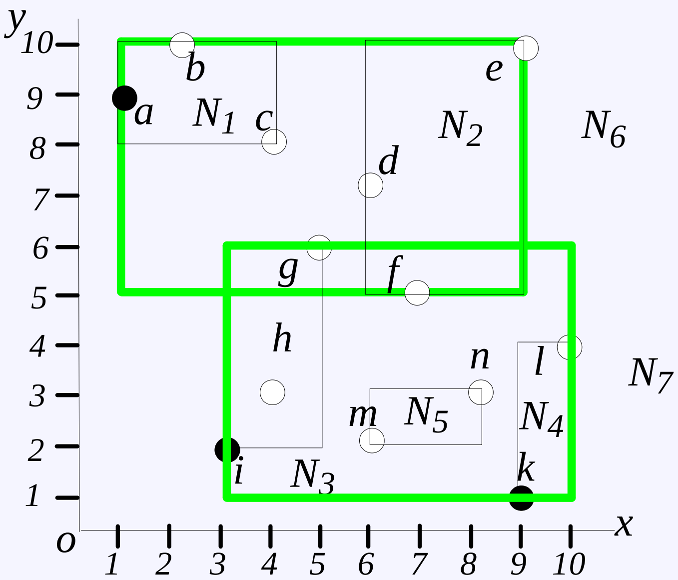
BBS Algorithm - example



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

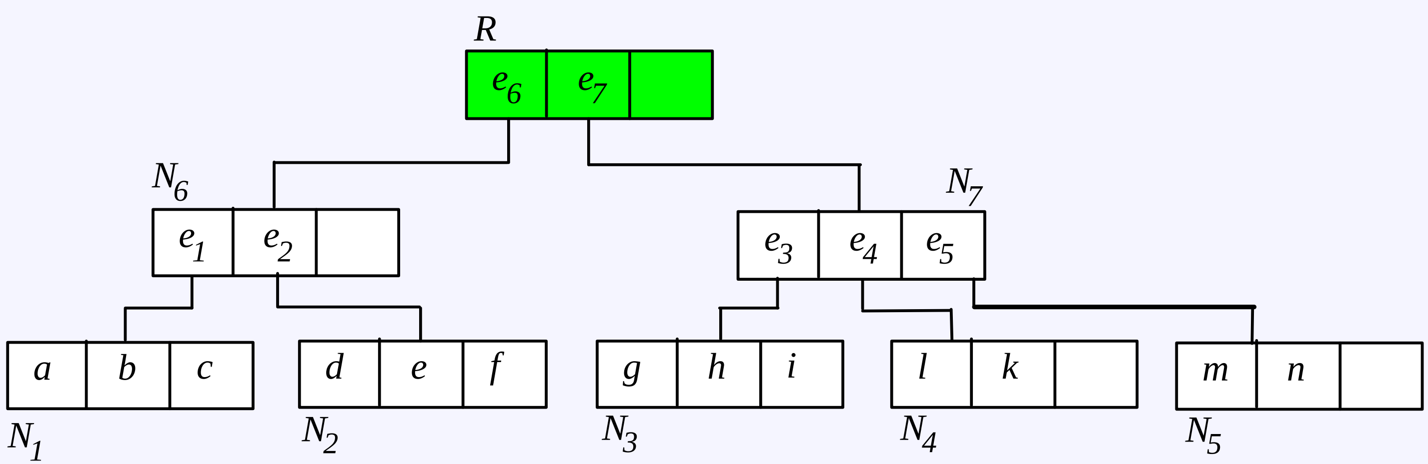


BBS Algorithm - example

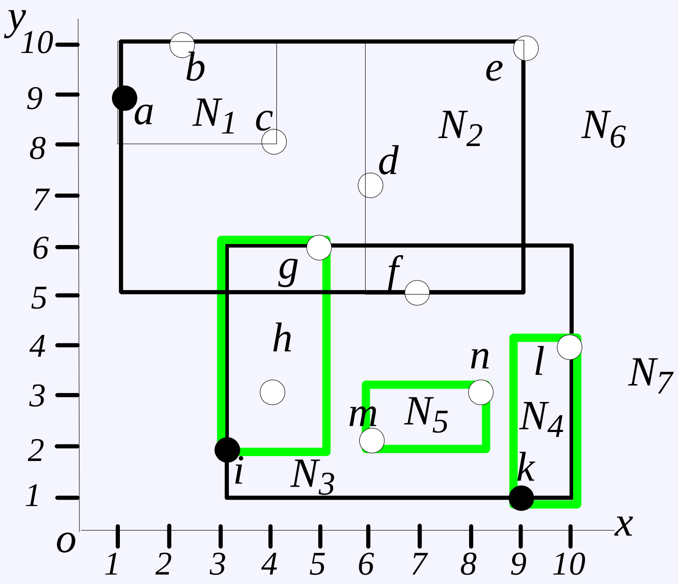


