The hB-pi* Tree: An optimized Comprehensive Access Method for Frequent-update Multi-dimensional Point Data

- Panfeng Zhou
  - Sybase, Dublin, CA, USA
- Betty Salzberg,
  - Northeastern University, Boston, MA, USA

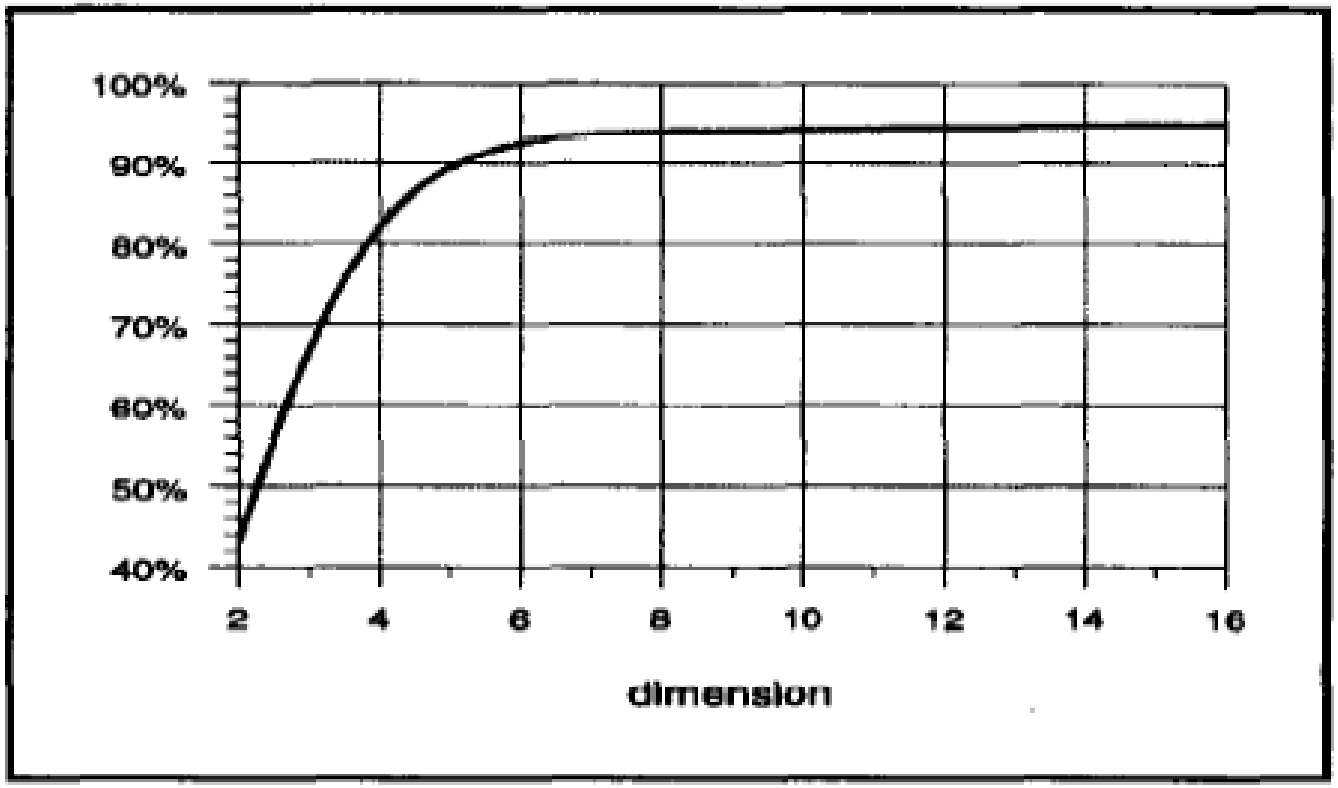
Overview

1. Why not R-tree
2. Why not other kd-tree
3. The motivation of the hB-pi* tree
4. The hB-pi* tree

1. Why not R-tree

1. Overlaps among index pages increase rapidly with growing dimensionality
2. Fanout of index pages decreases rapidly with growing dimensionality
3. Poor concurrency performance

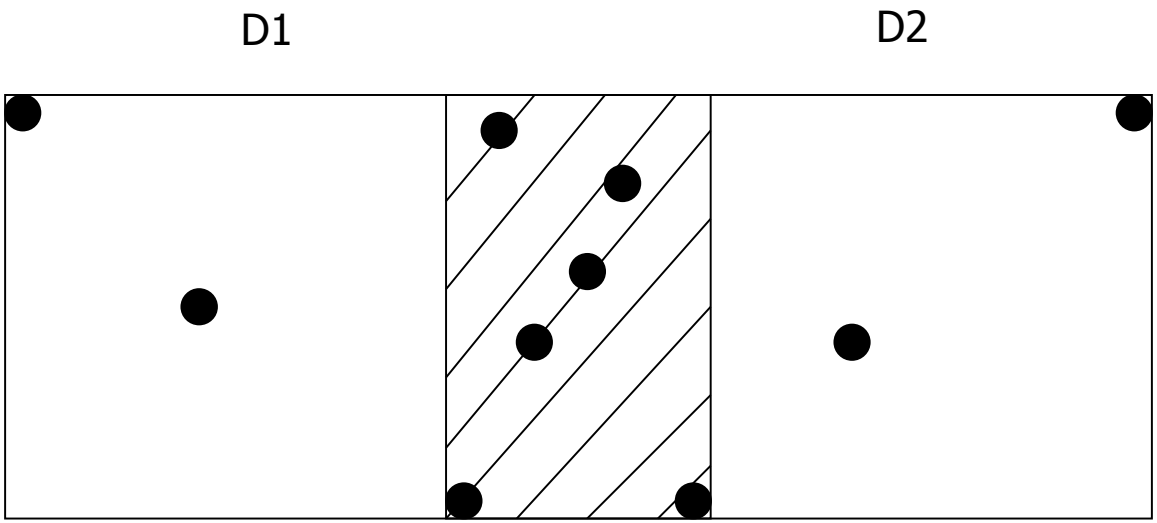
1.1 Overlaps in the R-tree



Overlap rate

Citation: The X-tree : An Index Structure for High-Dimensional Data, VLDB 1996

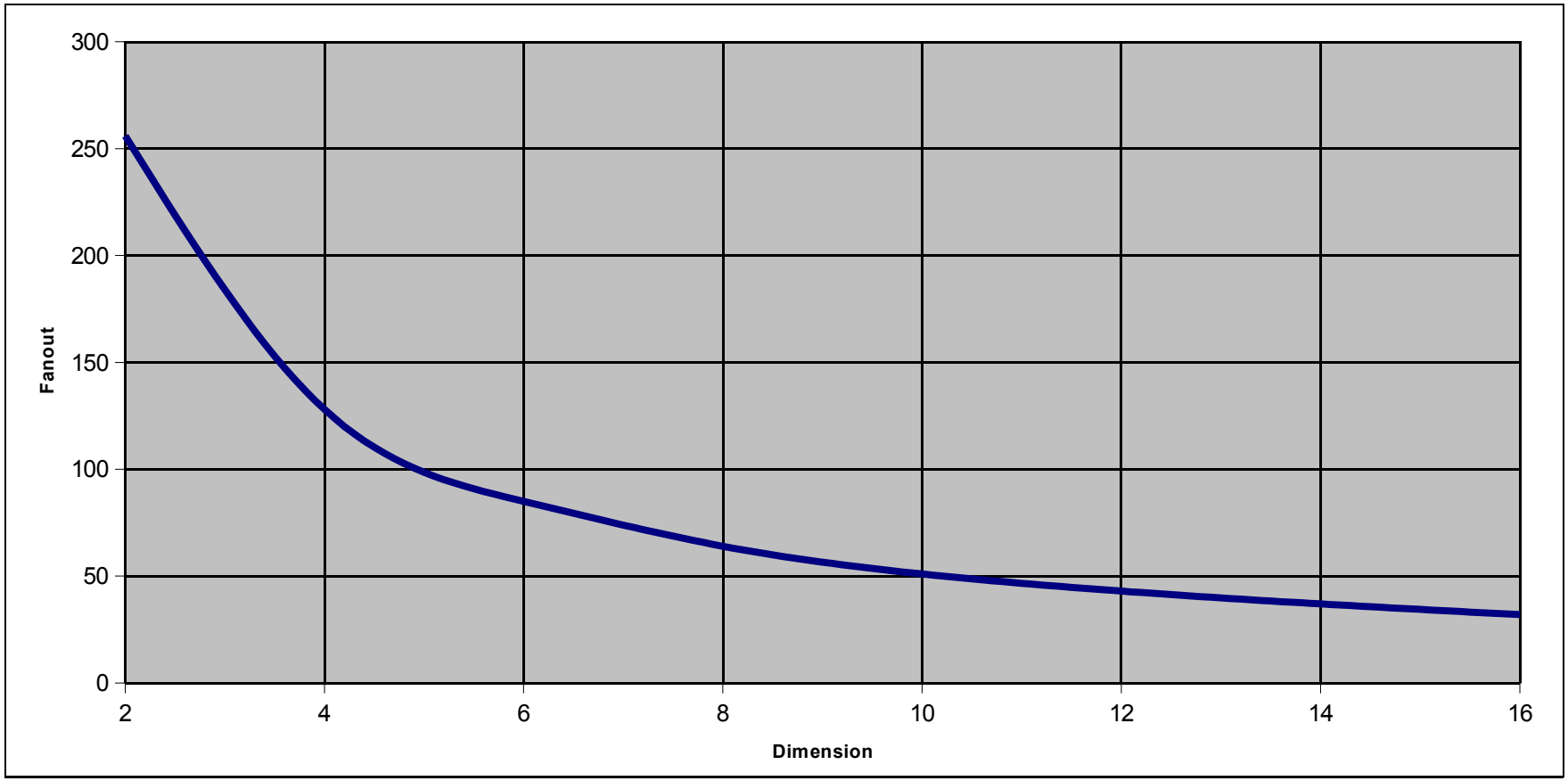
1.1 Definition of overlap



$$\text{Overlap} = \frac{\text{\# data in overlapping area}}{\text{\# data in whole area}}$$

Comment: in the figure above, the overlap is 60%

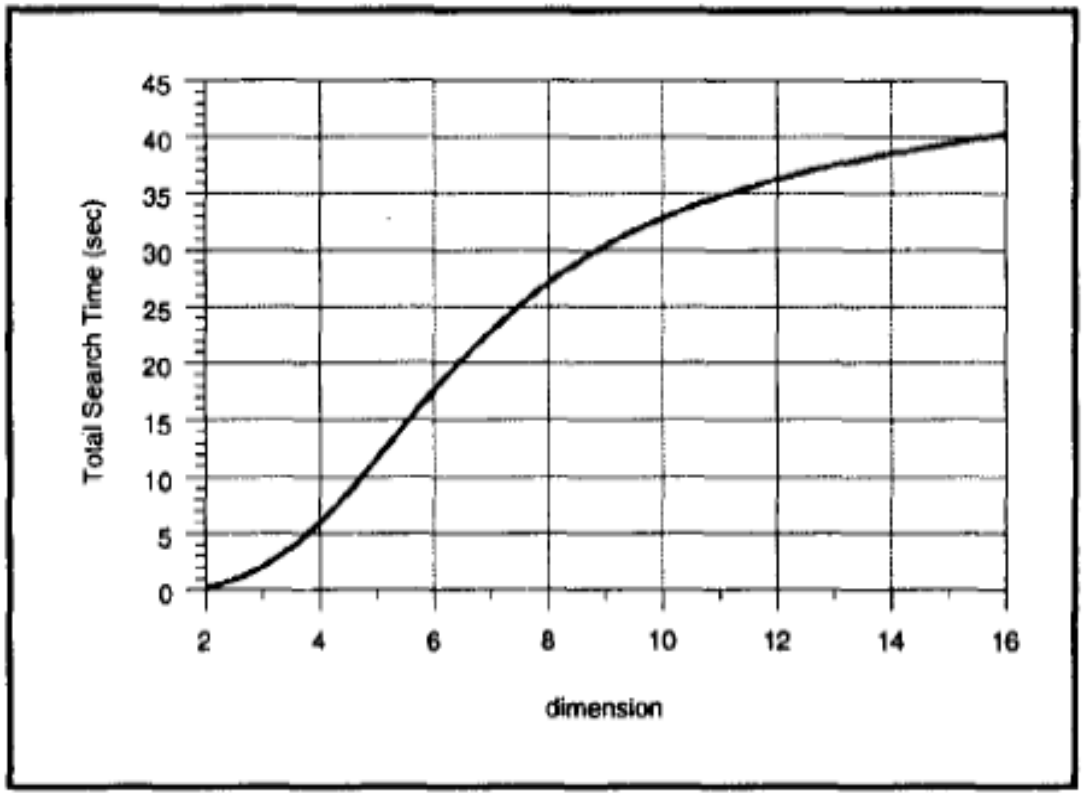
1.2 Index page fanout in the R-tree



Comment:

2. Page size is 4096 bytes.
3. Each MBR occupies 2*K*4 bytes where K is the number of dimension.
4. Fanout = 526/K.

1.2 Performance of the R-tree



Citation: The X-tree : An Index Structure for High-Dimensional Data, VLDB 1996

1.3 Poor concurrency performance

- The R-tree concurrency control cannot separate structure changes (e.g., page split, parent page update, etc) from content changes (e.g., insertion and deletion in data pages, etc).
- The R-tree needs extra structures (comparing with the II-tree) to achieve concurrency control.

