R-trees

Algorithms and Data Structures
for Database Systems

Jürgen Treml

2005-06-08

Overview

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

Overview – Where are we?

1. **Introduction, Motivation**
2. R-tree Index Structure – Overview
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

Introduction, Motivation

- Importance of effectively storing and indexing spatial data (CAD, VR, Image Processing, Cartography, …)
- One-dimensional indexes not suitable
- Strong limitations with most spatial structures (e.g. point data only, not dynamic, performance with paged memory, …)

Introduction, Motivation

→ Antonin Guttman, 1984

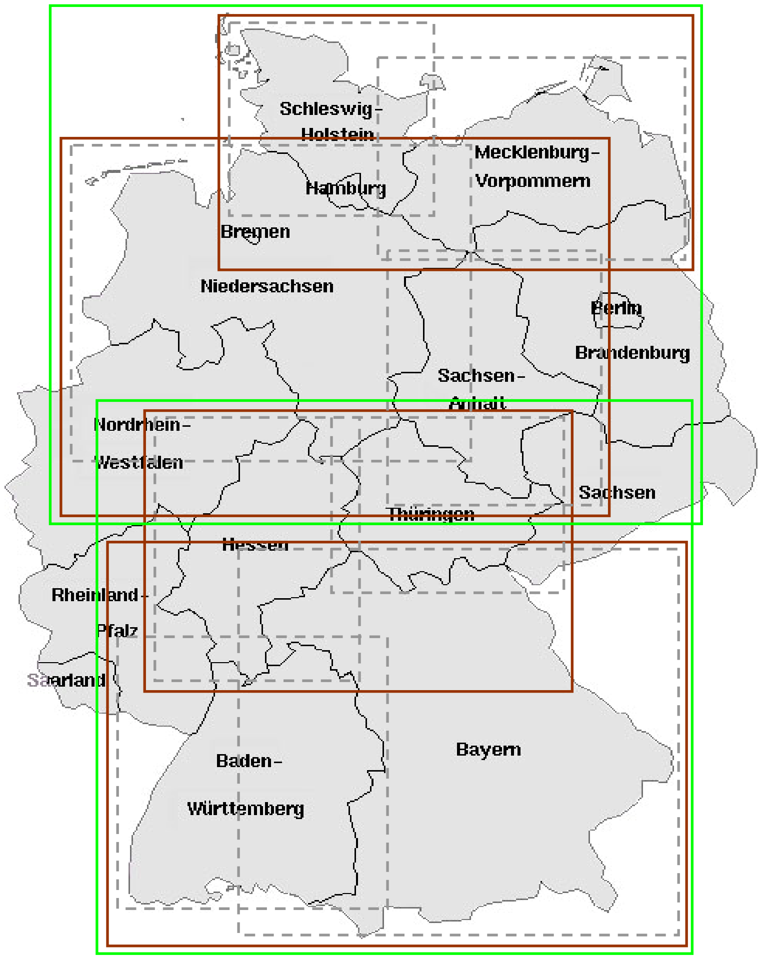
- "A Dynamic Index Structure For Spatial Searching"
- Based on B-trees

→ **Region-trees (R-trees)**

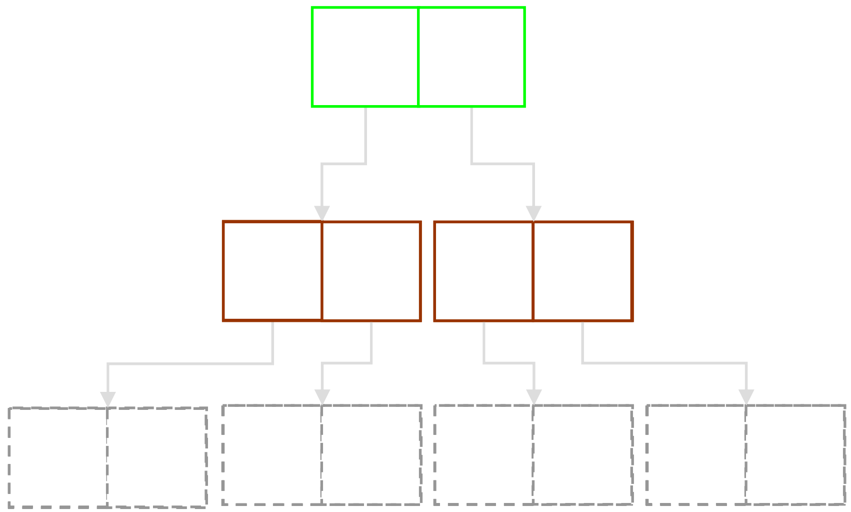
Overview – Where are we?