- Each heap entry keeps the mindist of the MBR.

action	**heap contents**	***S***
access root	<e_7,4><e_6,6>	∅

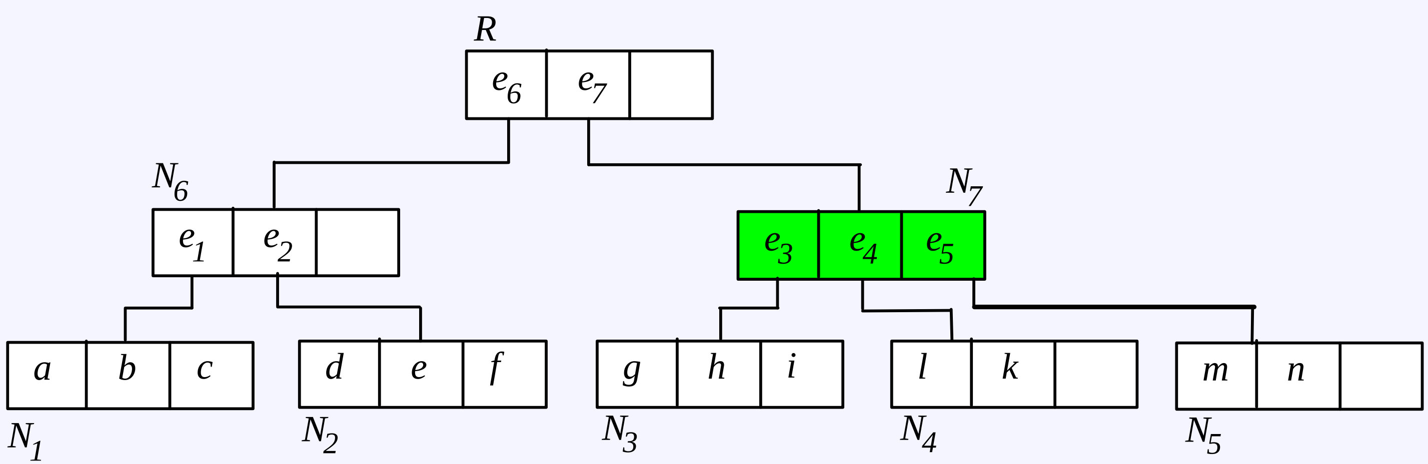


BBS Algorithm - example

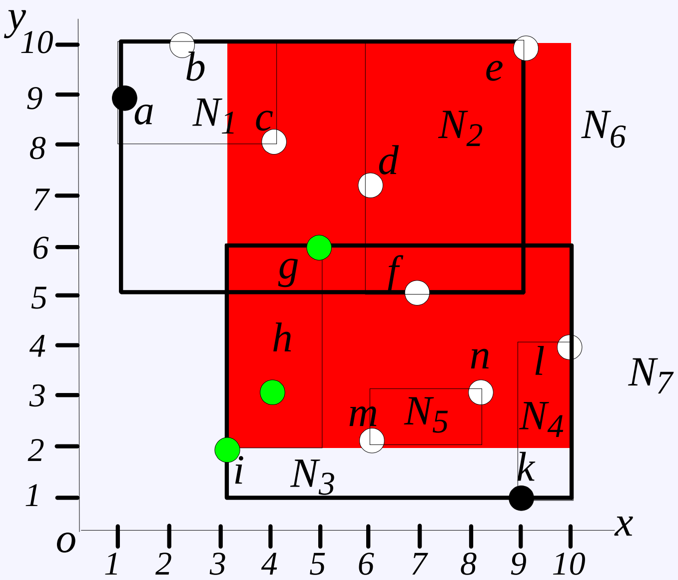


- Process entries in ascending order of their mindists.

