R-Trees: A Dynamic Index Structure for Spatial Searching

Antonin Guttman
University of California
Berkeley

Presented by Benjamin Krogh

May 3, 2012

Agenda

1. Motivation
2. Structure
3. Operations
4. Performance Evaluation
5. Conclusion

Motivation

Problem

Spatial data objects often cover areas in a multidimensional space, which cannot be represented by point locations.
Frequent operation is to query the database for all objects within a certain distance of a point.

Prior methods:

- B-trees, do not work because the data is multidimensional.
- Quad trees does not account for paging on disk.
- K-D-B trees considers only point data.

Structure

- R-tree is a height-balanced tree.
- Data is represented as minimum bounding rectangles in multiple dimensions.
- Leaf nodes contains pointers to index records.

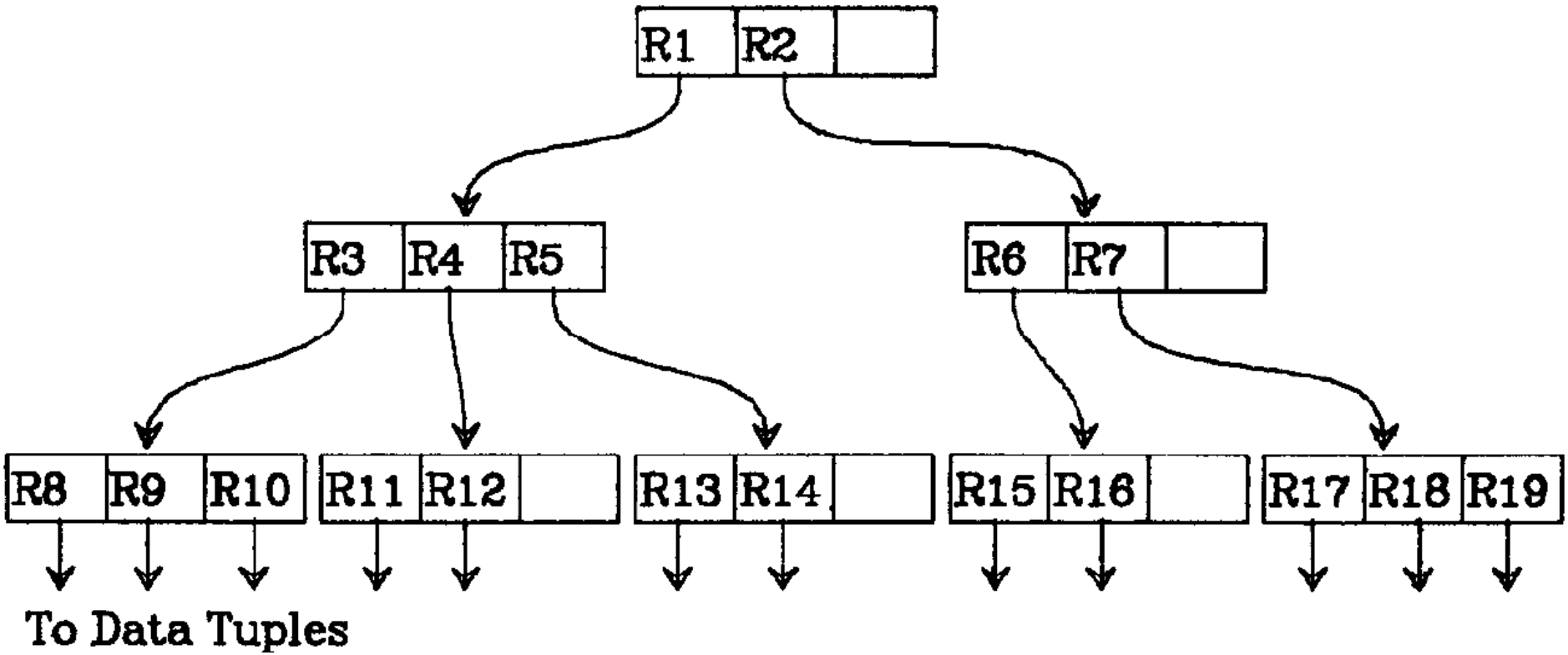
Leaf Entry Structure

$(I, tuple-identifier)$
Where I is the minimum bounding rectangle of the record.