1. Introduction, Motivation
2. **R-tree Index Structure – Overview**
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

R-tree Index Structure



Data objects are re-presented by their MBRs



R-tree Index Structure

Formal Description

- Structure consisting of (regular) nodes containing tuples

$$(I_n, CP)$$

- At the lowest level: leaf-nodes with tuples

$$(I_n, TID)$$

- MBR (minimum bounding rectangle)

$$I_n = (I_0, I_1, ..., I_n)$$

R-tree Index Structure

Important Parameters

- Maximum number of elements per node

$$M$$

- Minimum number of elements per node

$$m \leq \frac{M}{2}$$

- Height of an R-tree

$$\lceil \log_m N \rceil - 1$$

R-tree Index Structure

Important Restrictions

- All leaf-nodes are on the same level
- Root has at least two children (unless it is a leaf)

Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. **Algorithms on R-trees**
   - **Searching**
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

Algorithms on R-trees: Search

- Similar to B-tree search
- Quite easy & straight forward
(Traverse the whole tree starting at the root node)

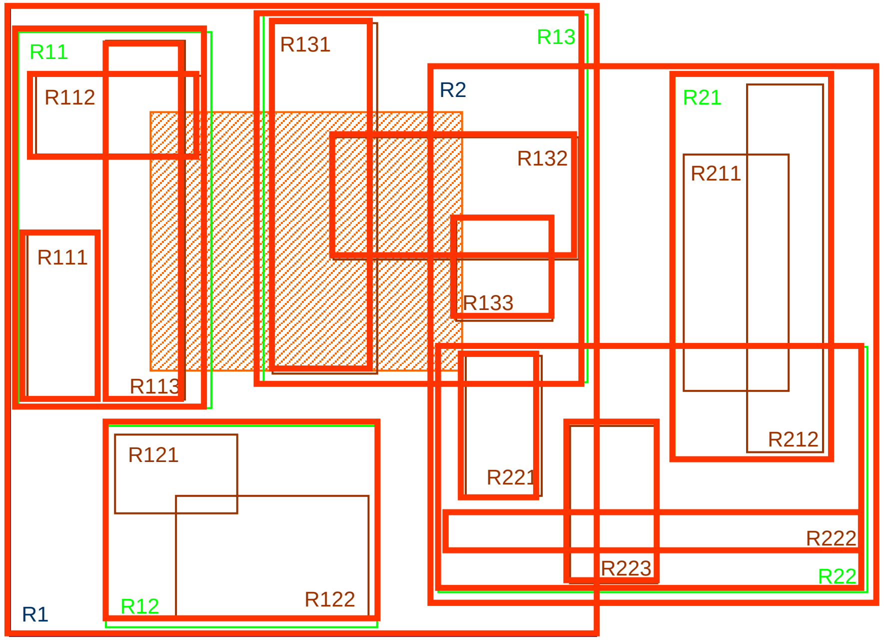
- No guarantee on good worst-case performance!
(Possible overlapping of rectangles of entries
within a single node!)