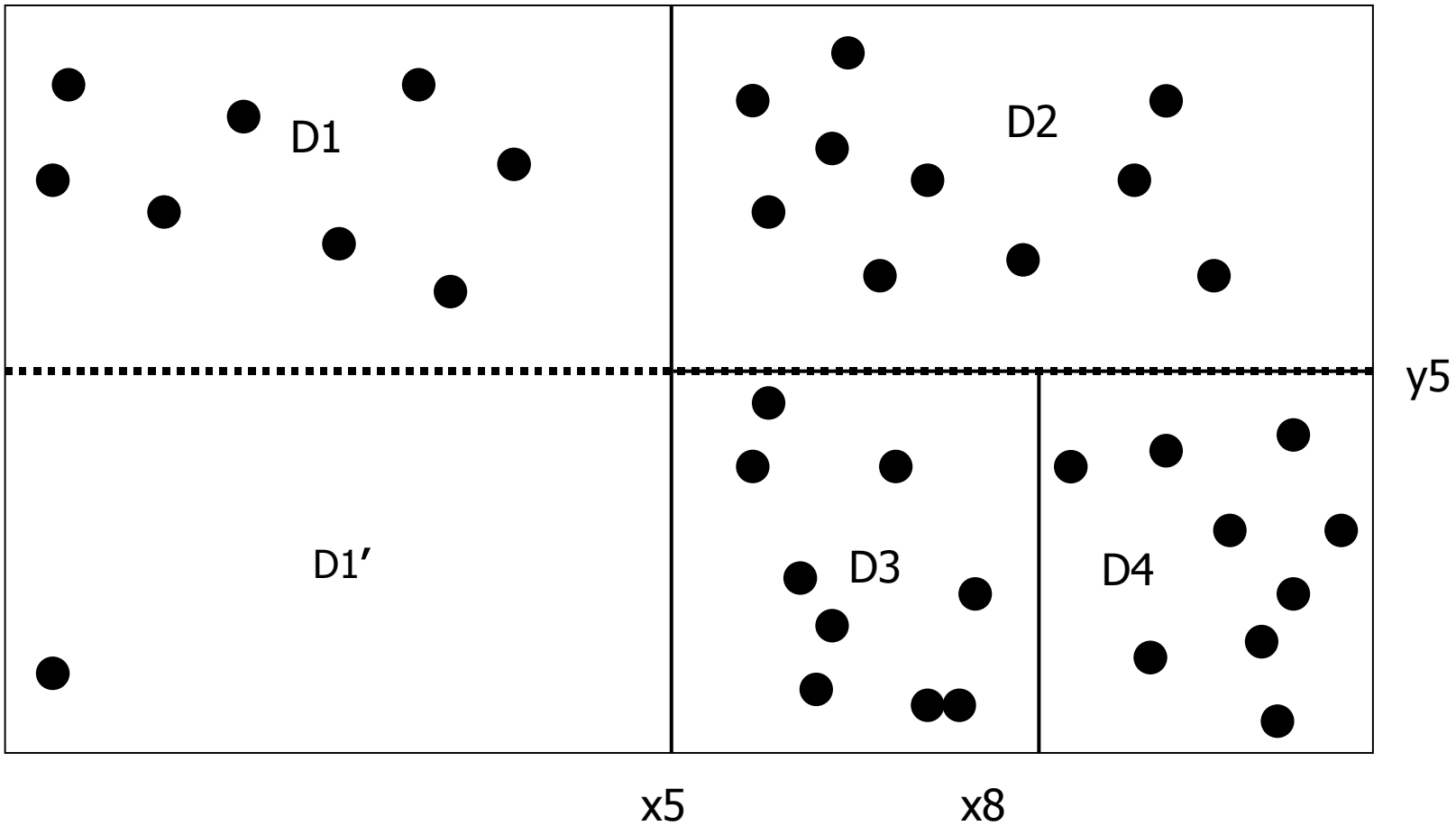
Overview

- Why not R-tree
- Why not other kd-tree variants
- The motivation of the hB-pi* tree
- The hB-pi* tree

2. Why not other kd-tree variants

1. The K-D-B-tree can not guarantee a minimum storage utilization
2. The LSD-tree is not a balanced tree and is sensitive to the data insert order.

2.1 K-D-B-tree can not guarantee a minimum storage utilization

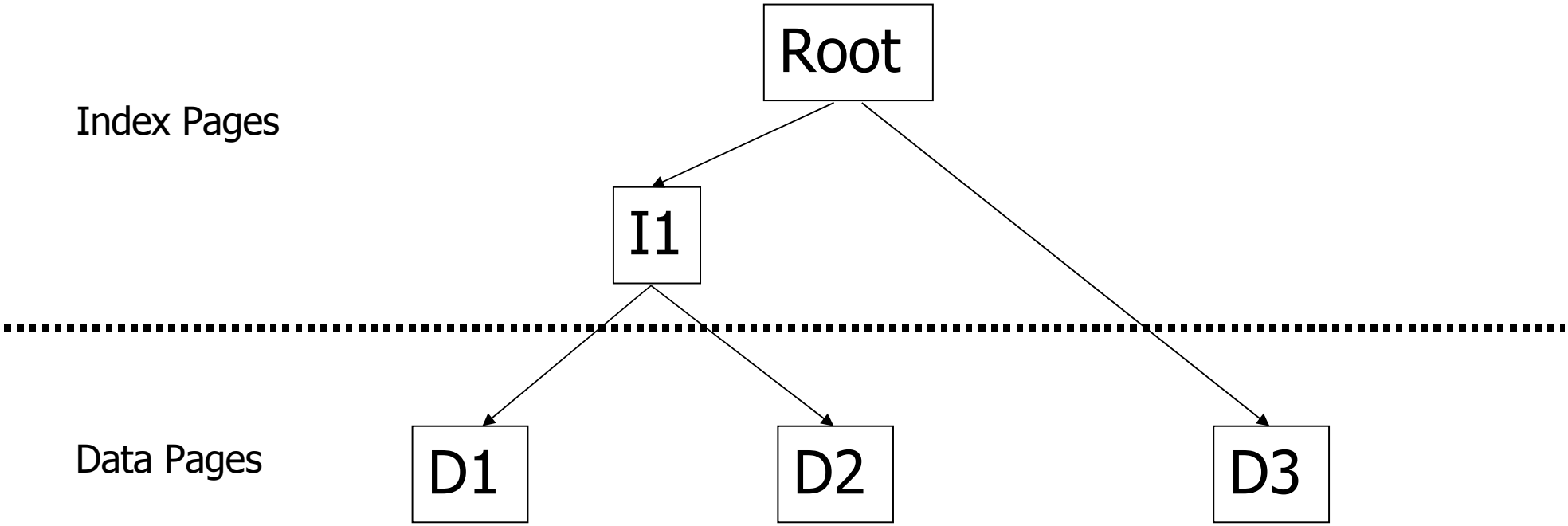


Comments:

1. the data page fanout is 18.
2. the index page fanout is 3.

2.2 LSD-tree is not a balanced tree

- The heights of LSD-tree's external sub-trees can be different.



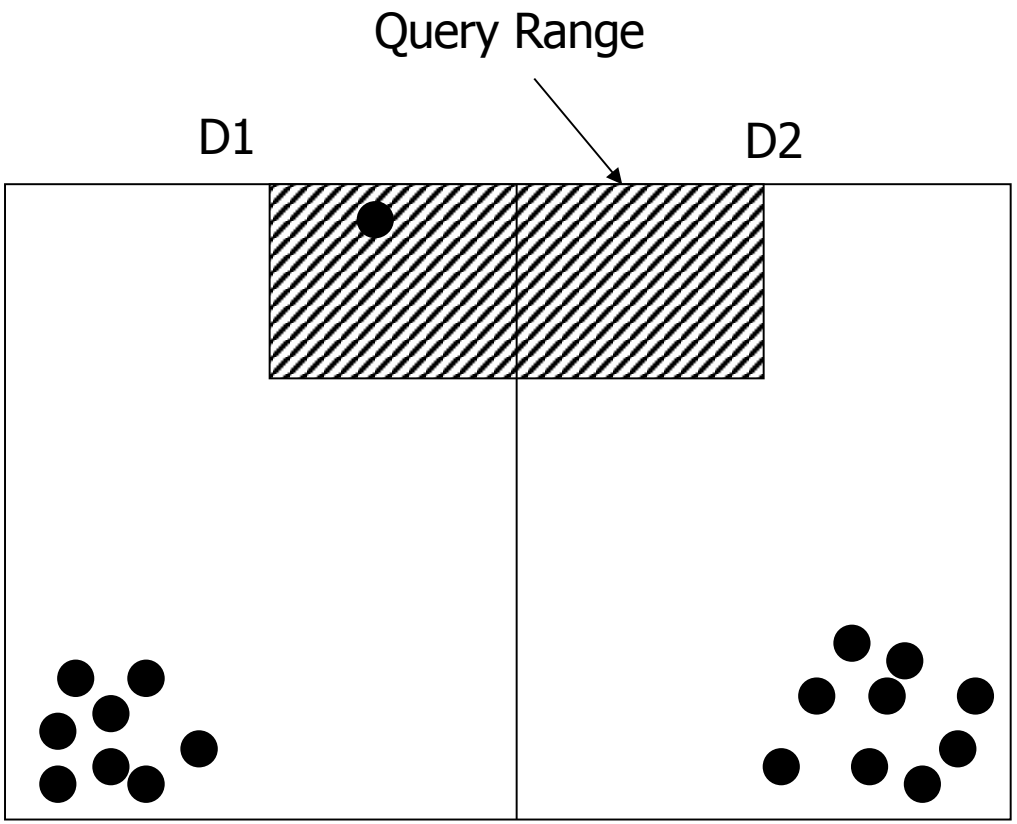
Querying multi-dimensional data

- Why not R-tree
- Why not other kd-tree
- The motivation of the hB-pi* tree
- The hB-pi* tree

3. The motivation of the hB-pi* tree

1. The impediment of the hB-pi tree
2. Straightforward solution

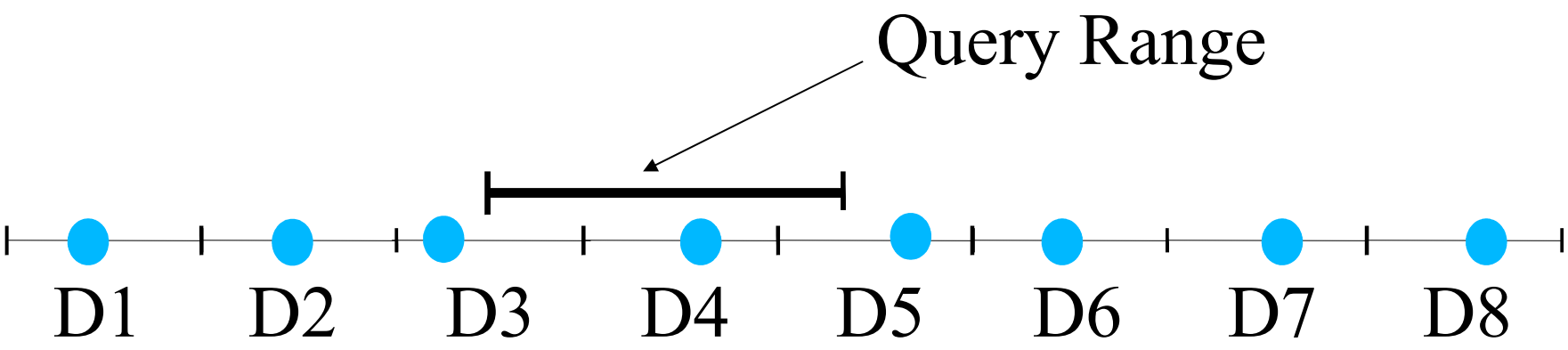
3.1 The impediment of the hB-pi tree - Example



"**Empty Data Page**": the data page that is accessed during the range query, but does not contain any data in the query range (e.g., D2).