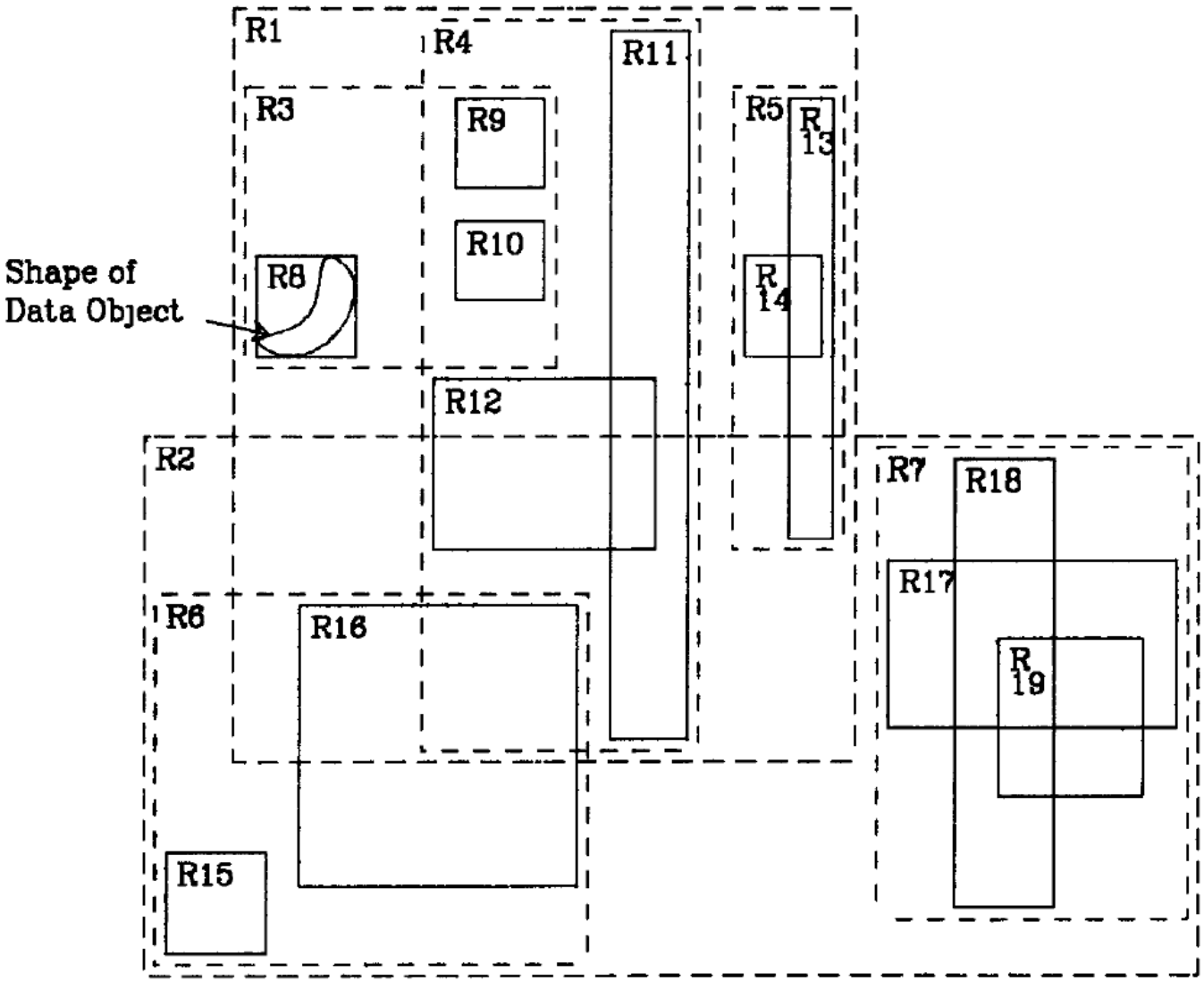
Node Entry Structure

$(I, node-pointer)$
Where I is the minimum bounding rectangle of all children.

R-tree Example



R-tree Example - cont



Structure - Revisited

R-tree Parameters

- M denotes the maximum node capacity
- m denotes the minimum fill.

