A New Approach for Optimization of Dynamic Metric Access Methods Using an Algorithm of Effective Deletion

Renato Bueno

Daniel S. Kaster

Agma J. M. Traina

Caetano Traina Jr

University of São Paulo – USP
São Carlos - SP - Brasil

20th International Conference on Scientific and Statistical Database Management (SSDBM)
July 9-11, 2008, Hong Kong, China

Outline

- ➡ Introduction and Background
- Effective deletion algorithm
- **An Overlap Reduction and Optimization Technique for MAM**
- Experiments
- Conclusions

Introduction

- Traditional DBMS:
  - Numbers and ter strings

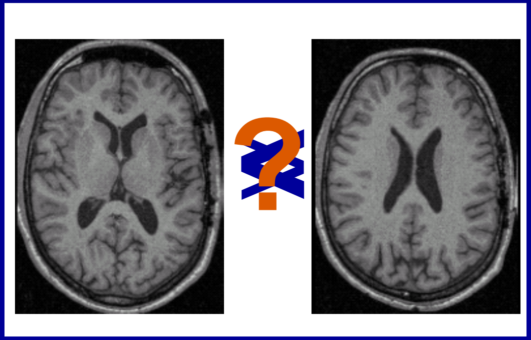
  Ex.:Payrolls, ba

Total ordering property

- Today:
  - **DBMS are being increasingly required to support other, much more complex, data types.**

Ex.: images, audio, finge ints, time series and netic sequences

~~**To l ord ring pr erty**~~



Introduction

The usual comparison operators (=, <, >) are meaningless for these data.

Images are rarely equal...

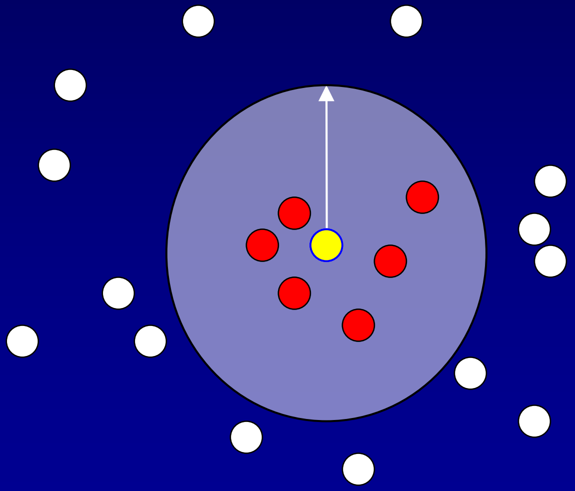
... they are similar.

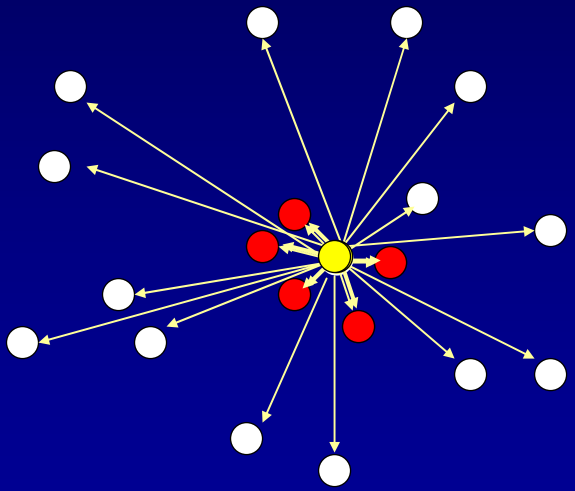
Similarity Queries

Similarity queries

Similarity operators are much more useful:

- **Range query - Rq:**
- **k-nearest neighbor query - k-NNq**

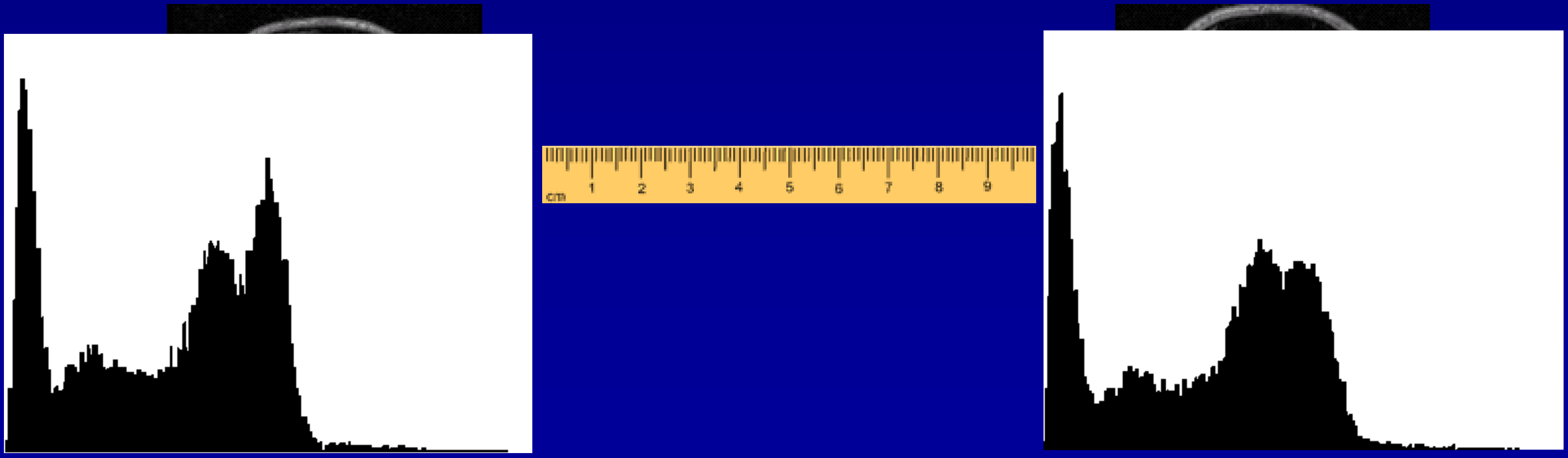




Select the 5
nearest neighbors

Introduction

- Similarity queries
  - Multimedia data comparison usually considers some features extracted from the data elements.
- Ex.: The definition of how to compare two images is a subjective factor.
  - Visual inspection: different results.