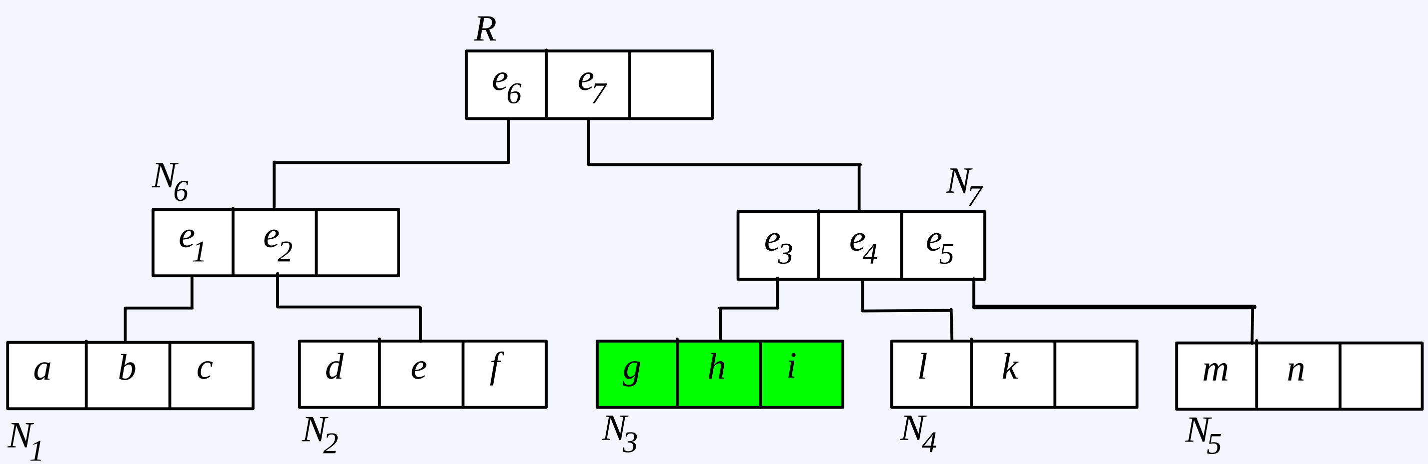
action	**heap contents**	***S***
access root	$<e_7,4><e_6,6>$	$\varnothing$
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	$\varnothing$