Properties of an R-tree

1. Every node contains between m and M nodes, unless it is the root.
2. The root node has at least two nodes, unless it is a leaf.
3. The tree is balanced, i.e., all leaves are on the same level.

Searching

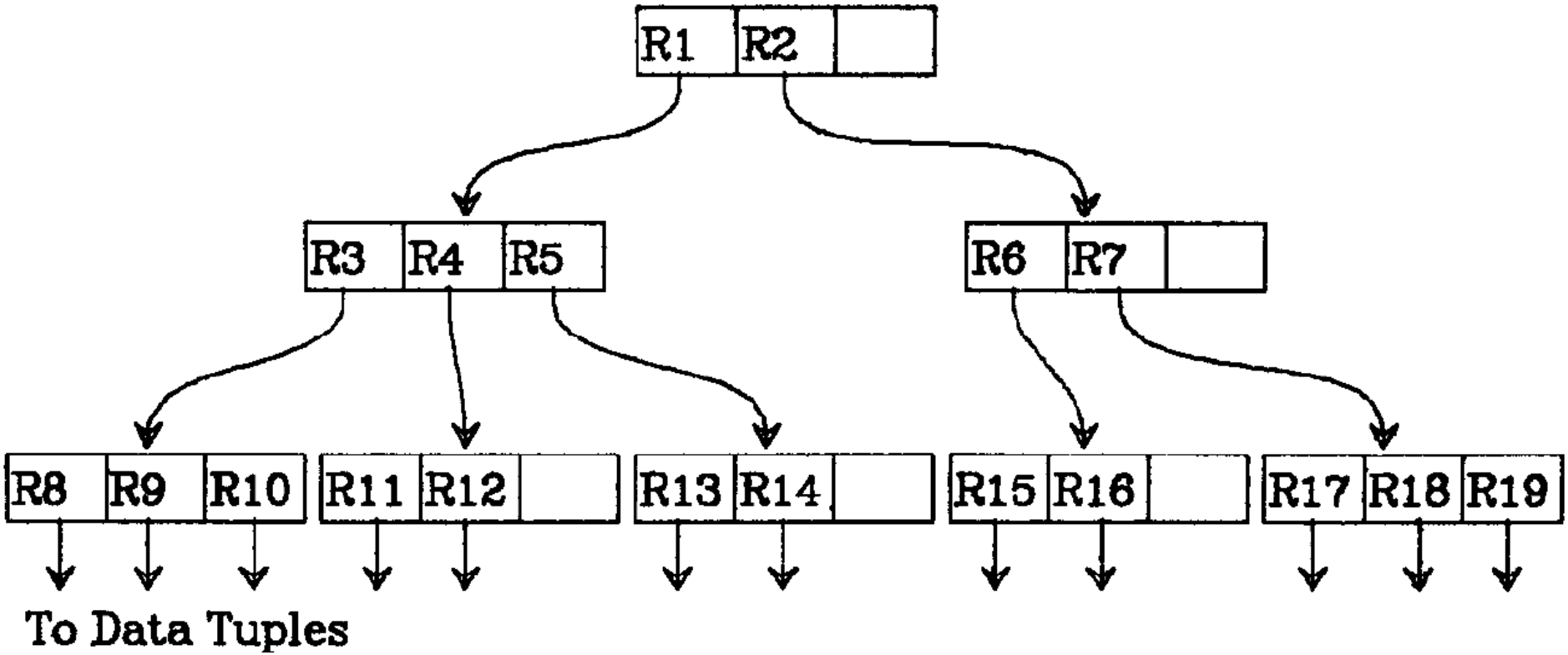
Search

Given a search rectangle S, find all the leaf nodes with an MBR that overlap with S.

1. Set T to be the root node.
2. If T is non-leaf, recursively call `search` on each child node with an MBR overlapping S.
3. If T is leaf, report those entries that overlap with S.

Searching - cont

$S = MBR(R12)$



Insertion

Insert

Insert new entry E.

1. Invoke `ChooseLeaf` to find the best leaf-node L for insertion.

Insertion

Insert

Insert new entry E.

1. Invoke `ChooseLeaf` to find the best leaf-node L for insertion.
2. If L has space for another entry, install E. Otherwise invoke `SplitNode` on L to obtain new L and LL.