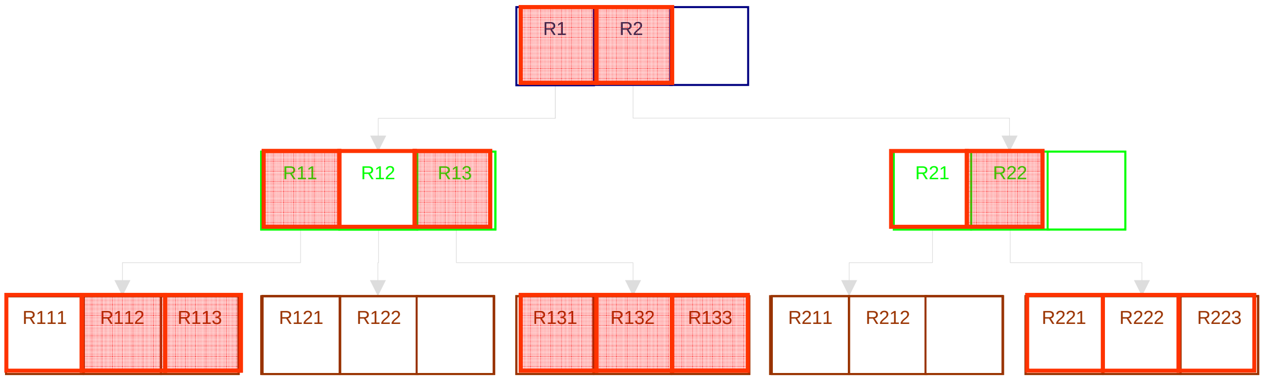
Algorithms on R-trees: Search

Short Description:

- For all entries in a non-leaf node:
  Check if overlapping
  → If yes: check node pointed to by this entry
- If node is a leaf-node:
  Check all entries if overlapping the search object
  → If yes: entry is a qualifying record!

Algorithms on R-trees: Search





Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. **Algorithms on R-trees**
   - Searching
   - **Insert / Delete**
   - Node Splitting
   - Updates & other Operations
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

Algorithms on R-trees: Insert

- Again: Similar to corresponding B-tree operation
- Basically consist of 3 parts:
  1. **CHOOSELEAF**
     (Find place to insert new object)
  2. **INSERT**
     (Insert the new object)
  3. **ADJUSTTREE**
     (Adjusting preceding nodes)

Algorithms on R-trees: Insert

Short Description:

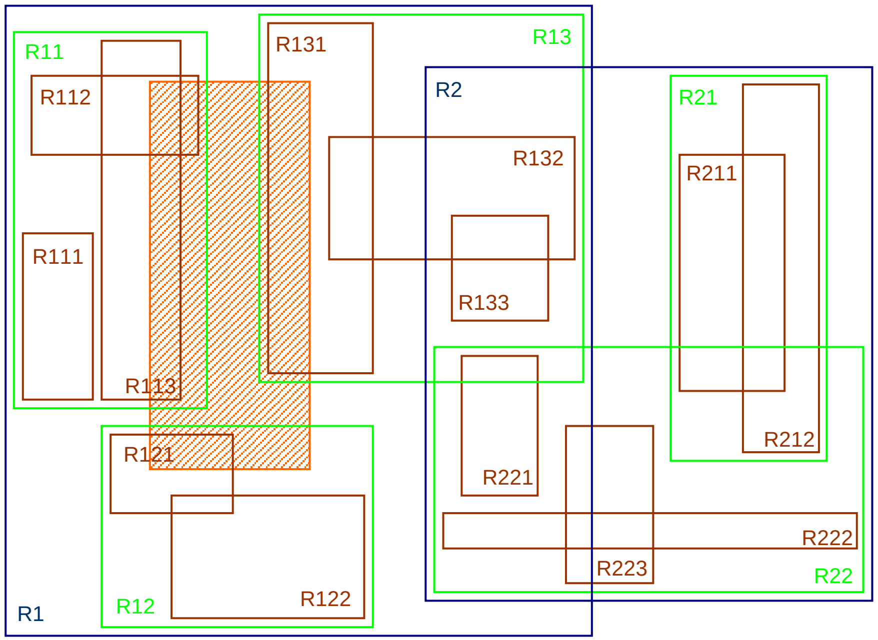
- **CHOOSELEAF**:
  Start with root → Run through all nodes:
  Find the one which would have to be least enlarged to include given object!
- **INSERT**:
  Check if room for another entry
  →Insert new entry directly OR after calling SPLITNODE (in case of no room)