Comment: The hB-pi tree (and all the other kd-tree variants) indexes the whole space no matter whether or not there is any data in some sub-spaces

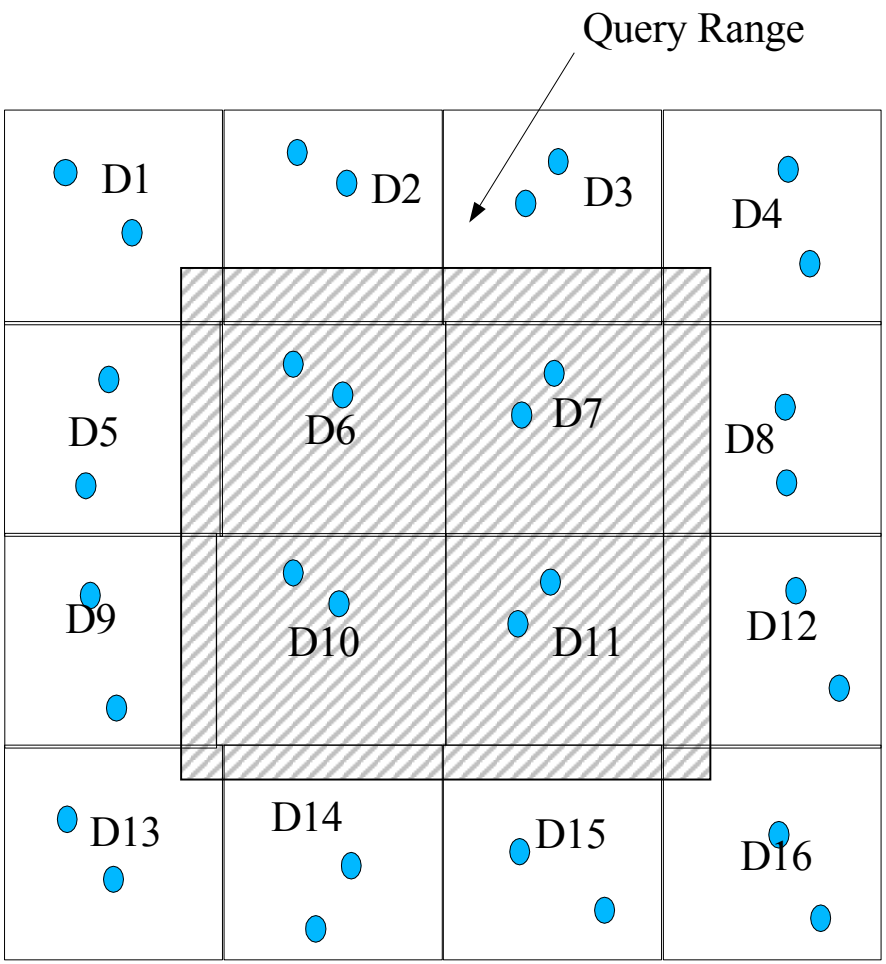
3.1 The impediment of the hB-pi tree – Example (cont)



Comments:

2. Query range is 1/3 of the whole space
3. At most access two empty data page, 25% of the total pages.

3.1 The impediment of the hB-pi tree – Example (cont)



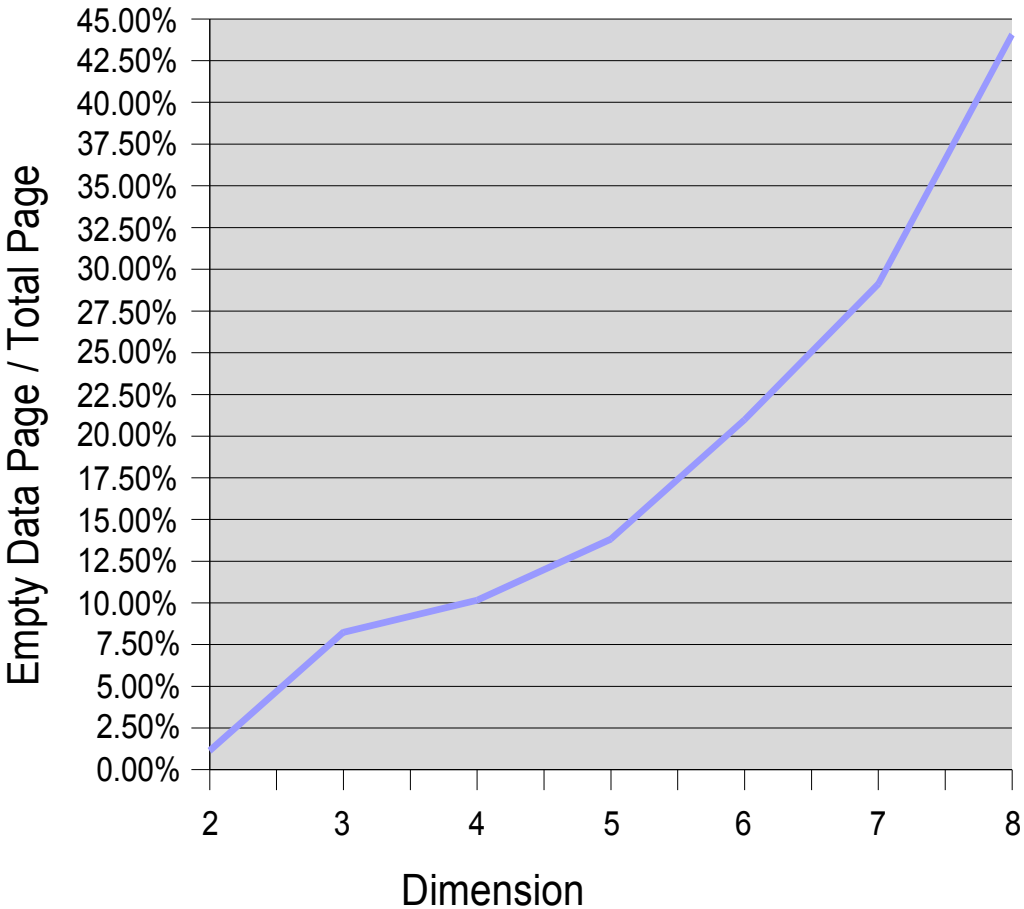
Comments:

2. Query range is 1/3 of the whole space
3. At most access 12 empty data page. They account for 75% of the total pages.

Observation:

the higher the dimension, the higher the empty data page rate could be

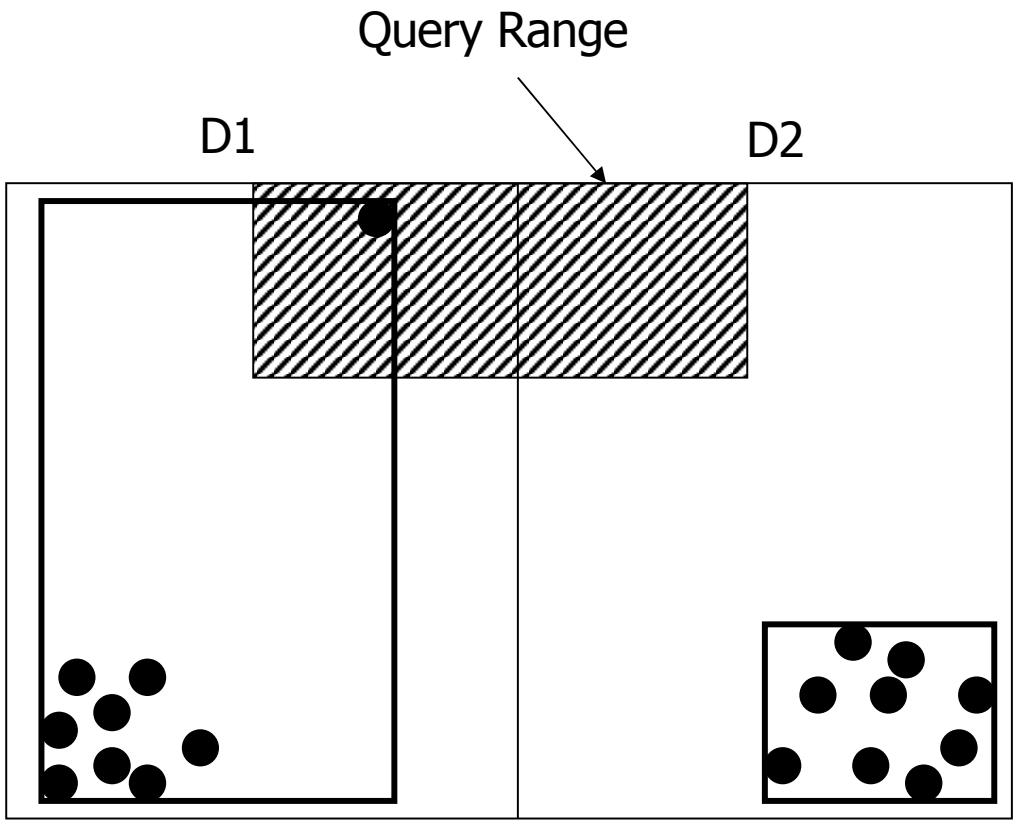
3.1 The impediment of the hB-pi tree– Experimental results



Fact:
The real data in high-dimensional space are highly clustered. The data occupy only some subspaces of the high-dimensional space.

Comment: The experimental results for 100,000 uniformly distributed point data sets

3.2 Straightforward solution - Example



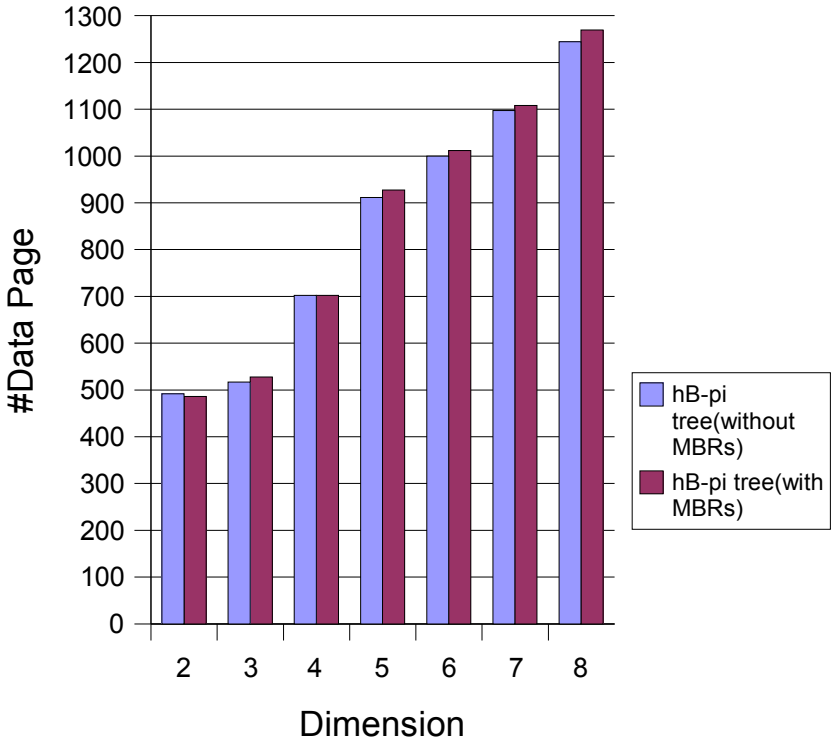
Problems:

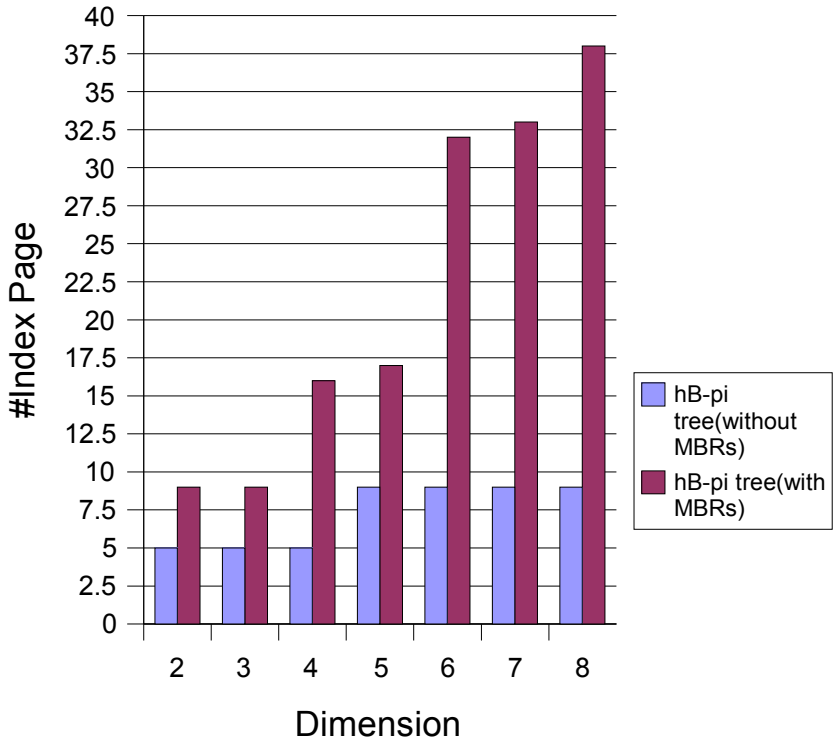
1. Fanout of index pages decreases rapidly with growing dimensionality.
2. Outlier can cause the MBR method to remain efficient.
3. The update of the MBR will decrease concurrency performance.

Comments:

1. The data page's page boundary is the MBR that encloses all the data inside D.
2. The index page's page boundary is the MBR that encloses all the MBRs of its children.
3. The index page stores the MBRs of its children.

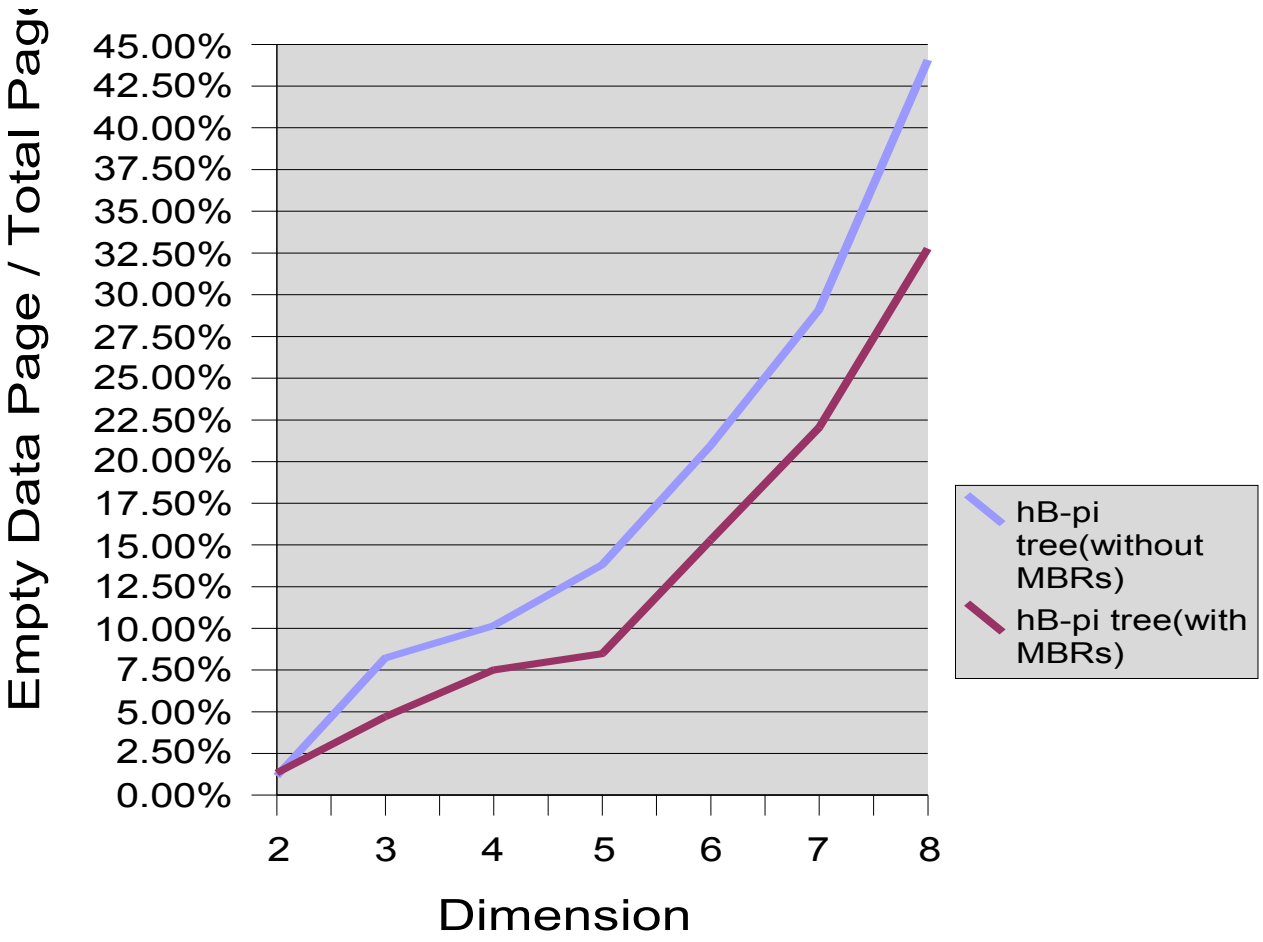
3.2 Straightforward solution – Tree structure



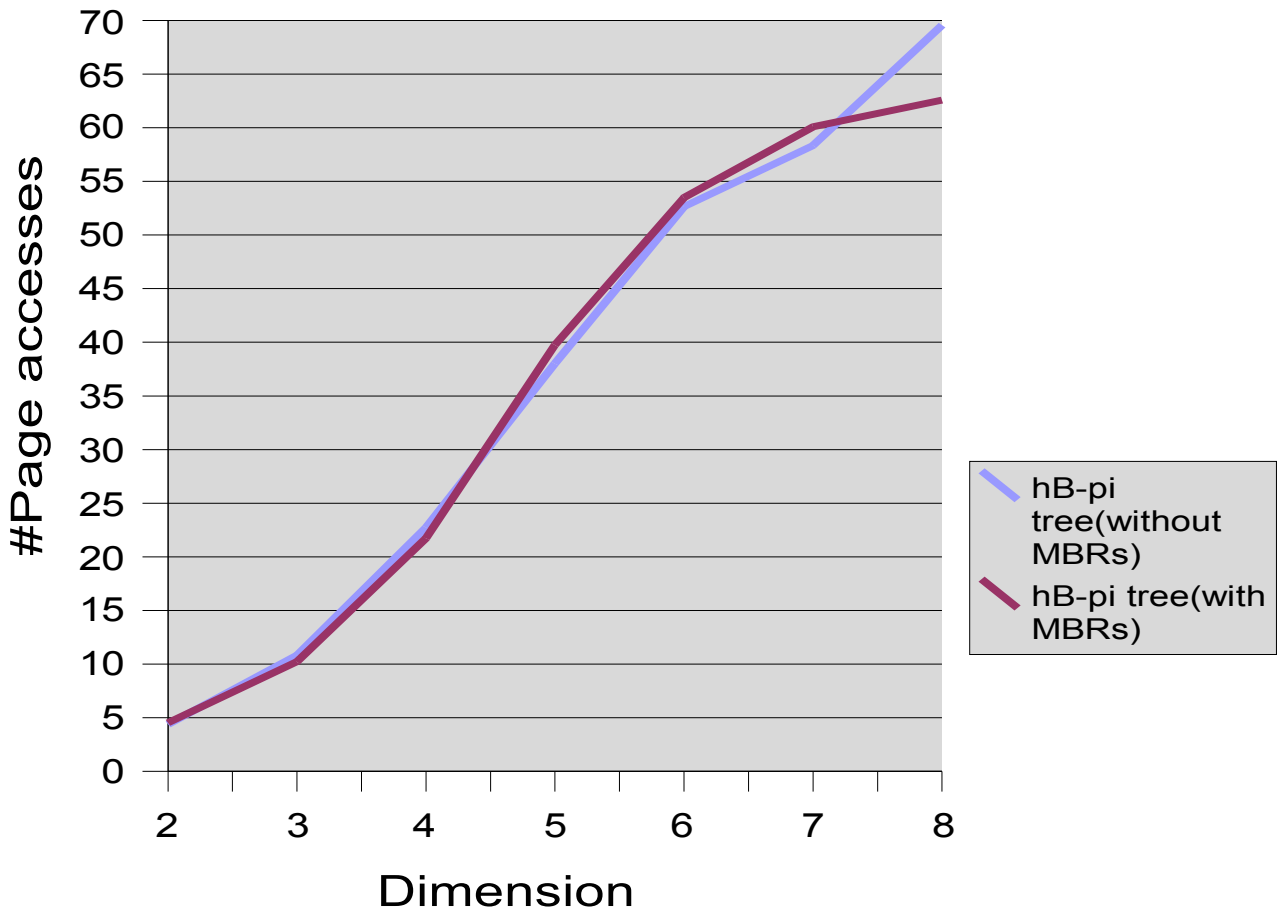


Comment: 100K uniformly distributed point data sets.

3.2 Straightforward solution – Empty data page rate



3.2 Straightforward solution – Query performance



Comments:

2. The hB-pi tree (with MBRs) requires more index page accesses.
3. A few outliers might make the MBRs unnecessarily large.