Metric Spaces

- Similarity can be adequately expressed when data is represented in a metric space:
  - only the data elements and the distances (dissimilarity) among them are available
  - there are not geometric relations
- Furthermore, there are domains that do not have a dimensionality,
  - Words, genetic sequences, audio, etc.;

Metric Spaces

- A metric domain is defined as a pair:

$$M = (S , d)$$

  - *S* : universe of valid elements
  - *d( )* : metric distance function
    - Metric distance function properties:

1 - Simmetry: **d(x, z) = d(z, x);**

2 – Non-negativity: **$0 < d(x, z) < \infty$; d(x, x) = 0**

3 – Triangular inequality: **$d(x,z) \le d(x,y)+d(y,z)$**

avoiding distance calculations

- The triangular inequality allows pruning subtrees

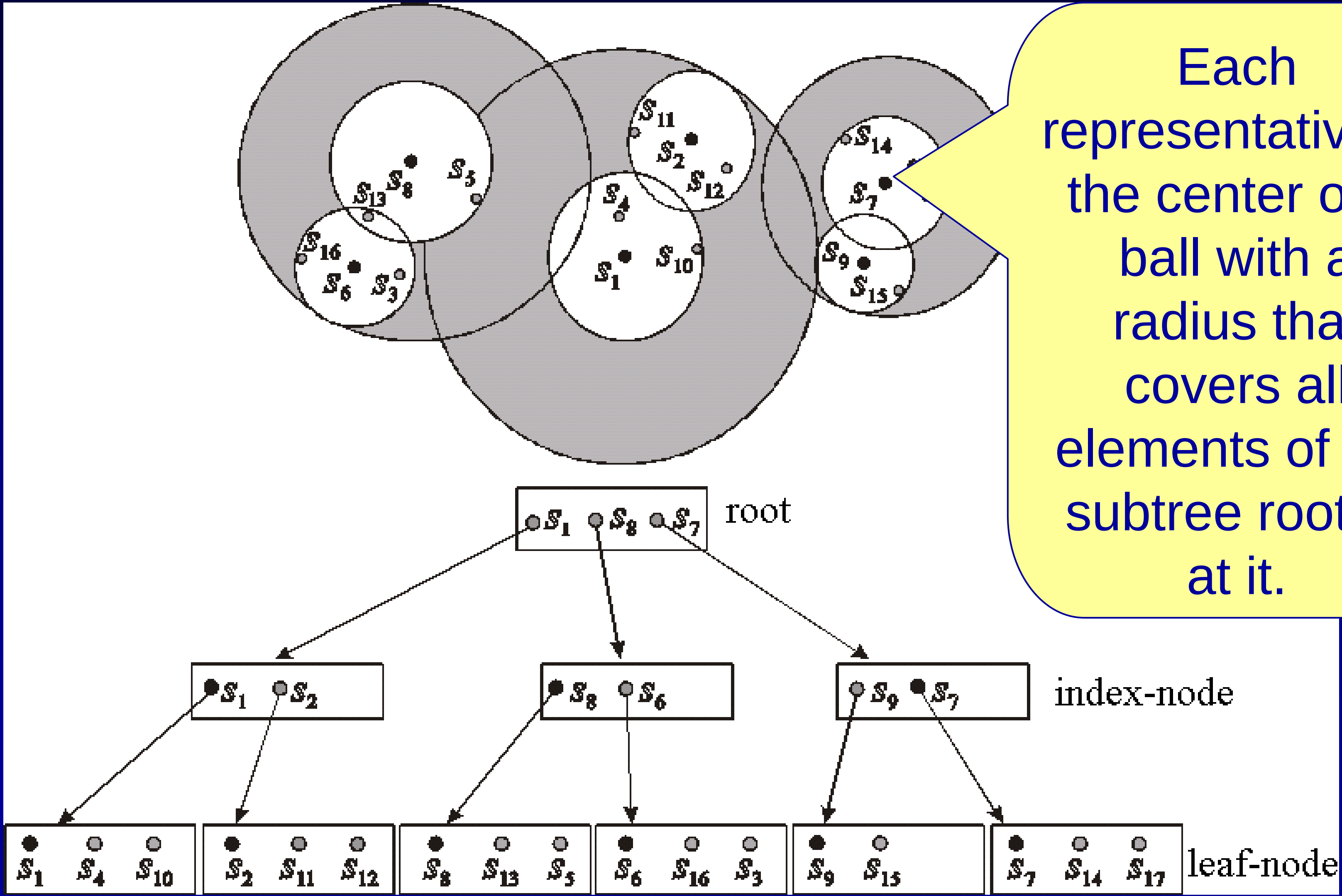
Metric Access Methods (MAM)

- Several MAM have been developed to speed up similarity query answering:
- Initially Static:
    - Vp-tree,
    - MVP-tree
    - GH-tree,
    - GNAT.
- Dynamic
    - M-tree
    - **Slim-Tree**
    - DBM-tree

Metric Access Methods

- **Slim-Tree**
  - balanced and dynamic hierarchical tree structure, with the elements stored in the leaves
  - elements are grouped around representative elements, in order to cluster similar elements
  - grows bottom-up
  - fixed size disk pages, each page corresponding to a tree node;