Algorithms on R-trees: Insert

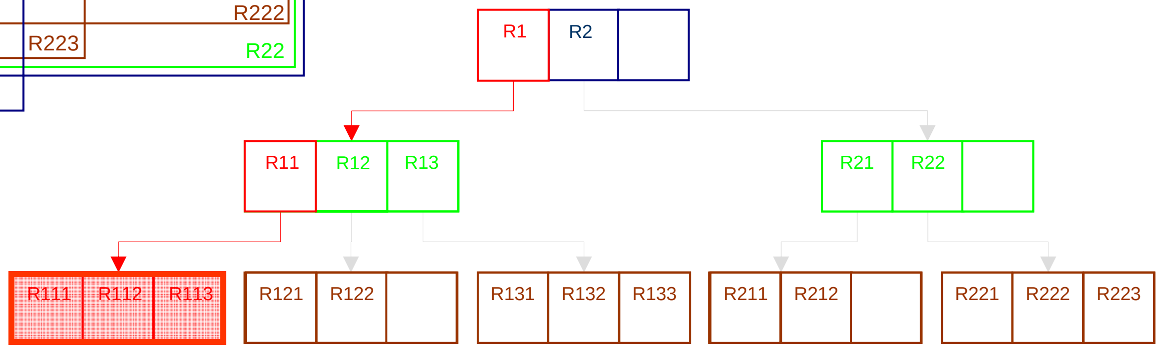
Short Description:

- **CHOOSELEAF (…)**
- **INSERT (…)**
- **ADJUSTTREE:**
  Ascend from node with new entry:
  Adjust all MBRs!
  In case of node split:
  Add new entry to parent node
  (If no room in parent node, invoke SPLITNODE again)
  Propagate upwards till root is reached!