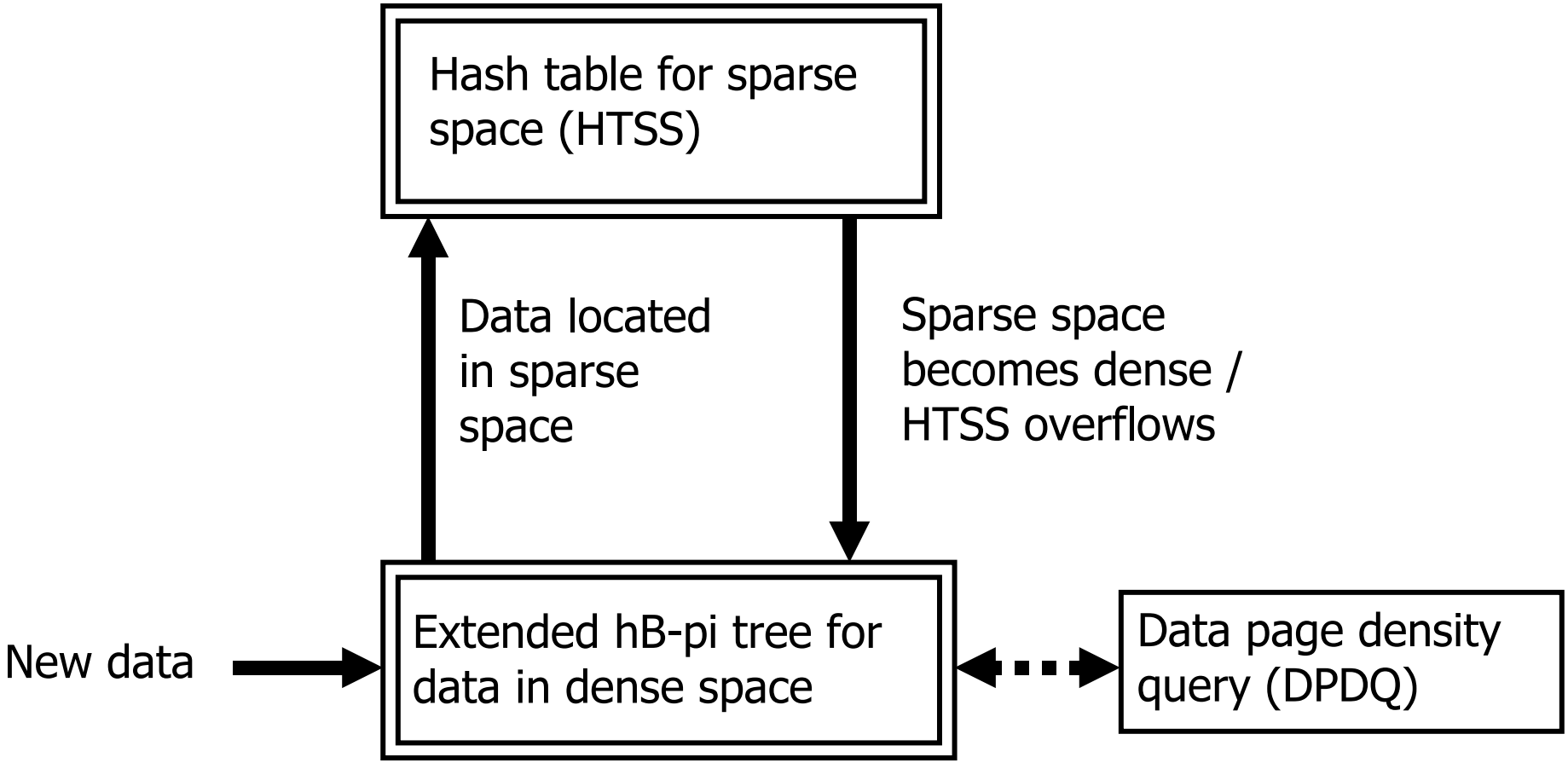
Querying multi-dimensional data

- Why not R-tree
- Why not other kd-tree
- The motivation of the hB-pi* tree
- The hB-pi* tree

4. The hB-pi* tree

1. Basic structure
2. Tree construction
3. Basic query processing
4. Experimental results

4.1 Basic structure



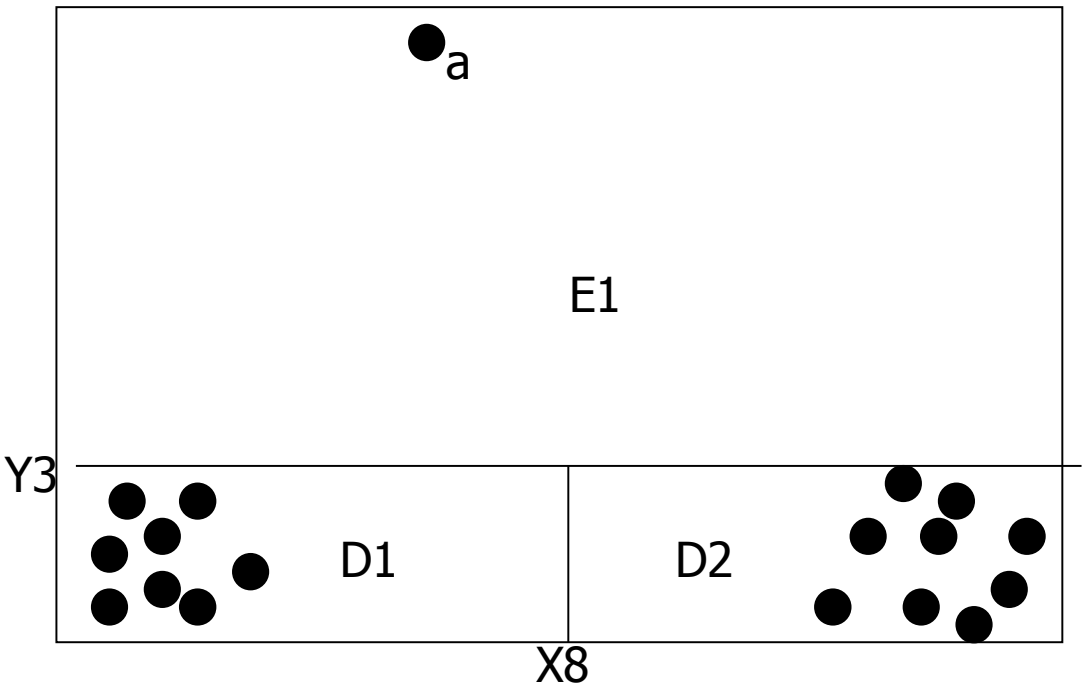
4.1 Basic structure - cont

HTSS

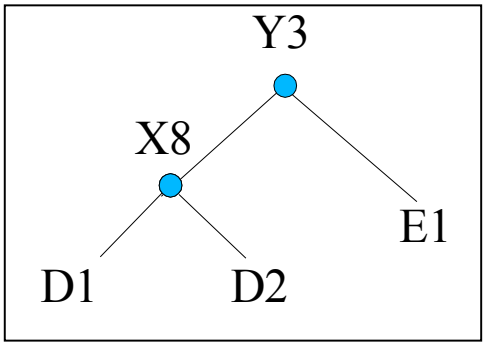
Record List	Data
E1	a

DPDQ

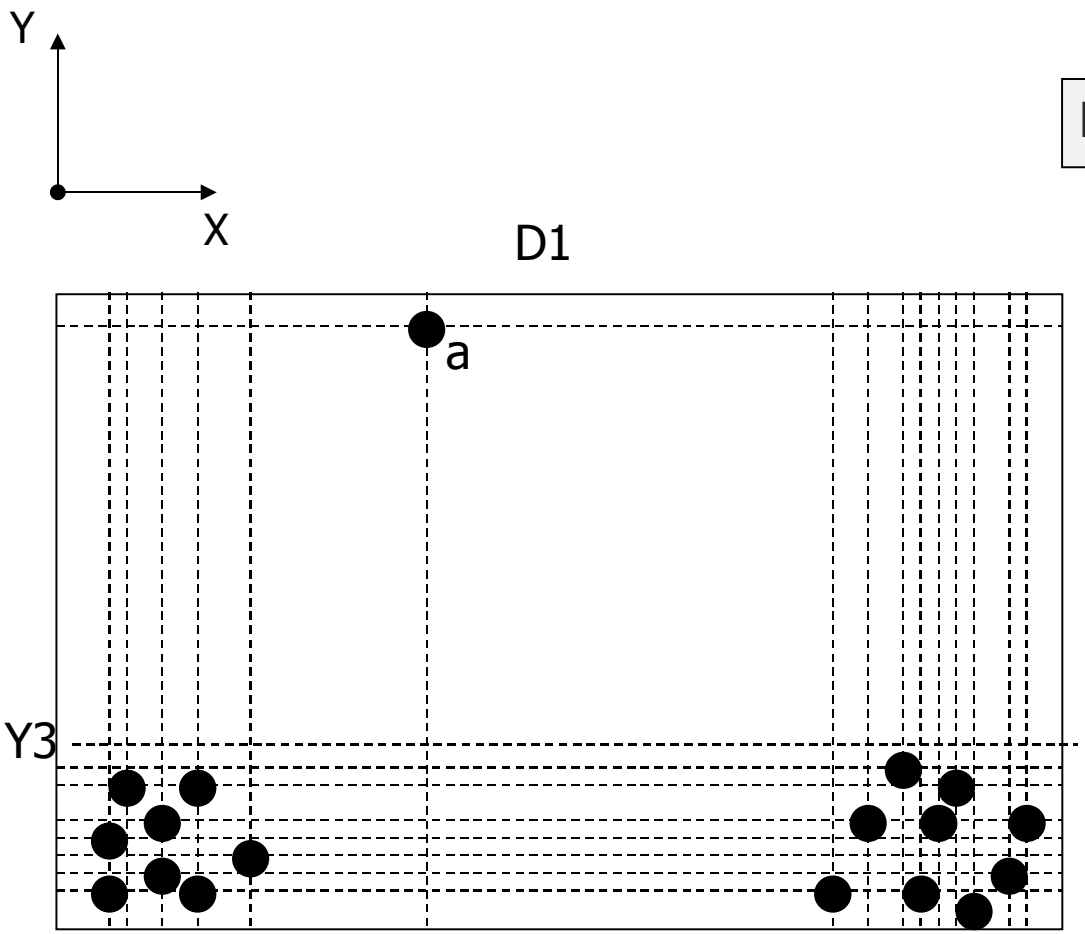
Data page	Density
D1	15.45
D2	16.09



P



4.2 Tree construction



HTSS

Record List	Data

DPDQ

Data page	Density

X-dimension: No sparse space

Y-dimension: Sparse space detected

Note:

1. Detecting sparse space is implemented by detecting dense space at each dimension
2. **Dense space**:

2.1 contains at least 2/3 data AND

2.2 density is $2^{dim/2}$ of the page density.