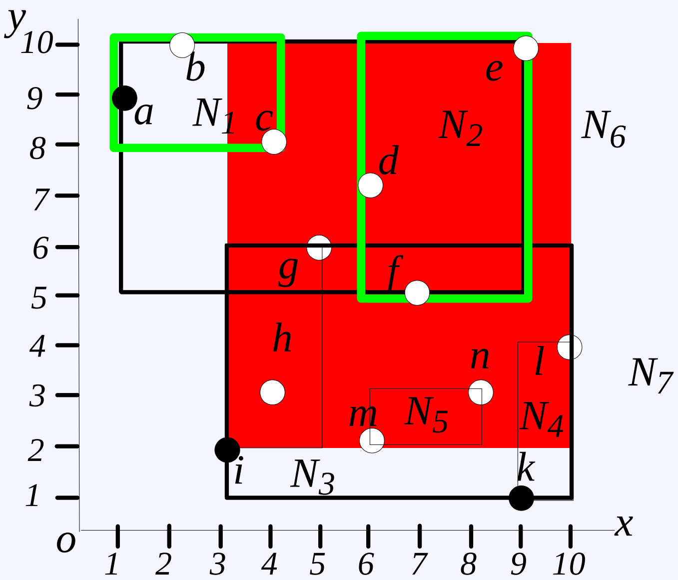
BBS Algorithm - example



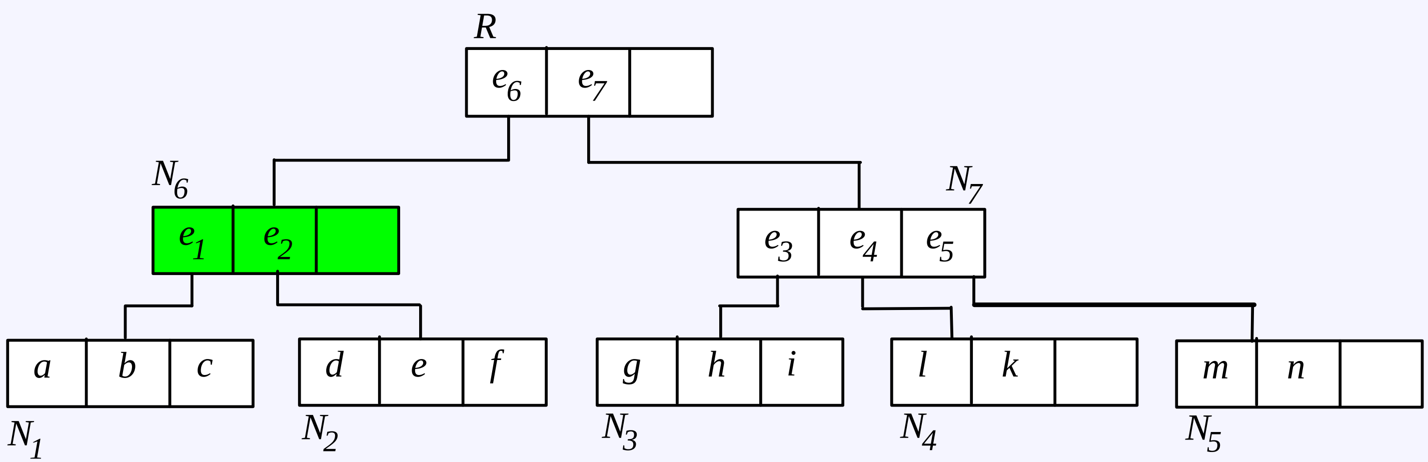
action	heap contents	*S*
access root	$<e_7,4><e_6,6>$	∅
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	∅
expand e_3	$\mathbf{<i,5>}<e_6,6><e_5,8><e_4,10>$	$\{i\}$