Algorithms on R-trees: Insert

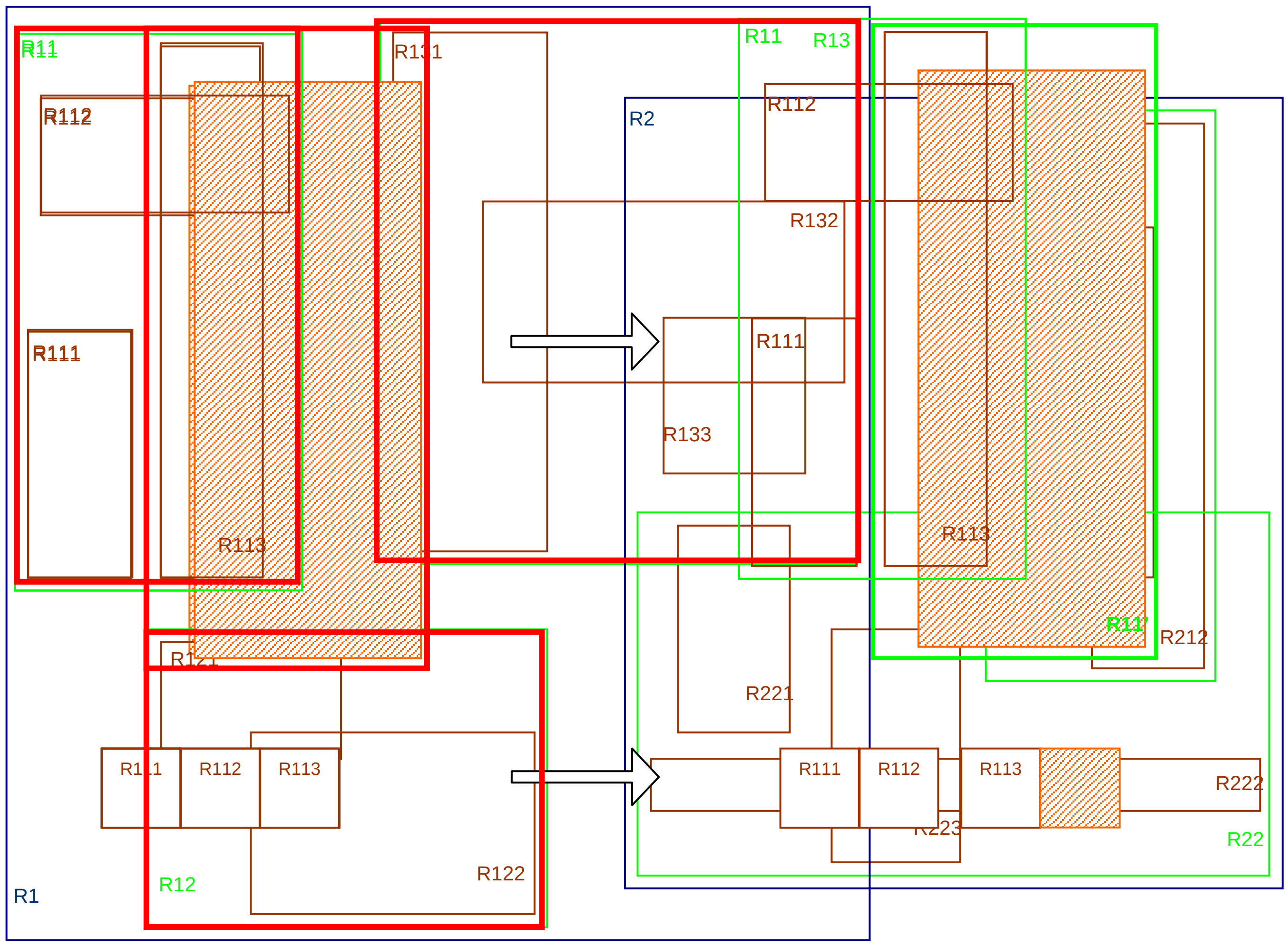


Find node to insert the new object into:

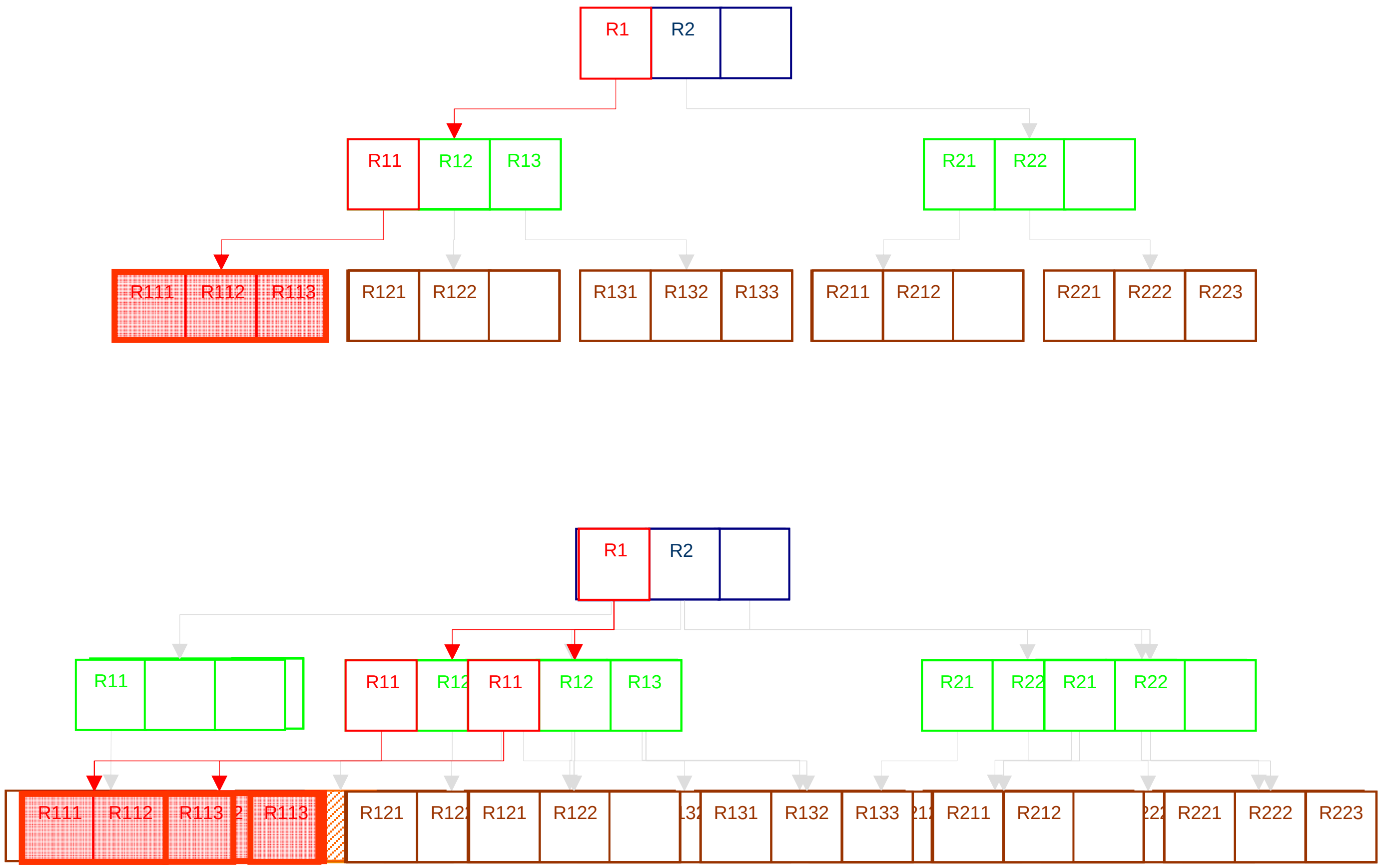


Leaf is full! Overflow!

Algorithms on R-trees: Insert



Algorithms on R-trees: Insert



Algorithms on R-trees: Delete

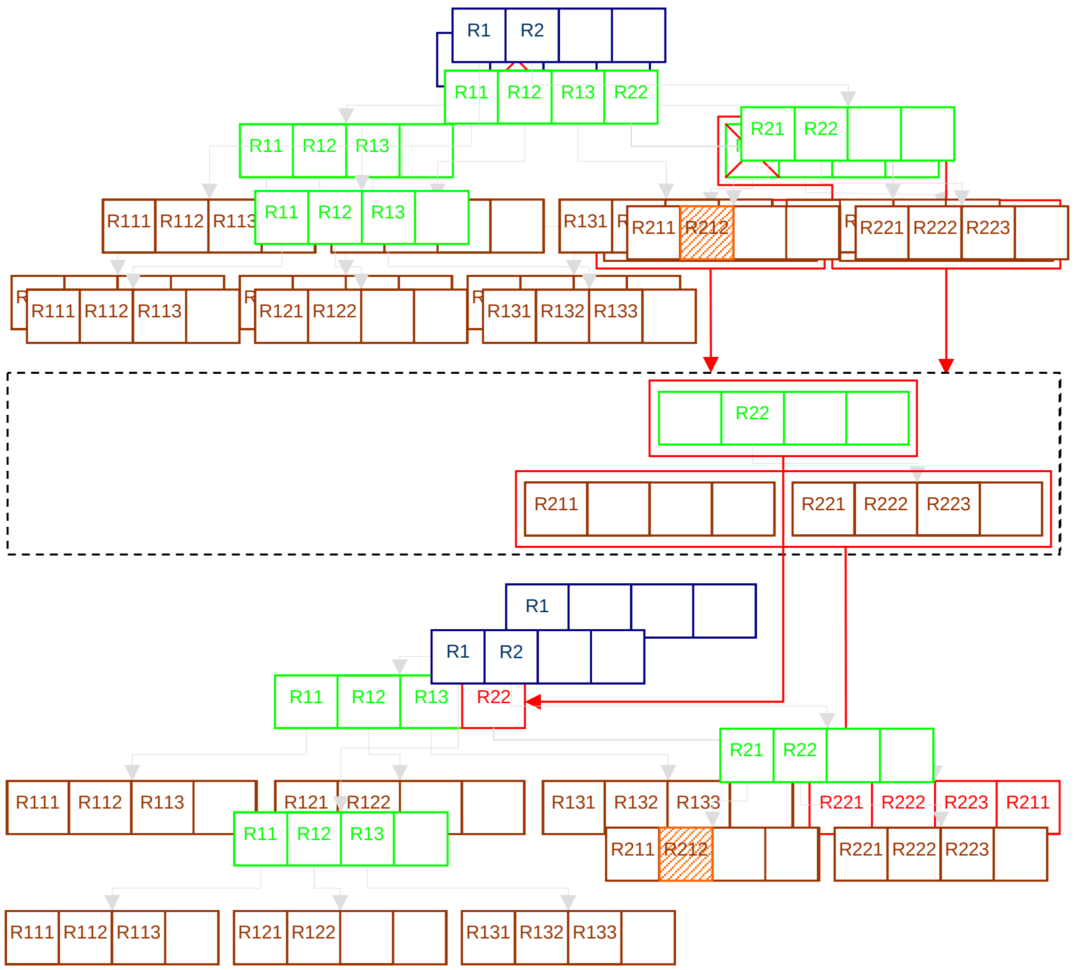
- NOT similar to B-tree DELETE
  (Treatment of underflows)
- B-tree:
  Merge under-full node with “neighbor”
- R-tree:
  Delete under-full node and re-insert
- Why?
  Re-use of INSERT routine
  Incrementally refines spatial structure