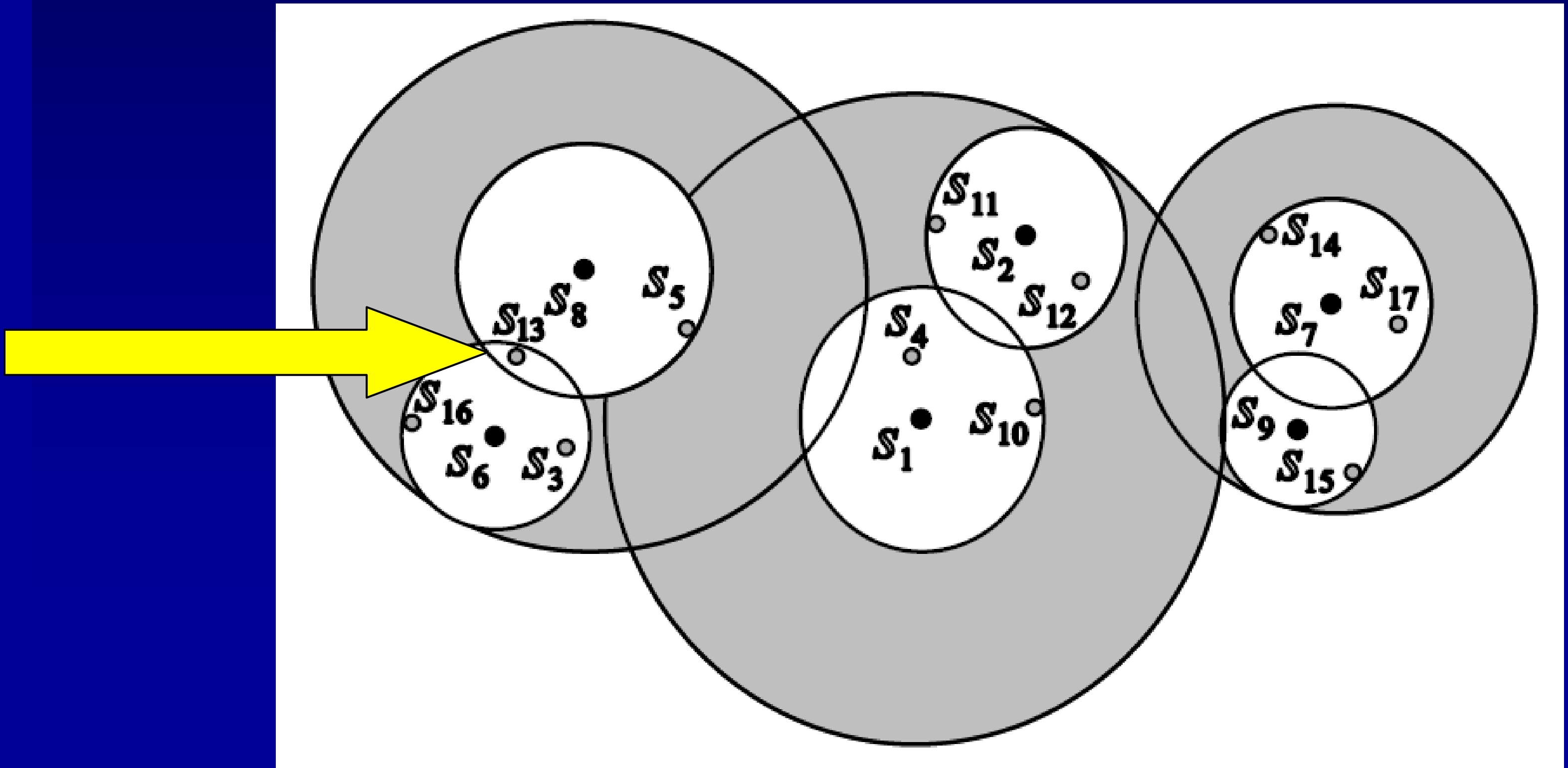
Metric Access Methods



Each representative is the center of a ball with a radius that covers all elements of the subtree rooted at it.

Overlap

- The division of the metric space of almost every dynamic MAM does not generate disjunct regions
  - it reduces the ability to prune subtrees.
- The degree of overlap directly affects the query performance of index structures



Overlap

- The well-known techniques to measure overlap of a pair of intersecting nodes cannot be used for metric data (absence of dimensionality)
- Overlap between two nodes: the number of elements covered by both regions divided by the number of elements in both subtrees.
- Fat-factor: quantify the degree of overlap between the nodes in a metric tree.

Our work

- In this work we proposed
  - an algorithm for the effective deletion of elements indexed in MAM
  - Push-pull, a new technique to optimize MAM
    - Removing and reinserting elements
  - Smart Push-pull: automatically defines a number of elements to be removed in each leaf node

Deletion Algorithm

- The development of dynamic MAM neglected the deletion and update of elements.

- Deletions
  - can force large tree reorganizations;
  - can be very expensive,

Deletion Algorithm

- The deletion operation is not even described in the great majority of MAM found in literature
- However, many applications handle complex data that evolve over time.
  - require removing or updating elements.
- Challenges of the deletion algorithm:
  - reduce the required reorganization
  - maintain the height-balancing property
    - not degenerate the structure
  - not increase node overlap

m-delete (mark-as-deleted)

- In almost every hierarchical MAM, the deletion of representative elements is performed just marking them as removed:
  - inappropriate when applications perform a large number of deletions.
    - increases the number of disk accesses and distance calculations
      - it forces comparisons with elements that do not exist anymore

Effective deletion algorithm

- enables effective deletion off any indexed element, maintaining the height-balancing of the structure
- uses a set of mechanism to reduce the reorganization caused in the structure
  - is based on importing the sibling subtrees when the Minimum Node Occupation (MNO) is violated;
  - uses mechanisms to enforce a reduced number of pages in the tree, improving the query performance.