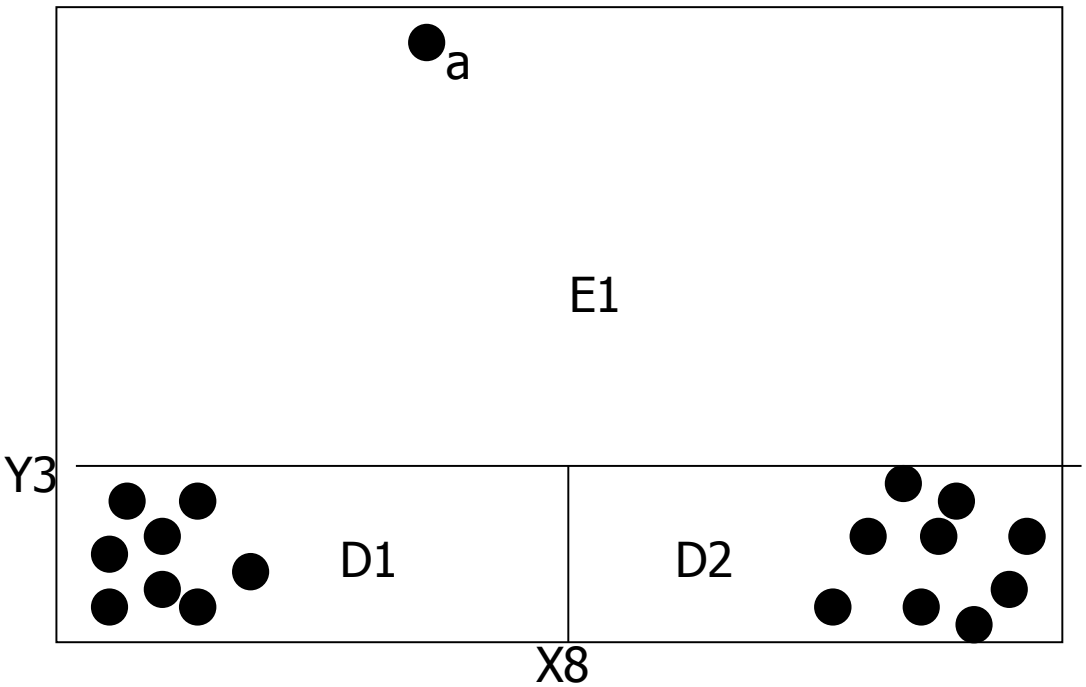
4.2 Tree construction - cont

HTSS

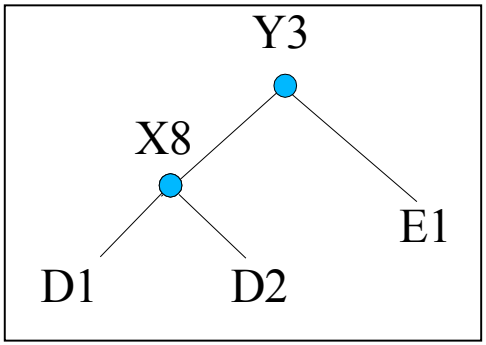
Record List	Data
E1	a

DPDQ

Data page	Density
D1	15.45
D2	16.09



P



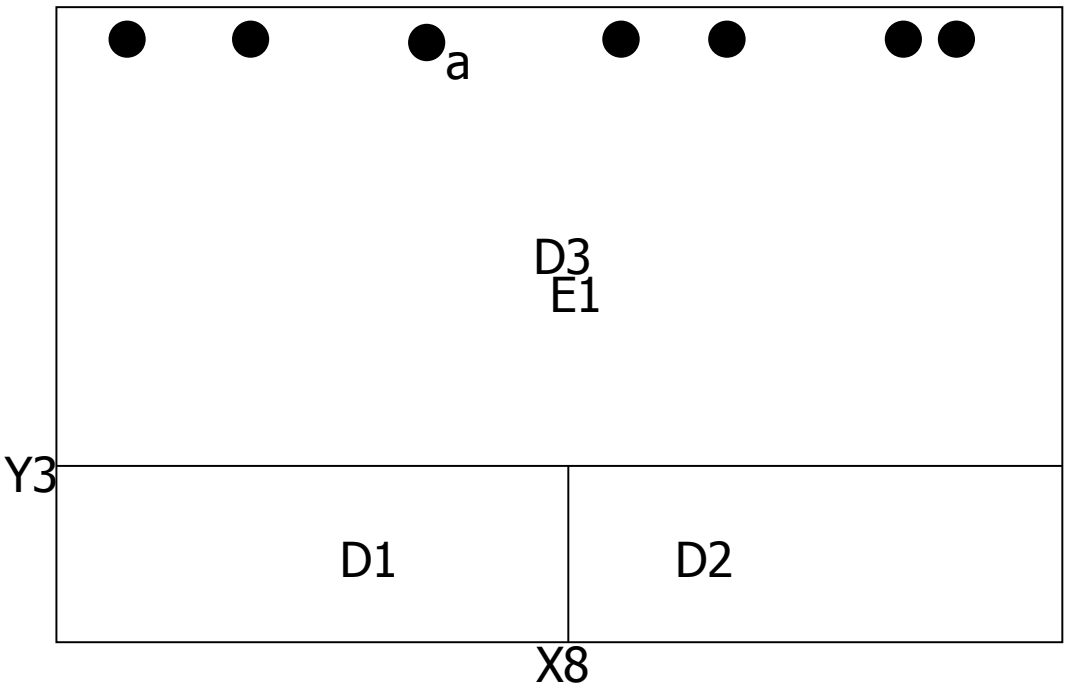
4.2 Tree construction – Page Reentry

HTSS

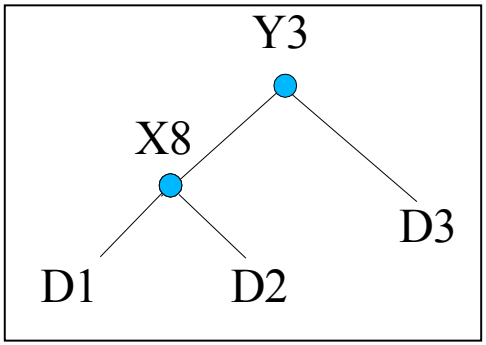
Record List	Data
E1	a

DPDQ

Data page	Density
D3	5.68
D1	15.45
D2	16.09



P



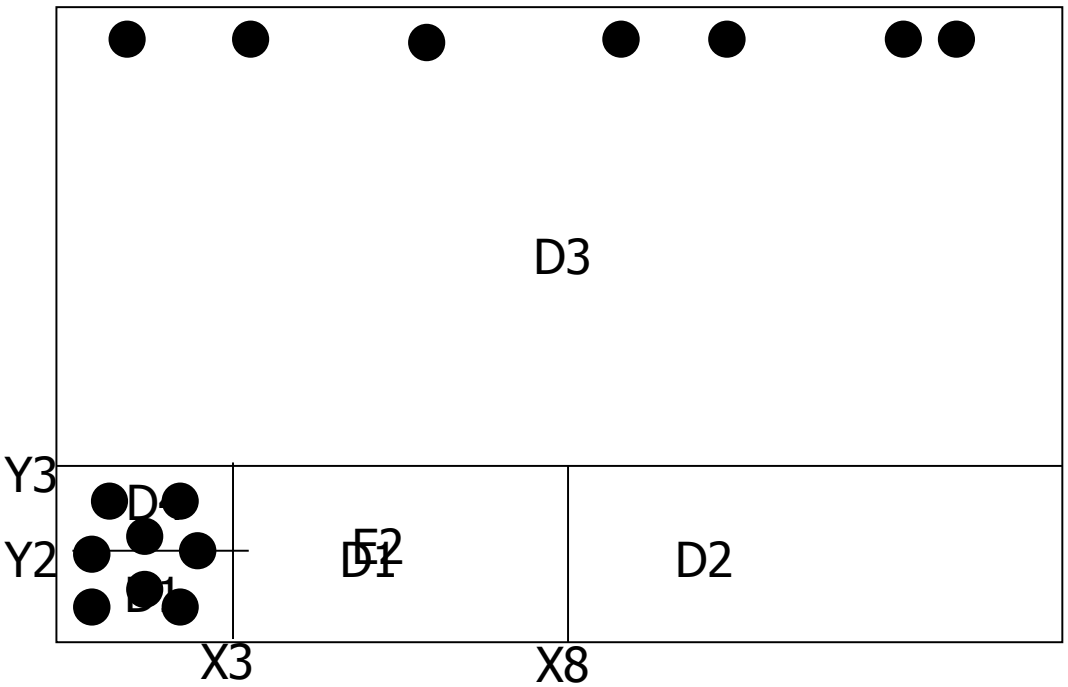
4.2 Tree construction - DPDQ

HTSS

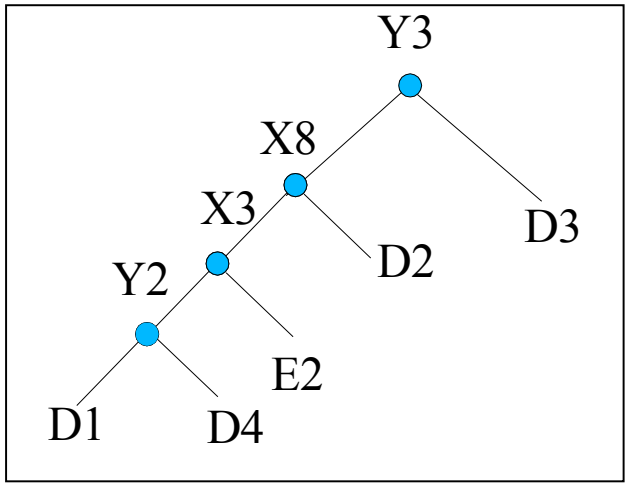
Record List	Data
E2	....

DPDQ

Data page	Density
D3	5.68
D2	16.09
D1	30.00
D4	30.00



P



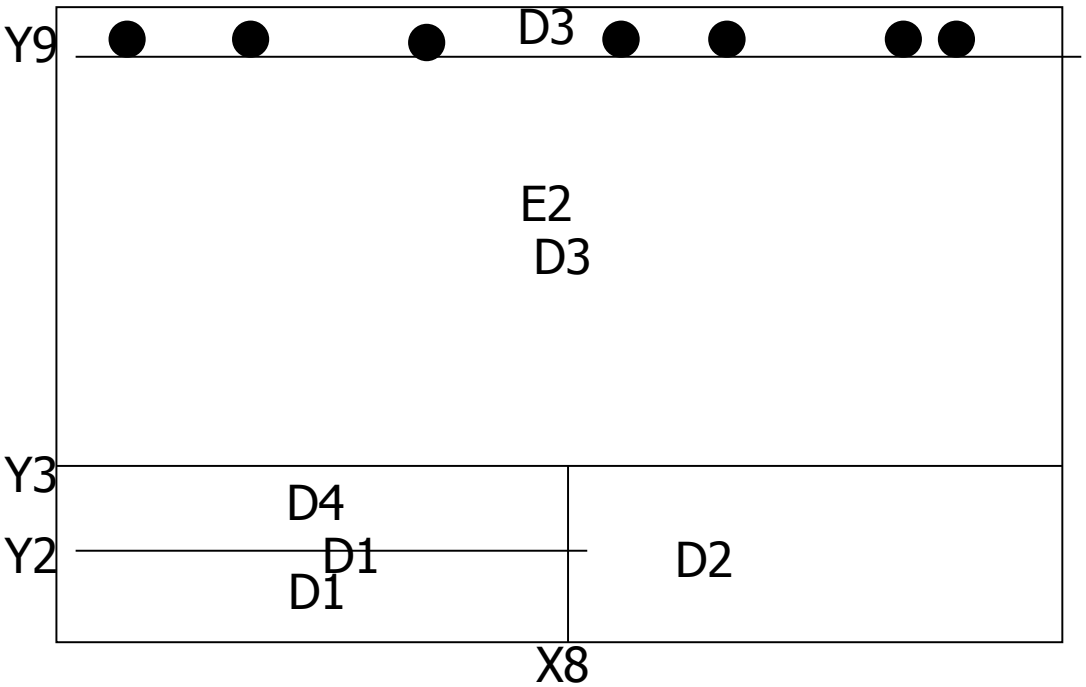
4.2 Tree construction - DPDQ

HTSS

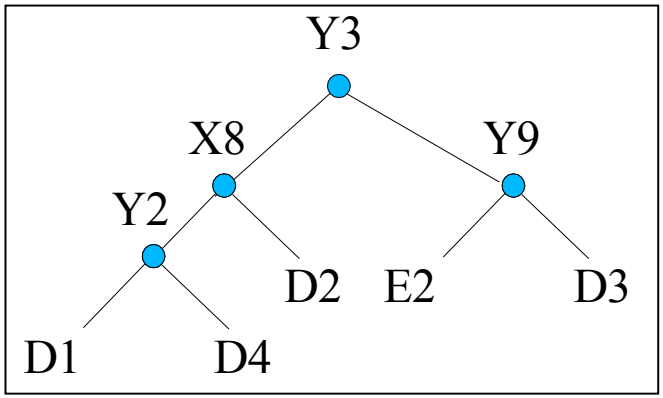
Record List	Data
E2	……

DPDQ

Data page	Density
D2	16.09
D3	17.05
D1	25.45
D4	26.37



P



4.2 Tree construction - algorithm

```
Algorithm Insert(dataRecord R)
Find the data page (D) or the record list
(L) that should contain data R
If R is contained by D
        Add R to D
        If D overflows
            Invoke PageSplit on D
Else if R is contained by L
        Insert R to L
        If the size of L  is bigger than 1/3 page size
            Invoke PageReEntry on L
        Else
            If HTSS overflows
                Choose record list L' from HTSS
                Invoke PageReEntry on L'
If page split or page reentry occurs, post the update to parent pages
```

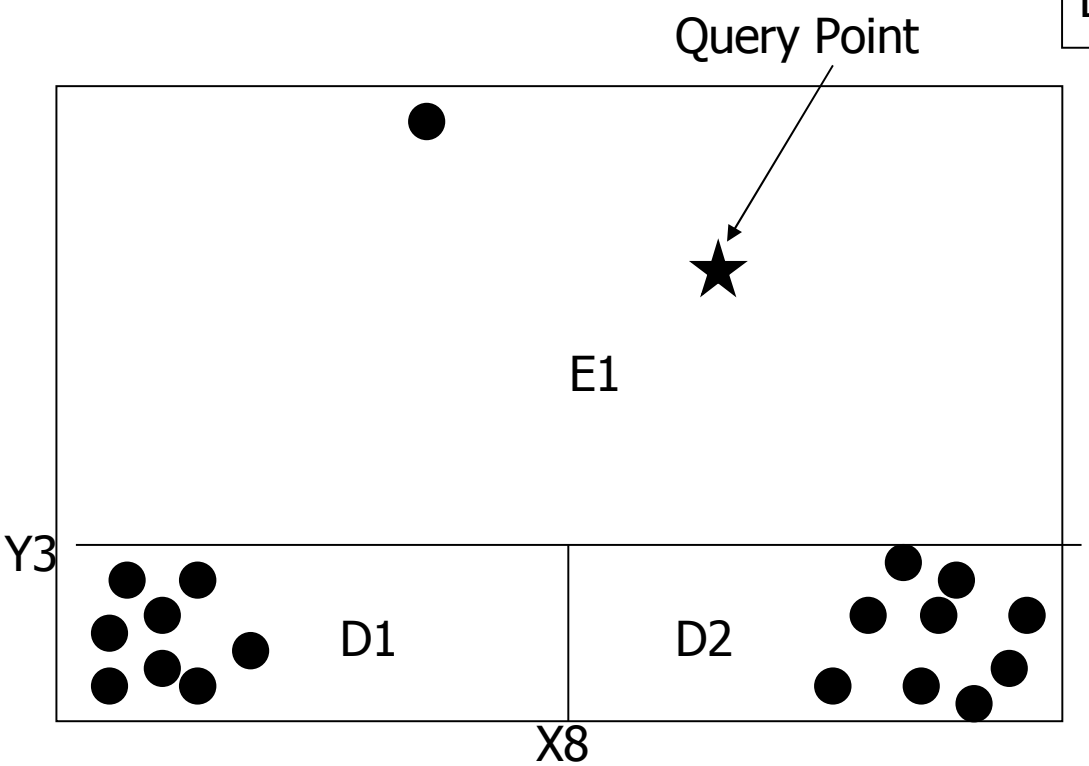
4.3 Basic query processing – exact match query