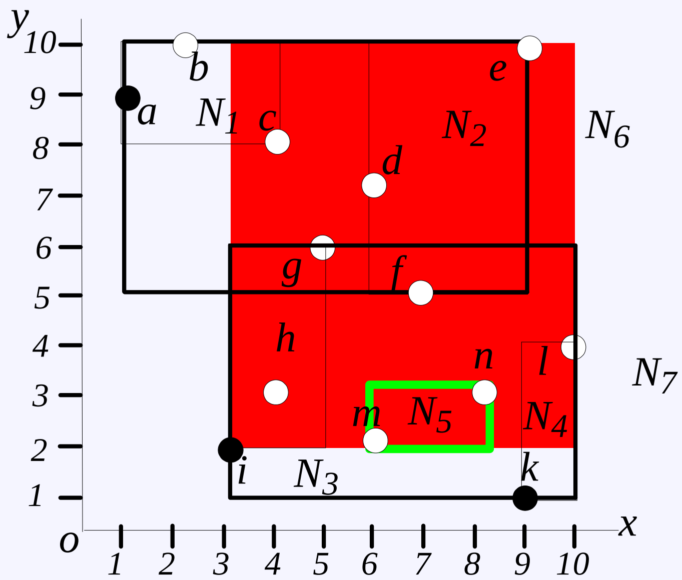
BBS Algorithm - example



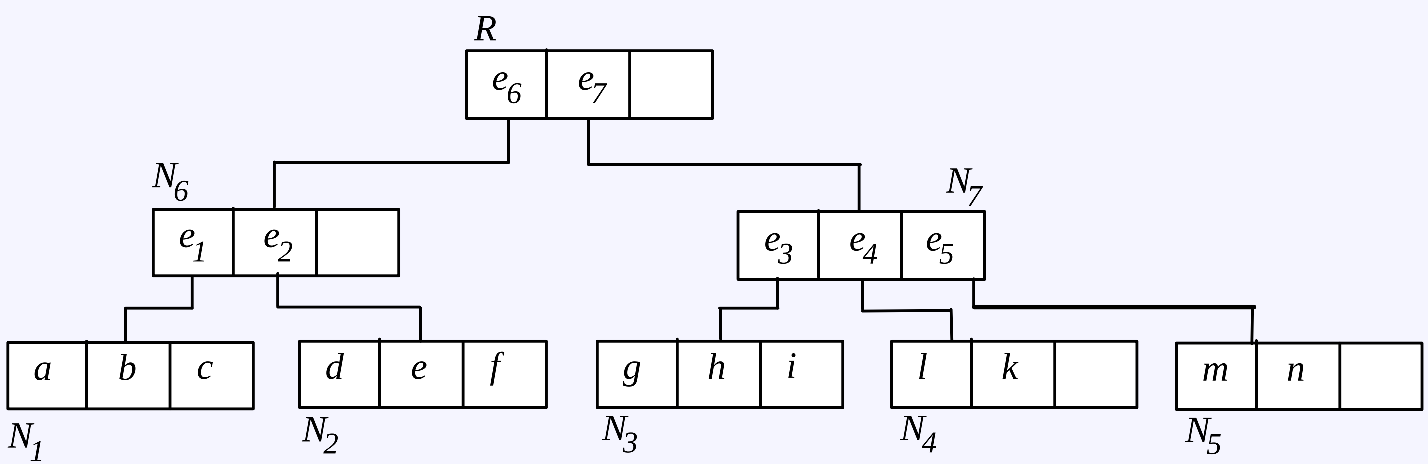
action	heap contents	S
access root	$<e_7,4><e_6,6>$	$\emptyset$
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	$\emptyset$
expand e_3	$\mathbf{<i,5>}<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$



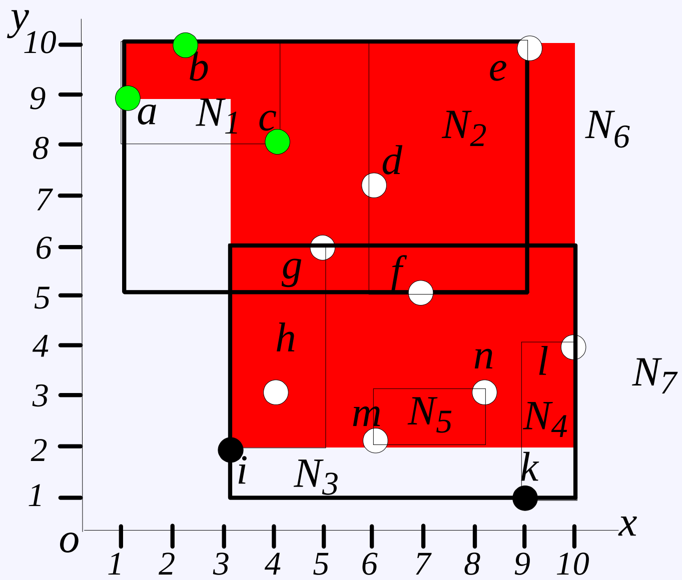
BBS Algorithm - example



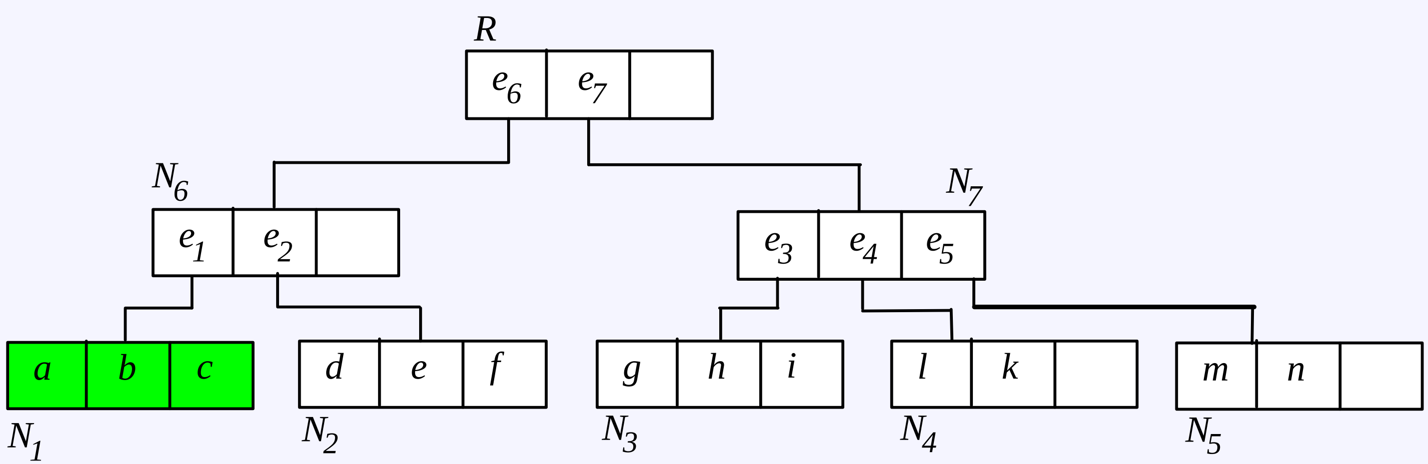
action	heap contents	S
access root	$<e_7,4><e_6,6>$	$\varnothing$
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	$\varnothing$
expand e_3	$\mathbf{<i,5>}<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$
remove e_5	$<e_1,9><e_4,10>$	$\{i\}$