Effective deletion algorithm

- If the *Node* MNO violation occurs deeper, the algorithm attempts to import an entry from a sibling node that will not violate the MNO:

  1st attempt: import an entry already covered by *Node*

  2n

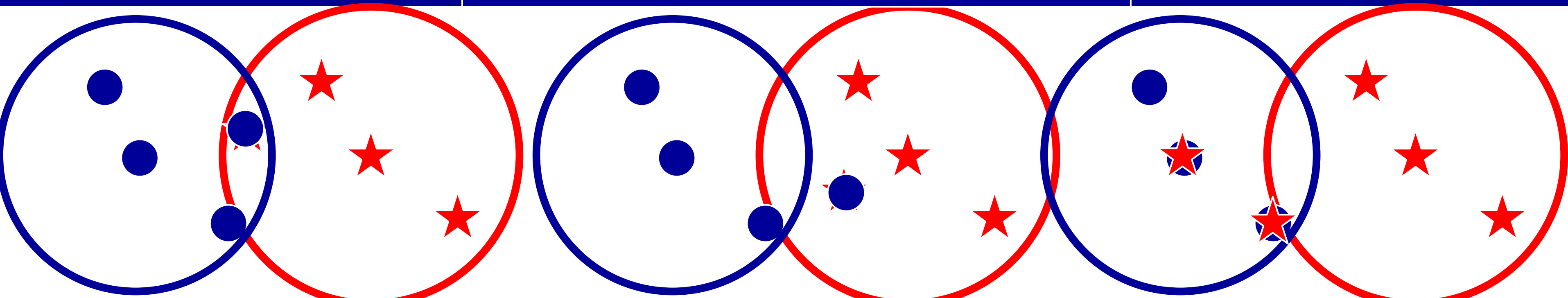
  this importation not increases the radius

  representative – increase the radius as little as possible

  3rd attempt: it is not possible import - export

- If the leaf node violates the MNO property, its remaining entries are reinserted
  - Empirical re

Ex.: Minimum Node Occupation = 3

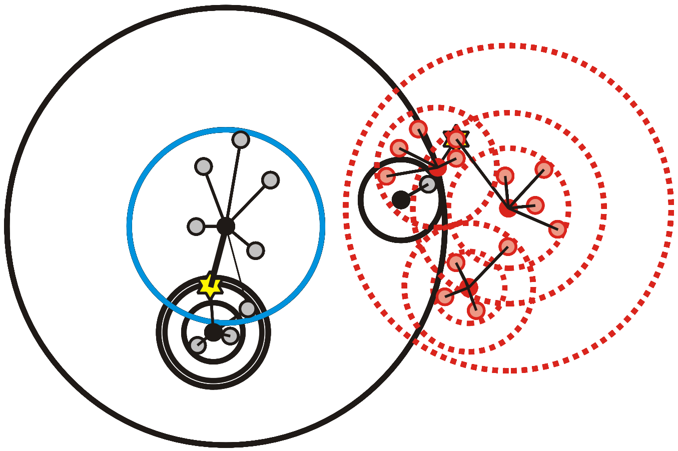


An Overlap Reduction and Optimization Technique for MAM

- We introduce a new optimization technique based on the effective deletion algorithm
- It searches for elements that are not close to the others on the node, thus increasing the covering radius.
- The idea is to remove several elements in the periphery of leaf nodes and reinsert them at once.

Slim-down

- When sibling leaf nodes overlap themselves, the Slim-down performs the “migration” of the farthest element of a node into a sibling node that also covers the element.
  - the overlap is reduced
- procedure is repeated until no element migrates between siblings nodes

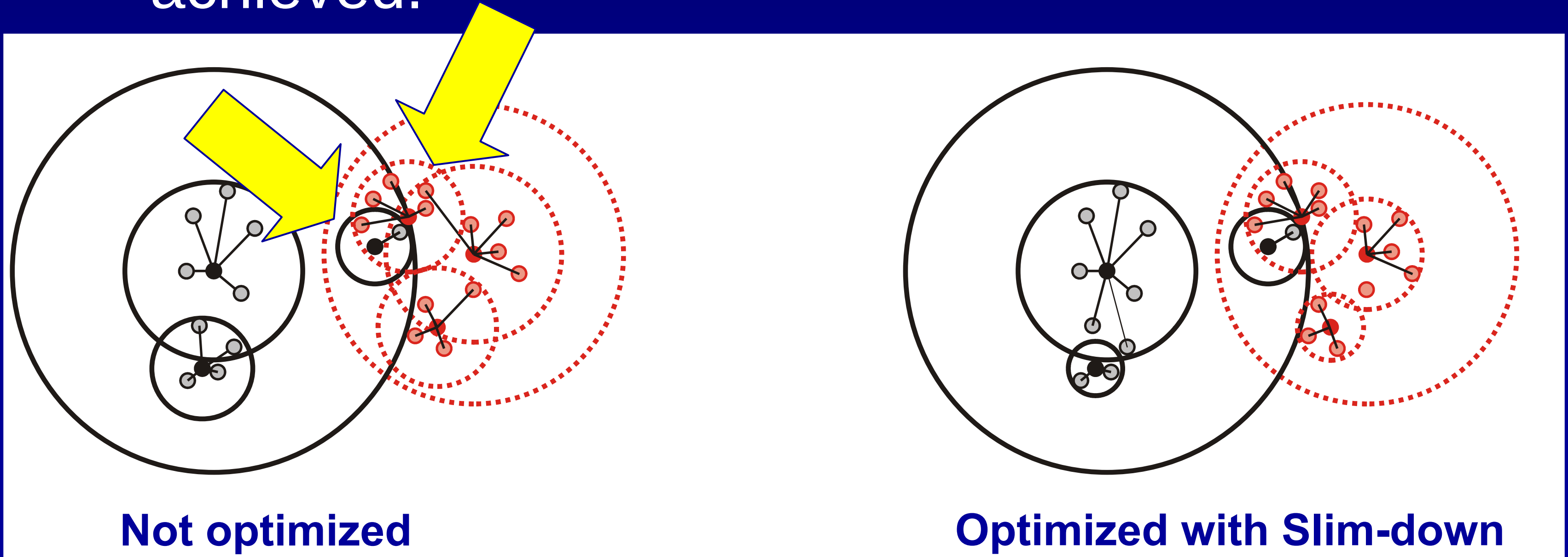


Not optimized

Optimized with Slim-down

Slim-down

- restricts the covering radius shrinking to the leaf nodes of the same subtree.
  - when two leaf nodes rooted at different index nodes overlap each other, no improvement is achieved.