Insertion

Insert

Insert new entry E.

1. Invoke `ChooseLeaf` to find the best leaf-node L for insertion.
2. If L has space for another entry, install E. Otherwise invoke `SplitNode` on L to obtain new L and LL.
3. Invoke `AdjustTree` on L, and LL.

Insertion

Insert

Insert new entry E.

1. Invoke `ChooseLeaf` to find the best leaf-node L for insertion.
2. If L has space for another entry, install E. Otherwise invoke `SplitNode` on L to obtain new L and LL.
3. Invoke `AdjustTree` on L, and LL.
4. If a node split caused the root node to split, create a new root node containing the resulting nodes.

Insertion - 2

AdjustTree

Adjust the covering rectangle of node L and propagate up until the root is reached. Propagate node splits as necessary.

1. Set N to L.

Insertion - 2

AdjustTree

Adjust the covering rectangle of node L and propagate up until the root is reached. Propagate node splits as necessary.

1. Set N to L.
2. If N is root, stop.

Insertion - 2

AdjustTree

Adjust the covering rectangle of node L and propagate up until the root is reached. Propagate node splits as necessary.

1. Set N to L.
2. If N is root, stop.
3. Set P to be the parent of N. Adjust the MBR for N in P.

Insertion - 2

AdjustTree

Adjust the covering rectangle of node L and propagate up until the root is reached. Propagate node splits as necessary.

1. Set N to L.
2. If N is root, stop.
3. Set P to be the parent of N. Adjust the MBR for N in P.
4. If N has a partner NN, create an entry for NN in P. If P has not enough space for NN, invoke *SplitNode* on P creating new P and PP nodes.

Insertion - 2

AdjustTree

Adjust the covering rectangle of node L and propagate up until the root is reached. Propagate node splits as necessary.

1. Set N to L.
2. If N is root, stop.
3. Set P to be the parent of N. Adjust the MBR for N in P.
4. If N has a partner NN, create an entry for NN in P. If P has not enough space for NN, invoke *SplitNode* on P creating new P and PP nodes.
5. Set N to P and NN to PP. Go to step 2.

Insertion - 3

SplitNode

Given a full node L, create two new nodes N and NN, such that the total area of the MBRs is minimized.

- Quadratic Cost Algorithm
- Linear Cost Algorithm

Both follow the same recipe:

1. Use a `PickSeed` function, to select two seeds.
2. Use a `PickNext` to find and add the next node to either of the groups.

Deletion

Delete

Delete entry E from an R-tree.

1. Invoke `FindLeaf` to locate the leaf L containing E.
2. Remove E from L.
3. Invoke `CondenseTree` on L.
4. If the root node has only one child, make the child the new root.

Deletion - 2

CondenseTree

Eliminate leaf-node L if it has too few children and relocate its children. Adjust all MBRs on the path to the root.

1. Set N to L. Set Q to be an empty set of nodes.

Deletion - 2

CondenseTree

Eliminate leaf-node L if it has too few children and relocate its children. Adjust all MBRs on the path to the root.

1. Set N to L. Set Q to be an empty set of nodes.
2. If N is root, go to step 6. Otherwise set P to be the parent of N.

Deletion - 2

CondenseTree

Eliminate leaf-node L if it has too few children and relocate its children. Adjust all MBRs on the path to the root.

1. Set N to L. Set Q to be an empty set of nodes.
2. If N is root, go to step 6. Otherwise set P to be the parent of N.
3. If N is under-full, delete N from P, and add N to the set Q.

Deletion - 2

CondenseTree

Eliminate leaf-node L if it has too few children and relocate its children. Adjust all MBRs on the path to the root.

1. Set N to L. Set Q to be an empty set of nodes.
2. If N is root, go to step 6. Otherwise set P to be the parent of N.
3. If N is under-full, delete N from P, and add N to the set Q.
4. If N is not under-full, update its MBR in P.

Deletion - 2

CondenseTree

Eliminate leaf-node L if it has too few children and relocate its children. Adjust all MBRs on the path to the root.

1. Set N to L. Set Q to be an empty set of nodes.
2. If N is root, go to step 6. Otherwise set P to be the parent of N.
3. If N is under-full, delete N from P, and add N to the set Q.
4. If N is not under-full, update its MBR in P.
5. Set N to P, go to step 2.