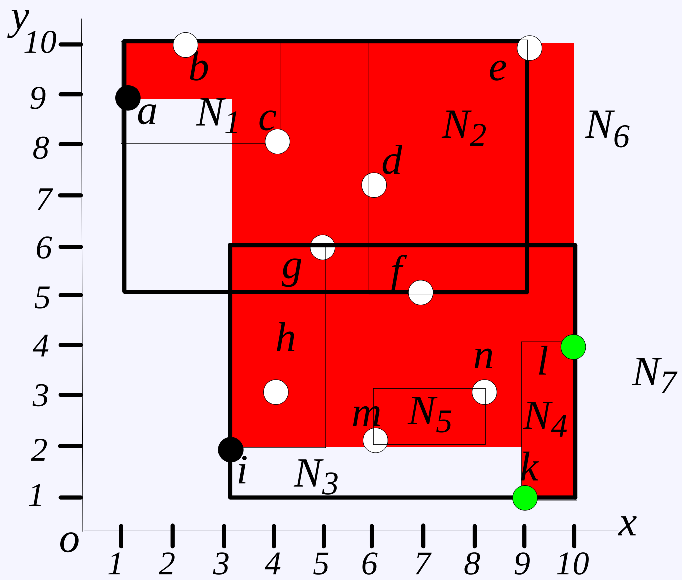
BBS Algorithm - example



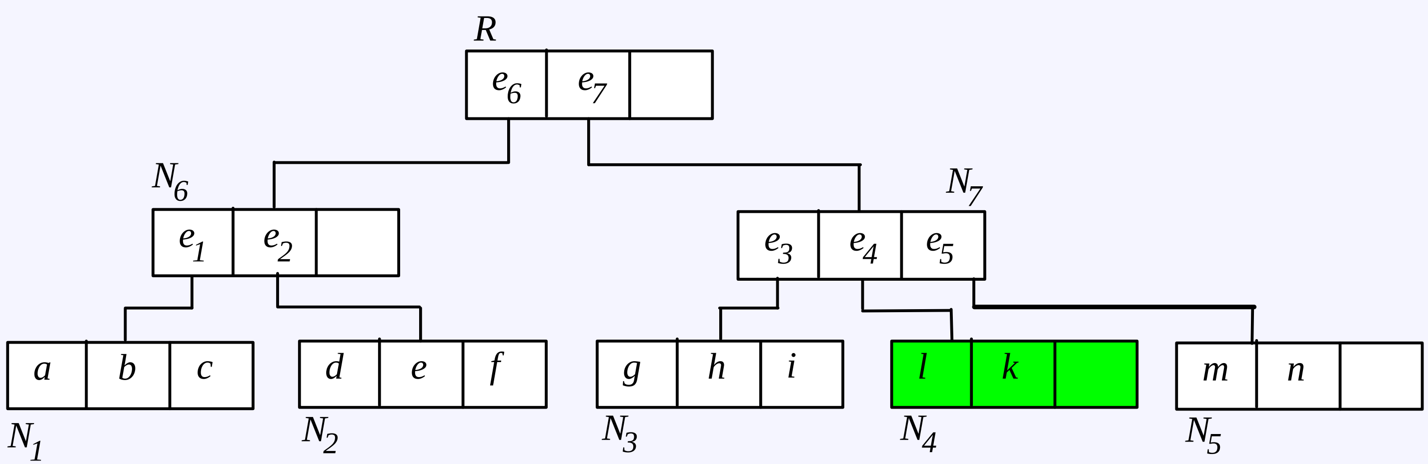
action	heap contents	S
access root	$<e_7,4><e_6,6>$	$\emptyset$
expand e_7	$<e_3,5><e_6,6><e_5,8><e_4,10>$	$\emptyset$
expand e_3	**$<i,5>$**$<e_6,6><e_5,8><e_4,10>$	$\{i\}$
expand e_6	$<e_5,8><e_1,9><e_4,10>$	$\{i\}$
remove e_5	$<e_1,9><e_4,10>$	$\{i\}$
expand e_1	**$<a,10>$**$<e_4,10>$	$\{i,a\}$