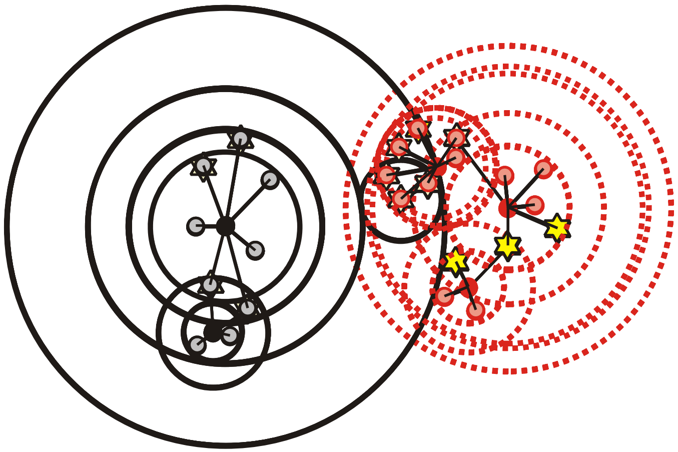
Not optimized

Optimized with Slim-down

Push-pull technique

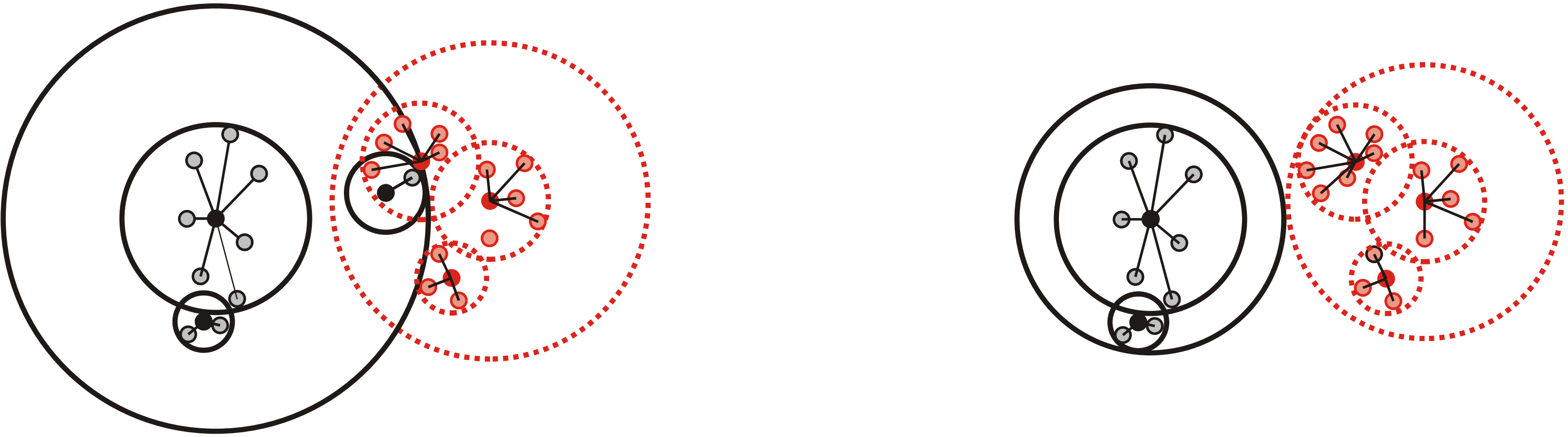
•The idea is to remove several elements in the periphery of leaf nodes and reinsert them at once.

- The elements selected to be removed are the farthest from their representatives
  - Reduction of leaf nodes covering radius
- Insertion operation tries first to reinsert elements in the nodes that do not increase their covering radius, or in nodes that require smaller radius increase
- Overall overlap reduction.



Slim-down vs. Push-pull

- Slim-Down: migration between leaf nodes linked to the same index node
- Push-pull: migration of elements between any leaf node



Optimized with Slim-down **Optimized with Push-pull**

Push-pull

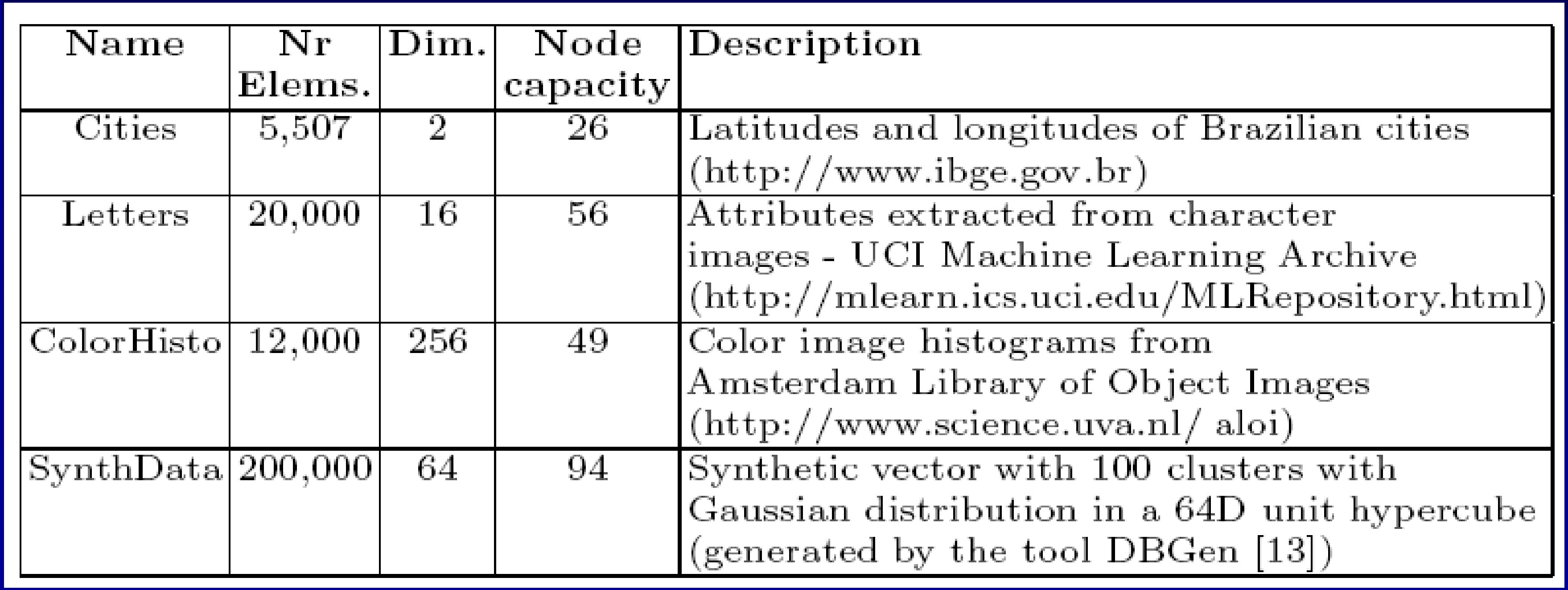
- Naïve Push-pull: users need to provide the quantity of elements to be removed from each node
- Experimental evaluation: the ideal percentage of elements to be removed vary from dataset to dataset, but it is limited by a saturation point
- Smart Push-pull: find automatically the quantity of elements to be removed in each node, based on statistics measured in the tree.