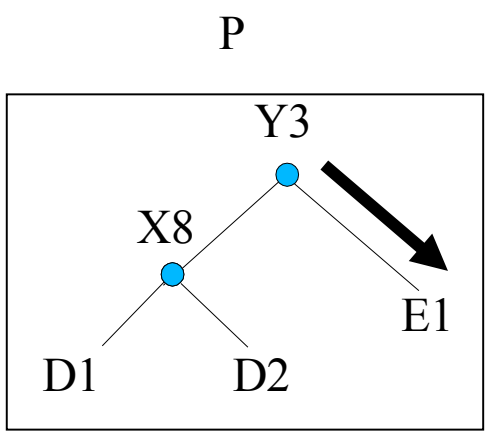
HTSS

Record List	Data
E1	a

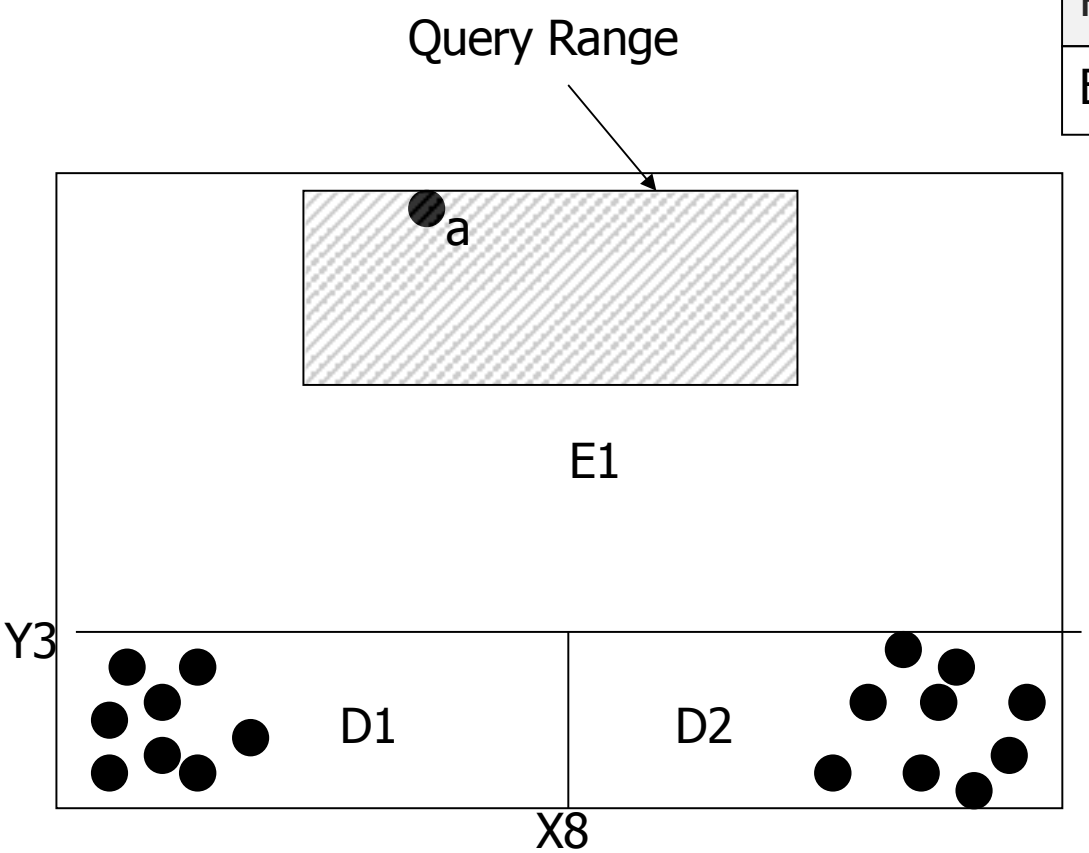
DPDQ

Data page	Density
D1	15.45
D2	16.09





4.3 Basic query processing – range query

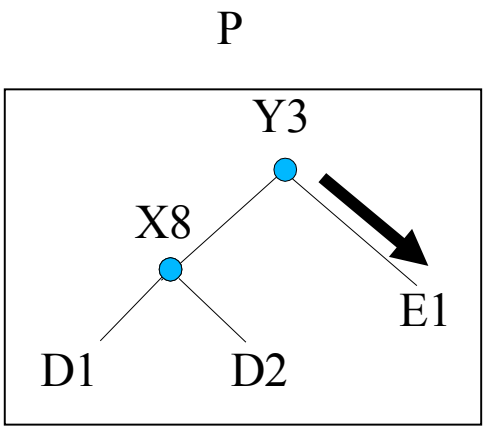


HTSS

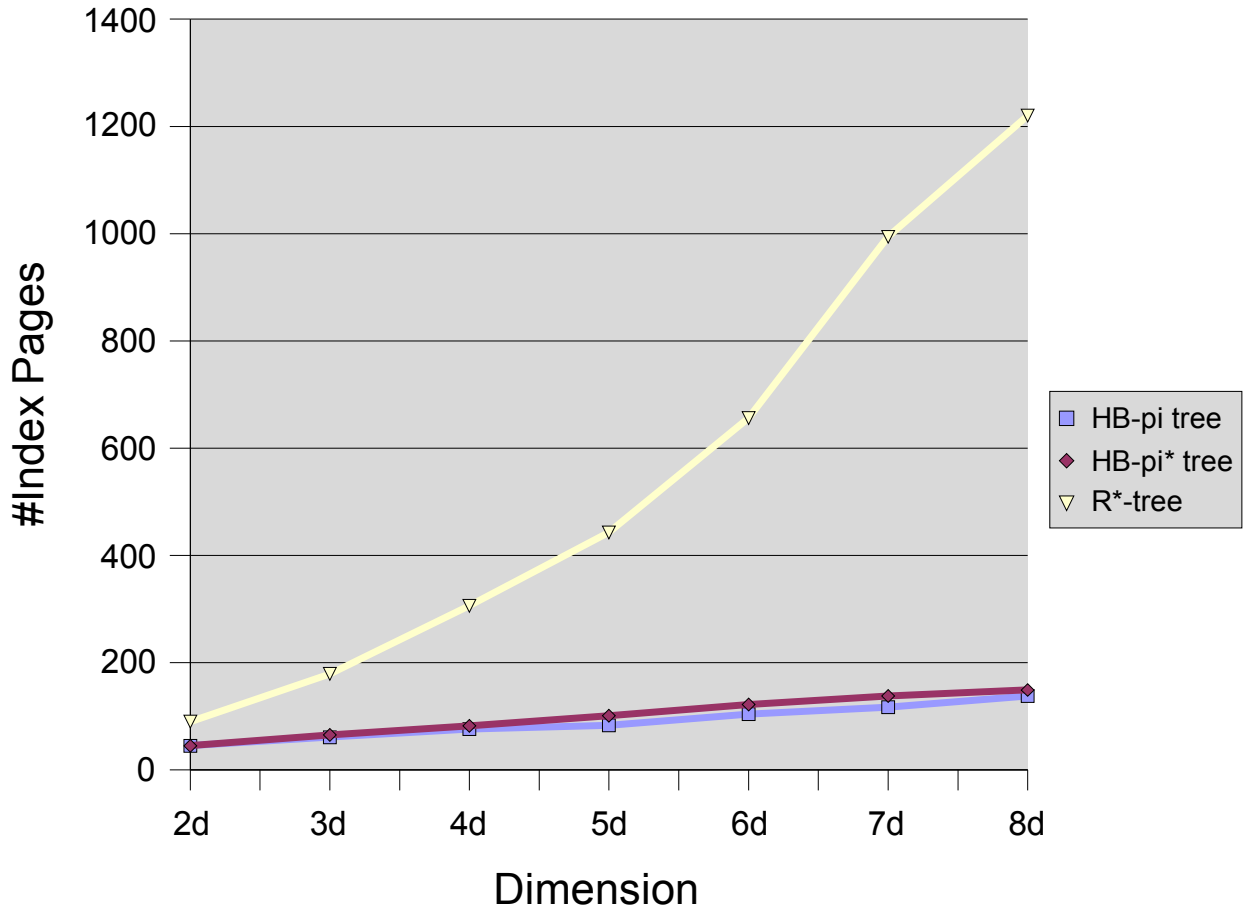
Record List	Data
E1	a

DPDQ

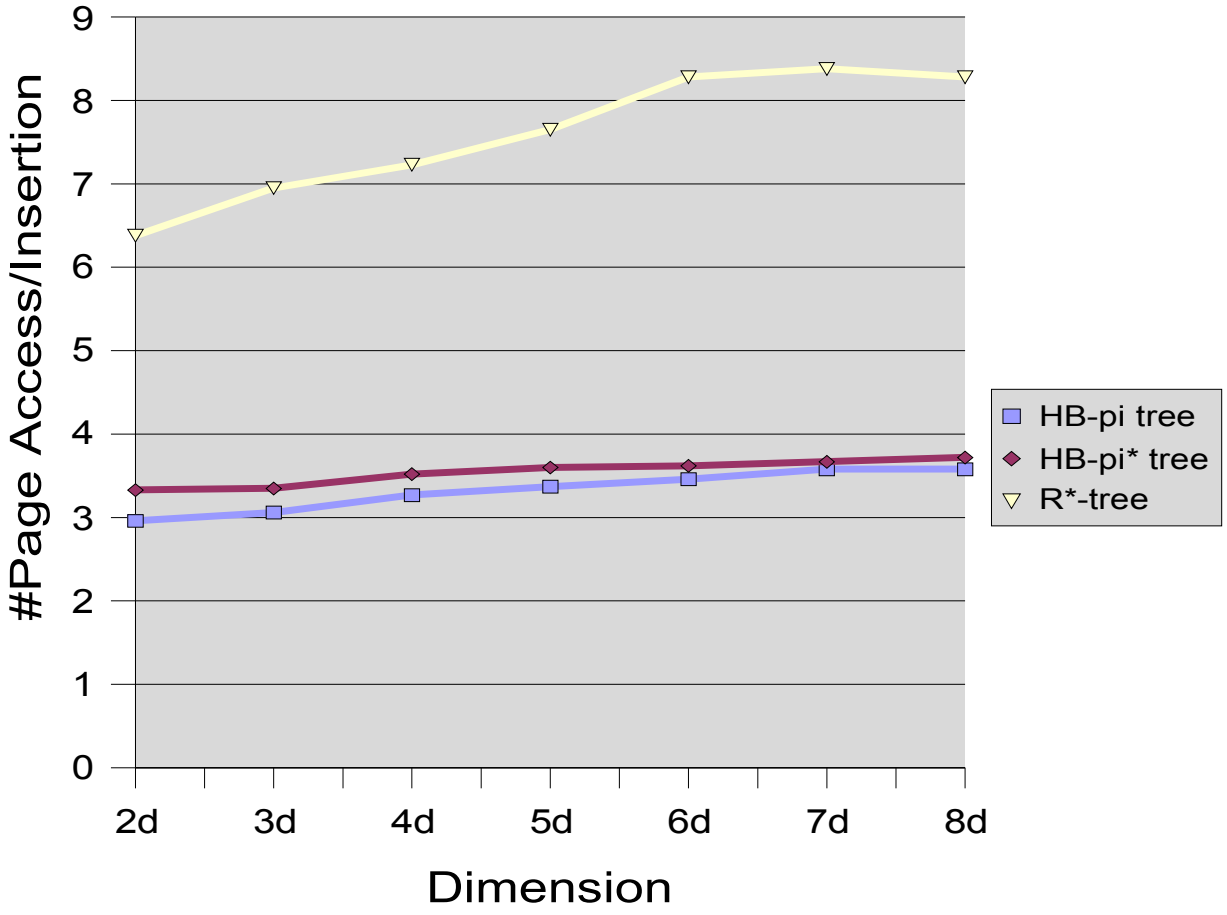
Data page	Density