BBS Algorithm - example

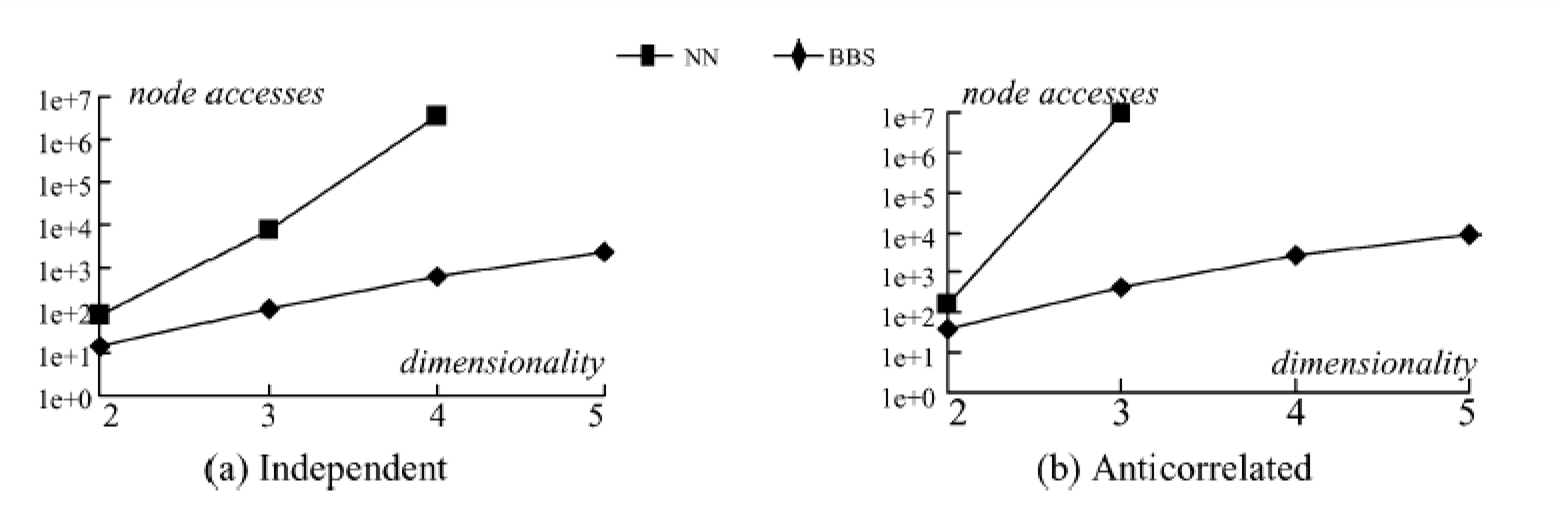


action	heap contents	S
access root	<e_7,4><e_6,6>	$\oslash$
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	$\oslash$
expand e_3	**<*i*,5>**<e_6,6><e_5,8> <e_4,10>	{*i*}
expand e_6	<e_5,8><e_1,9><e_4,10>	{*i*}
remove e_5	<e_1,9><e_4,10>	{*i*}
expand e_1	**<*a*,10>**<e_4,10>	{*i*,*a*}
expand e_4	**<*k*,10>**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