Smart Push-pull

- H : height of the tree.
- *Max_Occup* : node capacity
- AVG_{Node}: average number of accesses to retrieve every entry stored in each leaf node
  - – calculated during the computation of the tree's fat-factor

$$\#Obj_{Del} = \frac{AVG_{Node} - H}{AVG_{Max}} * Max\_Occup$$

Experiments
Datasets

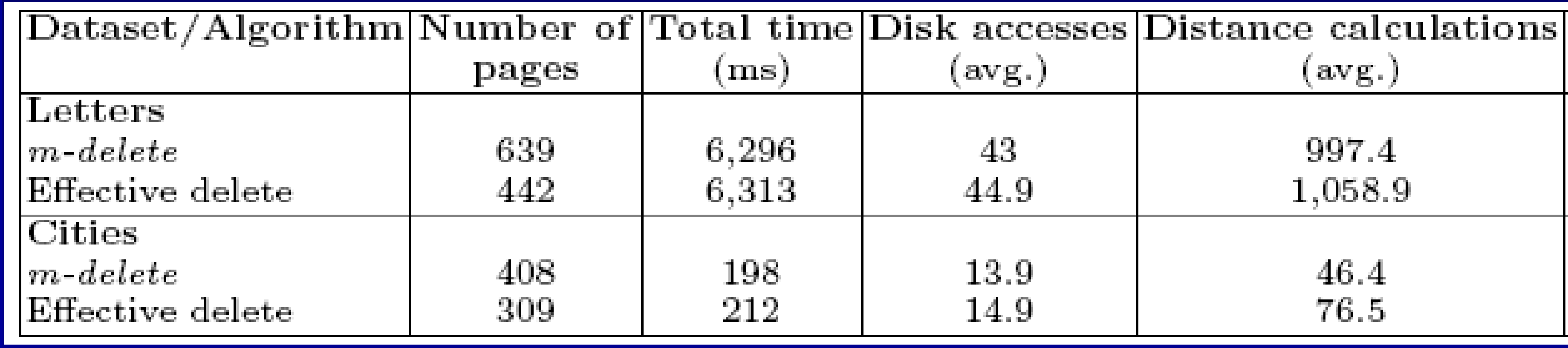


Name	Nr Elems.	Dim.	Node capacity	Description
Cities	5,507	2	26	Latitudes and longitudes of Brazilian cities (http://www.ibge.gov.br)
Letters	20,000	16	56	Attributes extracted from character images - UCI Machine Learning Archive (http://mlearn.ics.uci.edu/MLRepository.html)
ColorHisto	12,000	256	49	Color image histograms from Amsterdam Library of Object Images (http://www.science.uva.nl/ aloi)
SynthData	200,000	64	94	Synthetic vector with 100 clusters with Gaussian distribution in a 64D unit hypercube (generated by the tool DBGen [13])