D1	15.45
D2	16.09



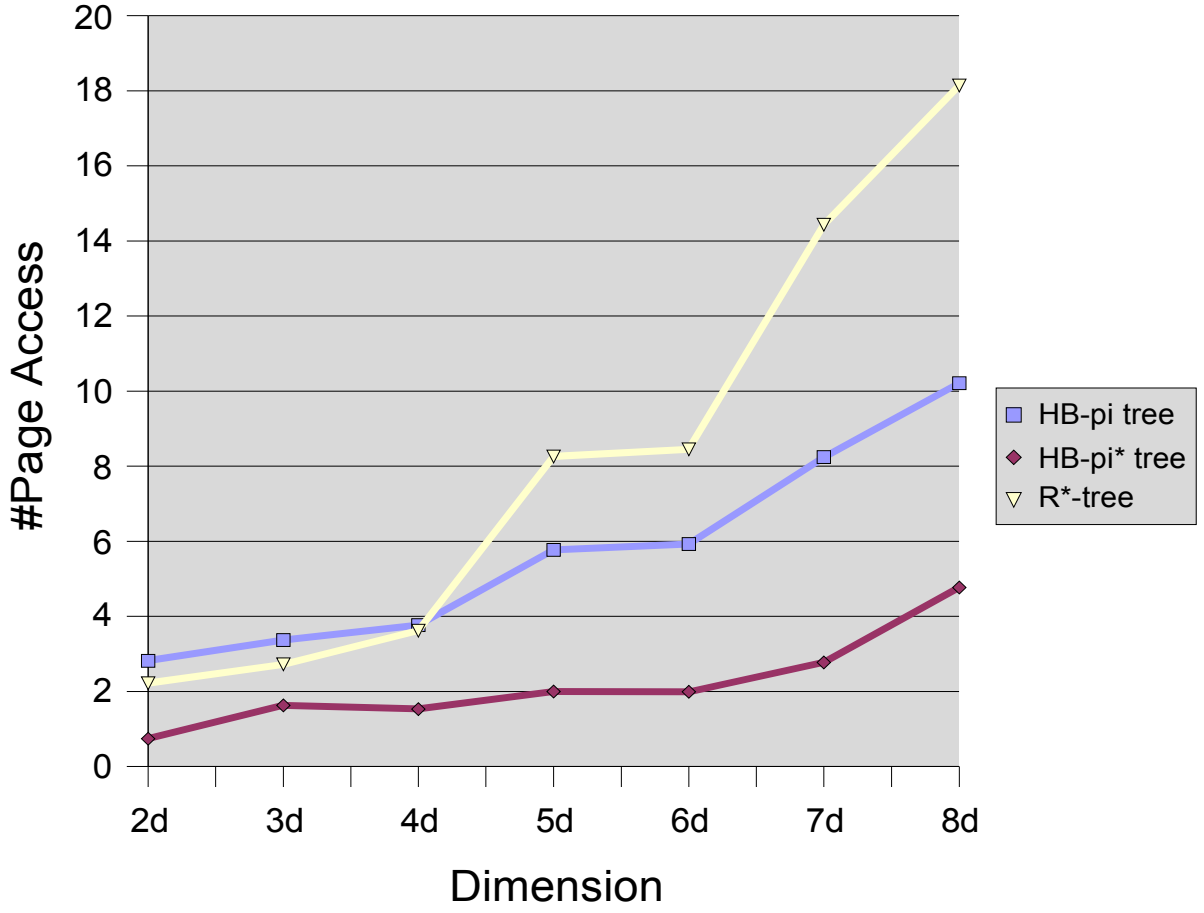
4.4 Experimental results - #Index pages



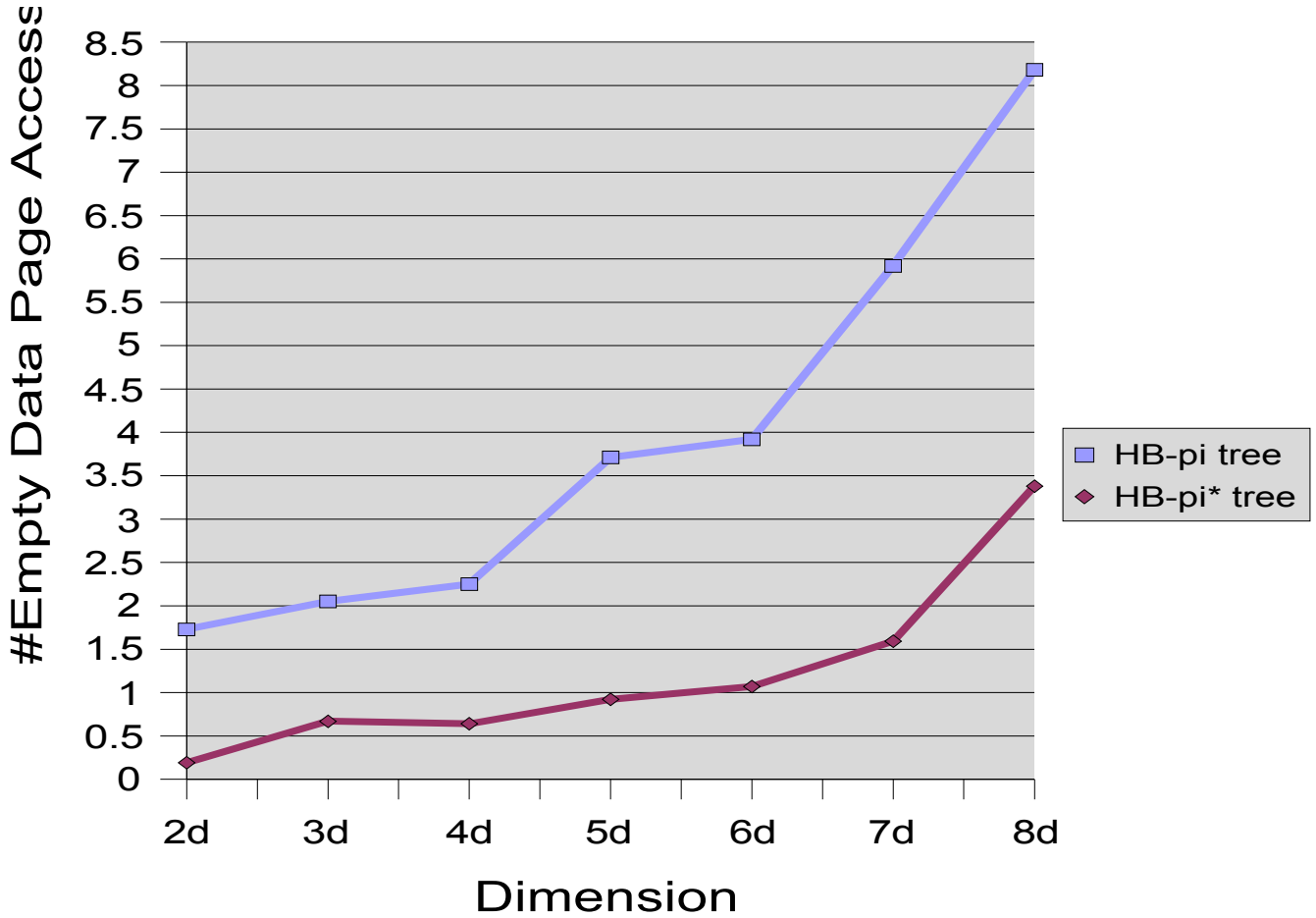
4.4 Experimental results – Insertion cost



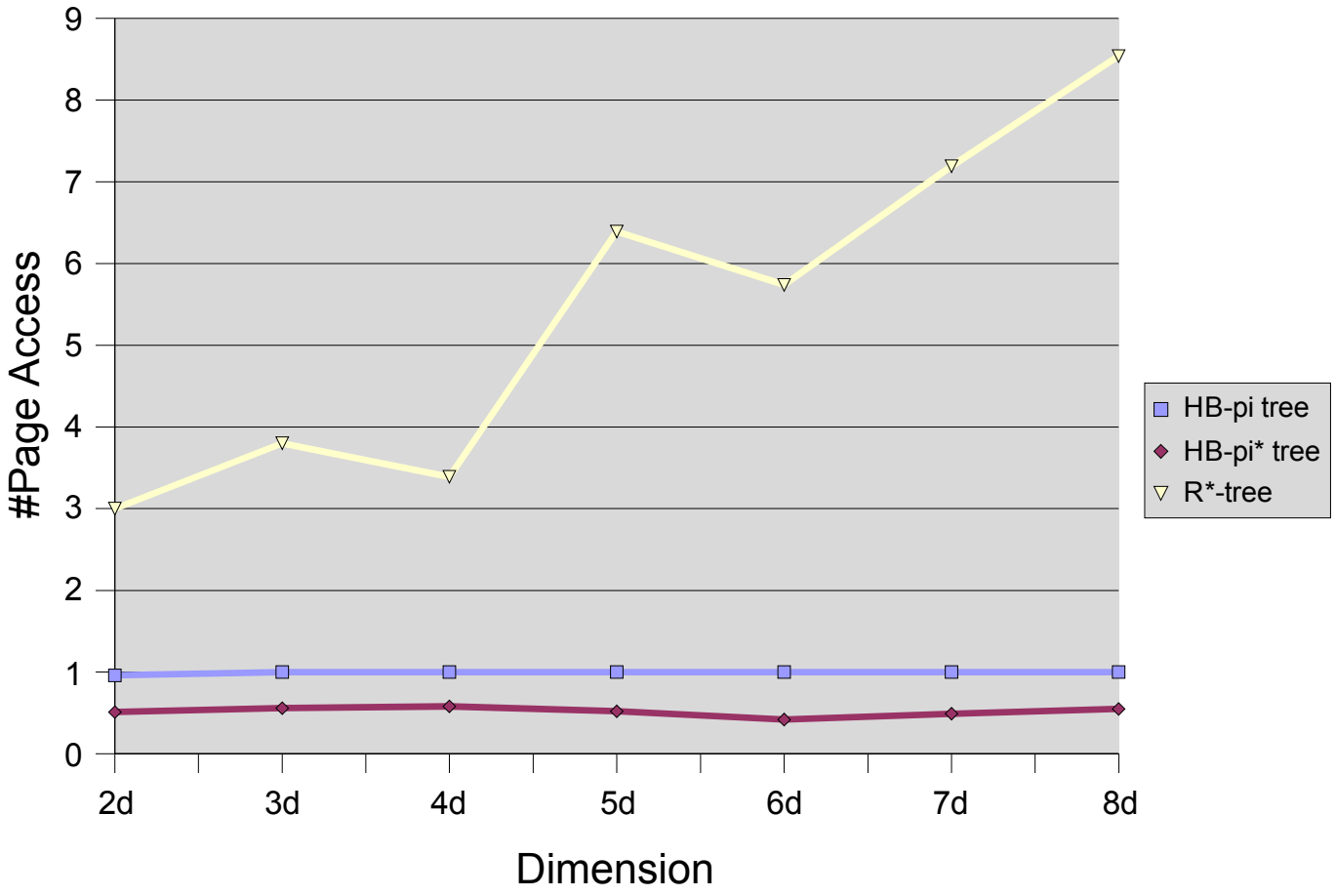
4.4 Experimental results – Range query



4.4 Experimental results – Range query - cont



4.4 Experimental results – Exact match query



Thanks!