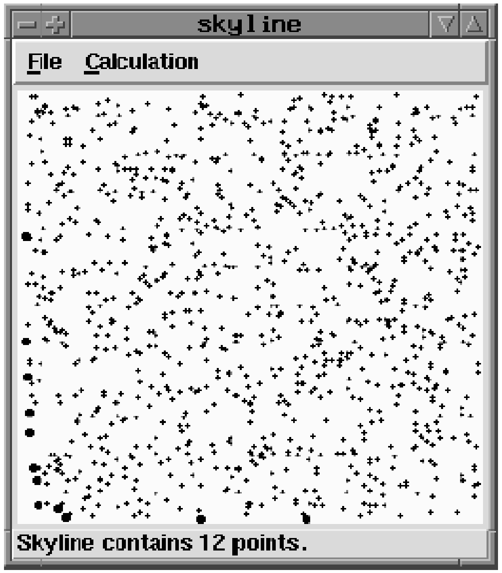
Skyline Computation - advanced topics I

Skylines in subspaces

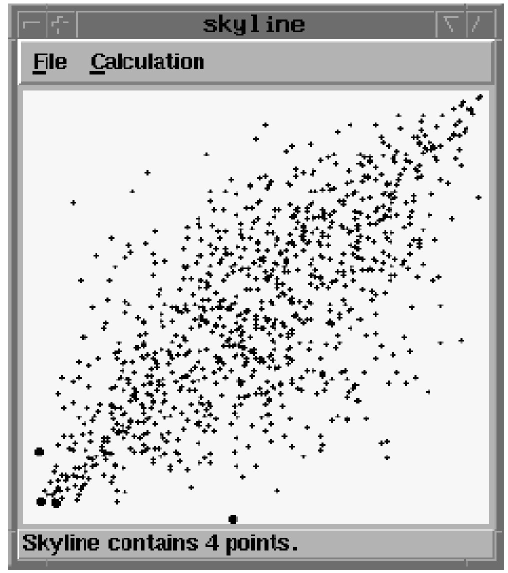
When the number of attributes (dimensions) increases, the number of points contained in the Skyline increases substantially. This happens because the probability that a point dominates another decreases.

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

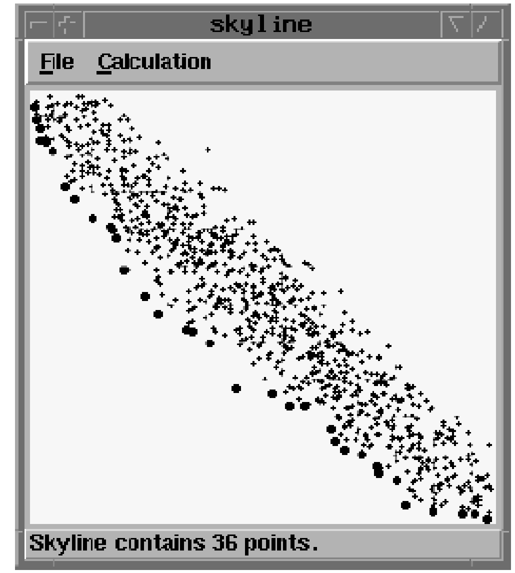
Skyline Computation - advanced topics I



independent



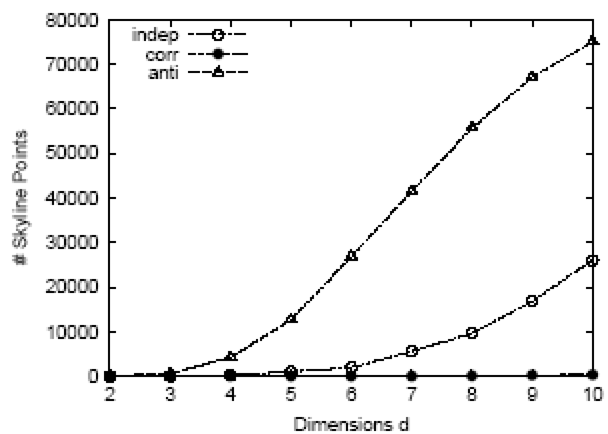
correlated



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.

Bibliography

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

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

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.