Experiments
Effective Delete - execution time

Inserted 80% of the dataset, and then remoded half of elements

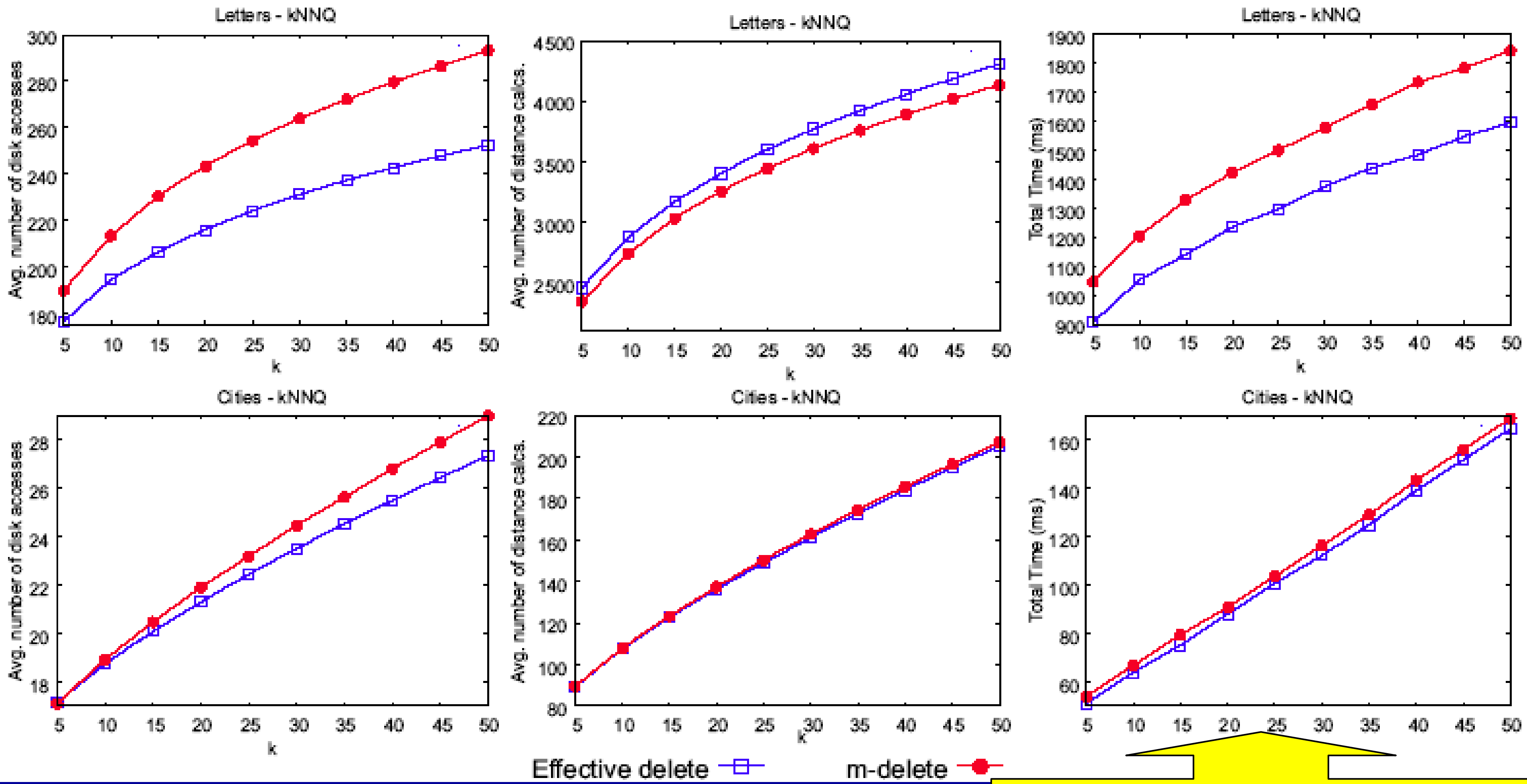
Very close execution time



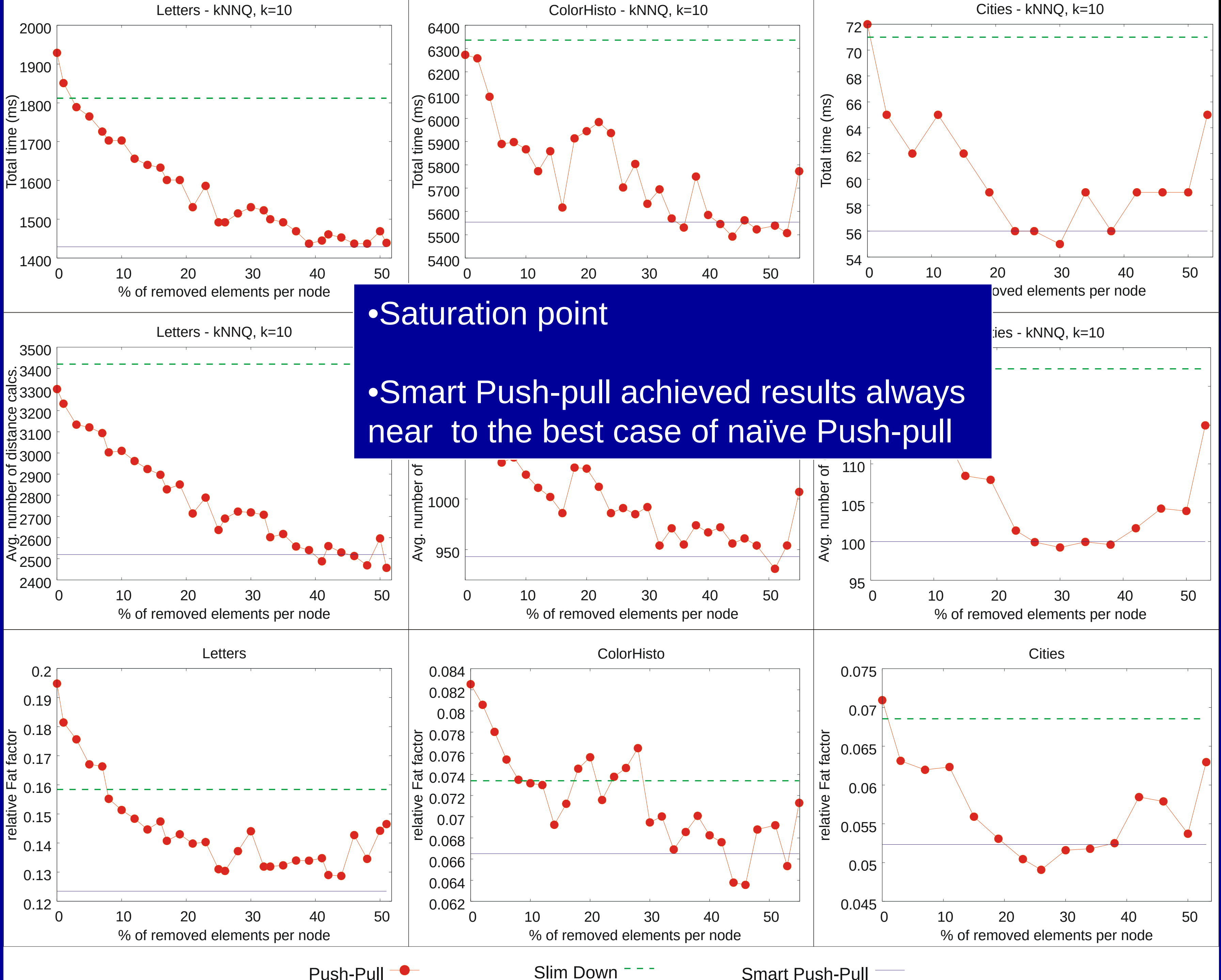
Dataset/Algorithm	Number of pages	Total time (ms)	Disk accesses (avg.)	Distance calculations (avg.)
Letters				
m-delete	639	6,296	43	997.4
Effective delete	442	6,313	44.9	1,058.9
Cities				
m-delete	408	198	13.9	46.4
Effective delete	309	212	14.9	76.5