Algorithms on R-trees: Delete

"Very" Short Description

- **FINDLEAF:**
  Locate the leaf that contains object to be deleted
- **DELETE:**
  Delete entry from node
- **CONDENSETREE:**
  Delete and re-insert node if under-full (and in the following all resulting under-full nodes)

Algorithms on R-trees: Delete



Overview – Where are we?

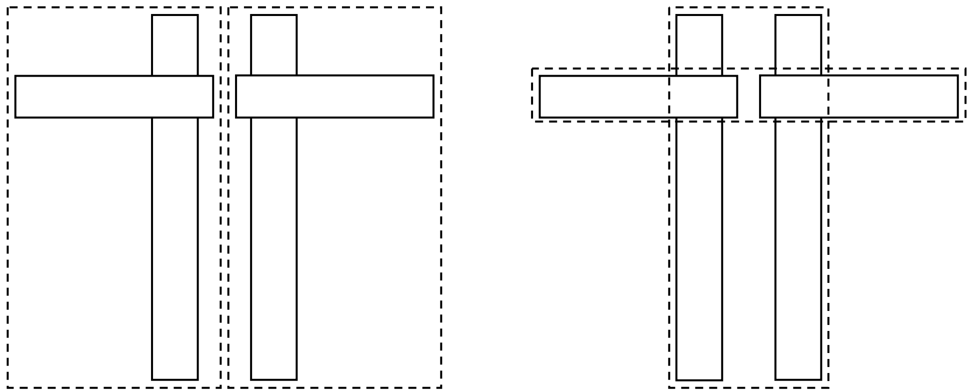
1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. **Algorithms on R-trees**
   - Searching
   - Insert / Delete
   - **Node Splitting**
   - Updates & other Operations
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

Algorithms on R-trees: Splitting

- Splitting of nodes necessary after underflow or overflow (as a result of a delete or insert operation)
- Ultimate goal: Minimize the resulting node's MBRs
- Secondary aim: Do it fast! ;-)
- 3 Implementations by Guttmann: Exhaustive, Quadratic-cost, Linear-cost

Algorithms on R-trees: Splitting

- Minimal resulting MBRs:



Bad split　　　　　　Good split

- **Exhaustive Approach:**
  Simply try all possible split-ups!

Algorithms on R-trees: Splitting

- **Quadratic-cost Algorithm**:
  Pick the 2 out of M entries that would consume the most space if put together.
  → Put one in each group
  For all remaining entries: Pick the one that would make the biggest difference in area when put to one of the two groups
  → Add it to the one with the least difference
  Finished when all entries are put in either group!
- Quadratic in M, linear in the number of dimensions

Algorithms on R-trees: Splitting

- **Linear-cost Implementation**:
  Basically the same as quadratic-cost algorithm
  Differences:
  First pair is picked by finding the two rectangles with the greatest normalized separation in any dimension.
  Remaining pairs are selected randomly.
- Linear in M as well as in the number of dimensions

Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. **Algorithms on R-trees**
   - Searching
   - Insert / Delete
   - Node Splitting
   - **Updates & other Operations**
4. Performance, Benchmarks
5. R-tree Modifications
6. Conclusion