Deletion - 2

CondenseTree

Eliminate leaf-node L if it has too few children and relocate its children. Adjust all MBRs on the path to the root.

1. Set N to L. Set Q to be an empty set of nodes.
2. If N is root, go to step 6. Otherwise set P to be the parent of N.
3. If N is under-full, delete N from P, and add N to the set Q.
4. If N is not under-full, update its MBR in P.
5. Set N to P, go to step 2.
6. Reinsert nodes from the set Q, such that all nodes will be on the same level as before.

Performance Evaluation

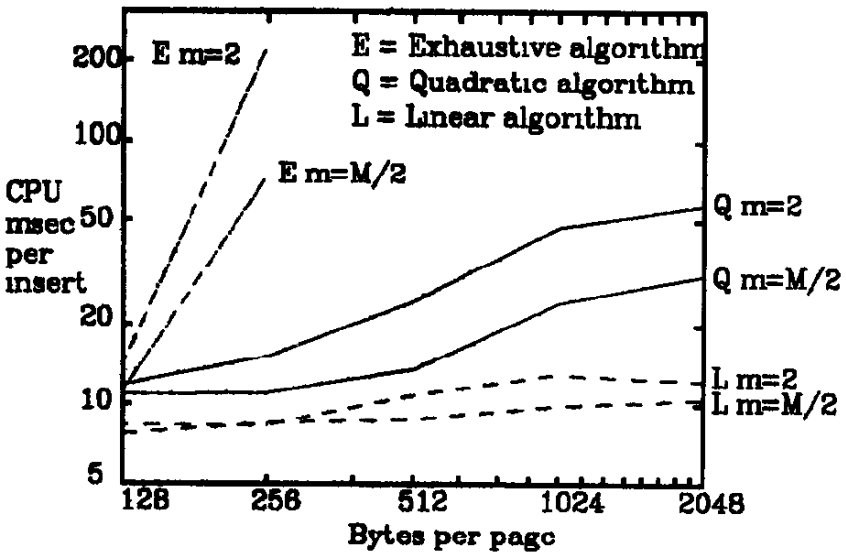


Figure: CPU cost of inserting records

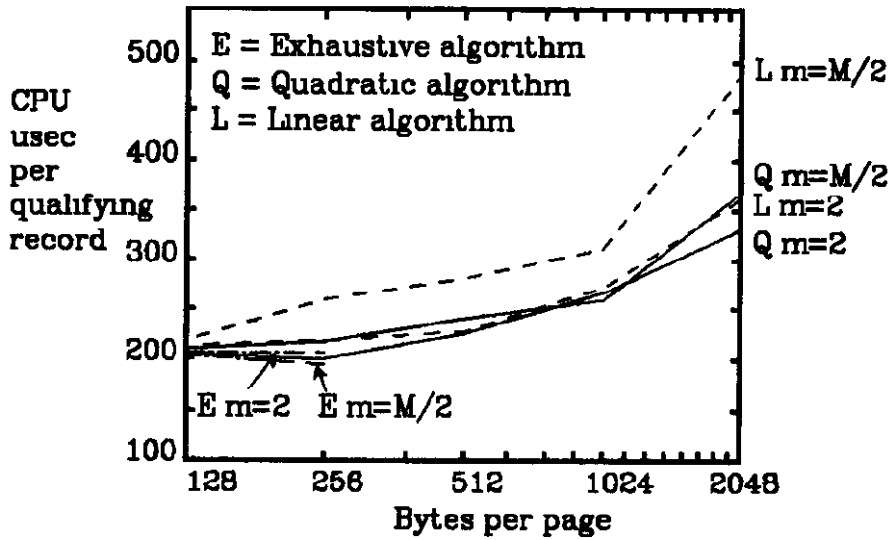
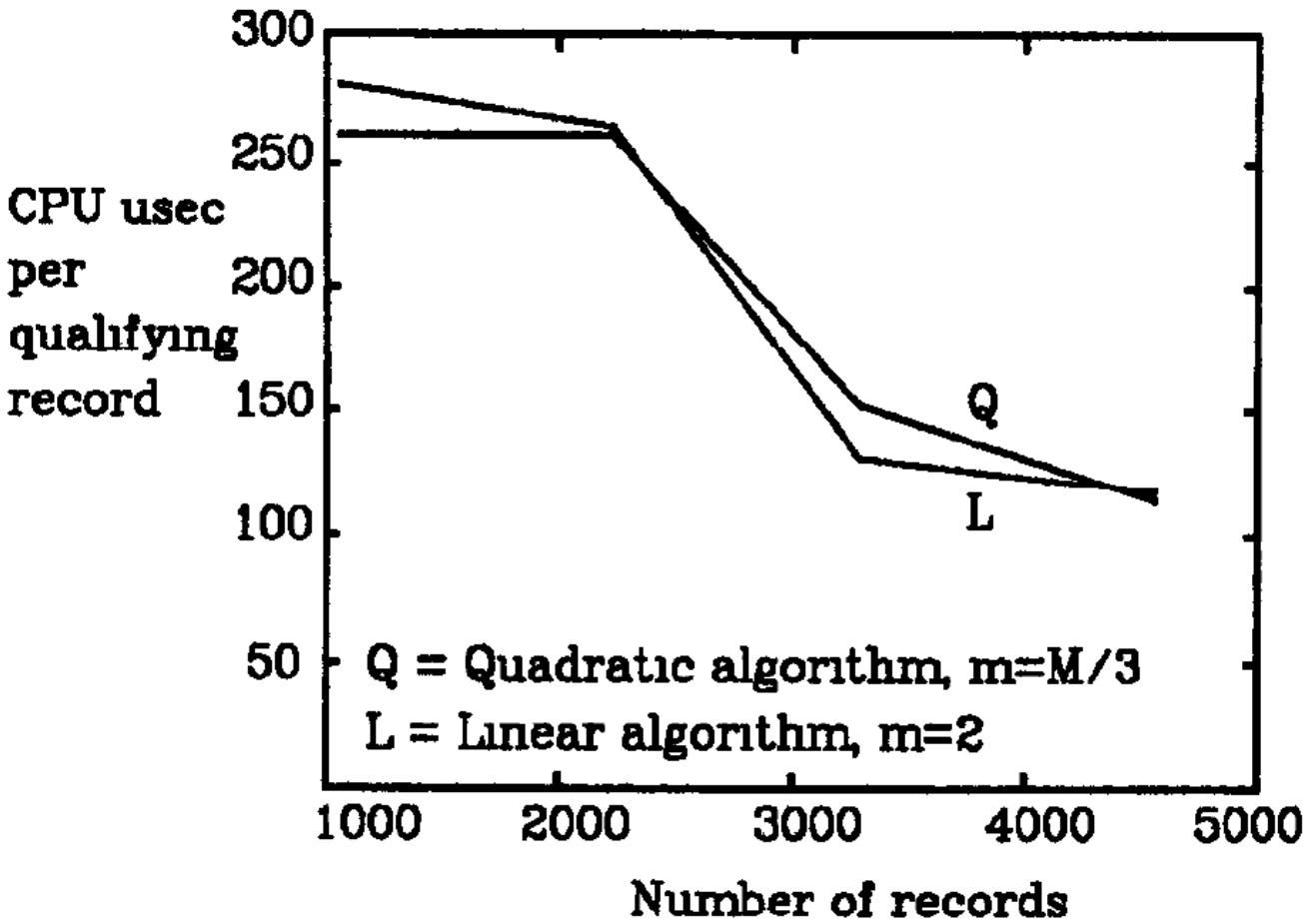


Figure: Search performance - CPU cost

Performance Evaluation - Cont



Conclusion

- Useful dynamic structure for indexing spatial objects.
- Disk page size, dictates values of M that yields good performance.
- Using smaller values of M should enable good main-memory indexes.
- Linear node-split algorithm proved to be just as good as more expensive approaches.
  - Slightly worse quality of node split.
  - Performance not affected noticeably.

Critique

Strong Points

- Massive impact on spatial databases (∼5.7K citations)
- Generally thought-through, leaving few direct improvements.
- Very to-the-point, without diverting from main topic.
- Completely dynamic data-structure that needs few parameters.

Weak Points

- Comparison with prior methods is omitted.
- The test data-set is very small, by today's standards.
- Claims of producing good performance for in-memory indexes not supported.