Number of pages left in the tree after m-delete was more than 40% larger

Experiments
Effective Delete – query performance



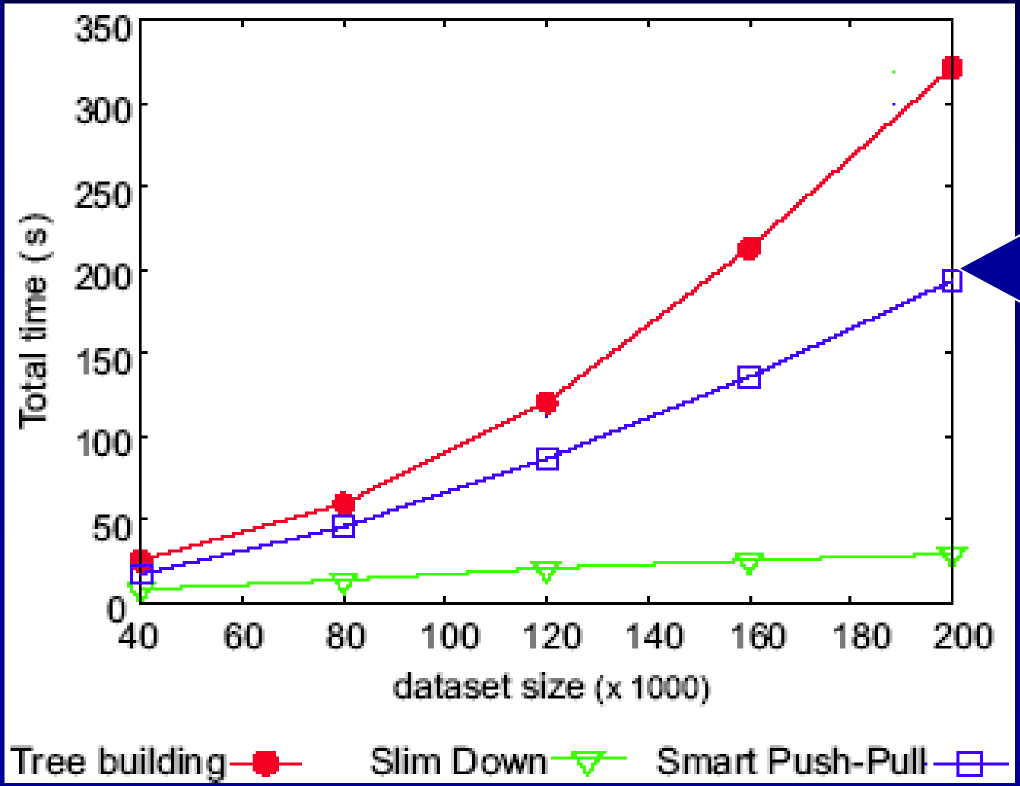
Queries after effective deletion were always faster



- Saturation point

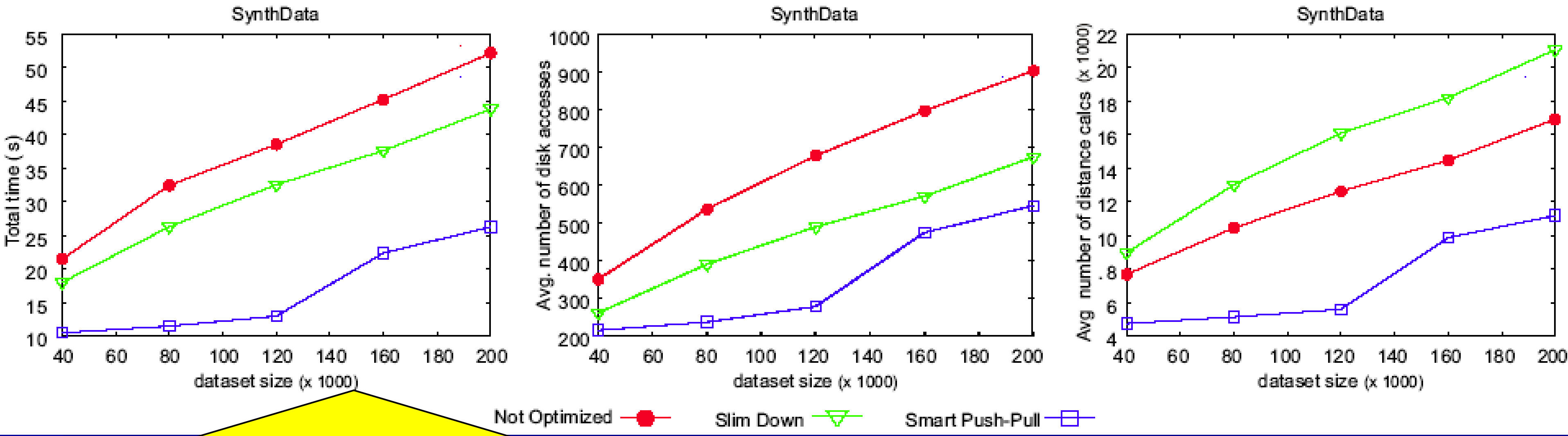
- Smart Push-pull achieved results always near to the best case of naïve Push-pull

Experiments
Smart Push-pull – execution time



Between 60% and 77%
of the time to build
the original tree

Experiments
Smart Push-pull - query performance



- 190% faster than not-optimized trees
- 150% faster than trees optimized with Slim-down

Conclusions

- This work proposed
  - an algorithm for the effective deletion of elements indexed in MAM, allowing to delete any element

queries were always faster after the application of the proposed algorithm, compared to the previous algorithm use

  - Push-pull, a new technique to optimize MAM
  - Smart Push-pull: automatically defines a number of elements to be removed in each leaf node

trees optimized by the Smart Push-pull tend to answer queries up to 150% faster than trees optimized by Slim-down.

A New Approach for Optimization of DynamicMetric Access Methods Using an Algorithm of Effective Deletion

Renato Bueno

Kaster

São Ca

rbueno@icmc.usp.br

Thanks

SSDBM 2008