Algorithms on R-trees: Other Ops

- Modified search algorithm
- Key search / search for specific entries

- Range deletion

- R-trees are quite as well extensible and efficient as B-trees for the matter of algorithms

Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. **Performance, Benchmarks**
5. R-tree Modifications
6. Conclusion

Performance of R-trees

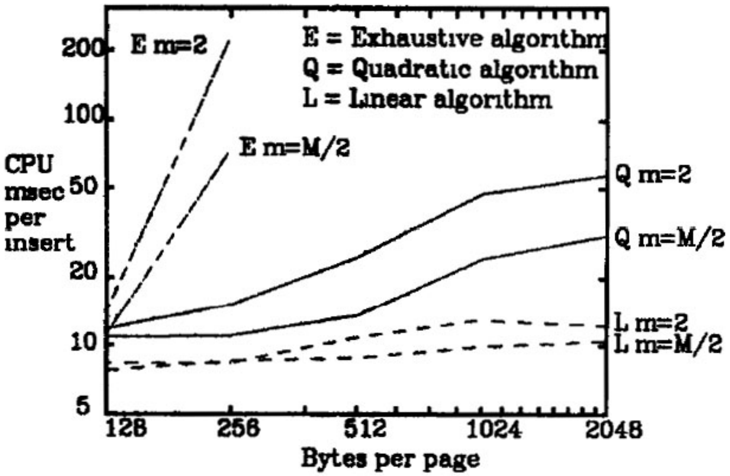
Why do Benchmarking?

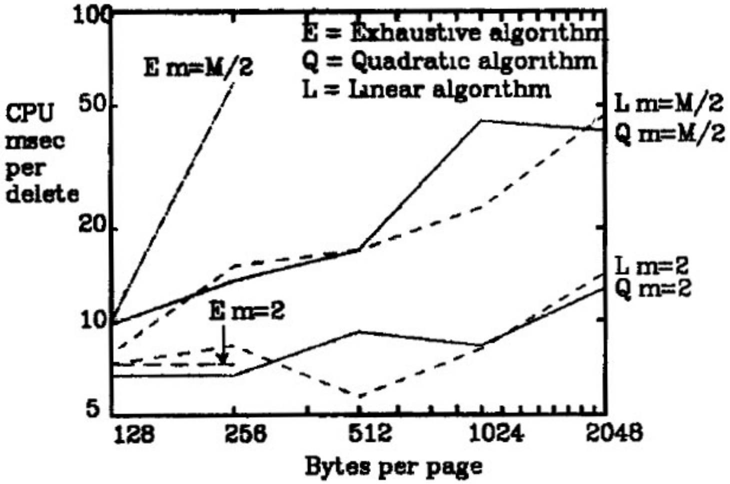
- Proof practicality of the structure
- Find suitable values for m and M
- Test various node-splitting algorithms

Testing environment:

- C-implementation running under UNIX
- Layout of a RISC-II chip with 1057 rectangles
- Different page sizes (128 – 2048 bytes)
- Various values for M (6 – 102)

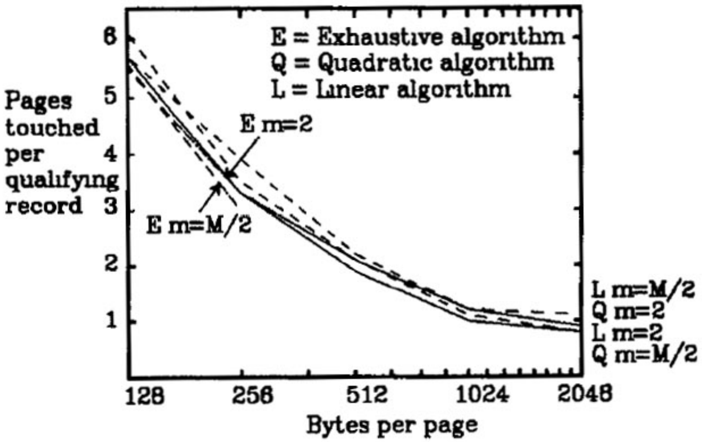
Performance of R-trees

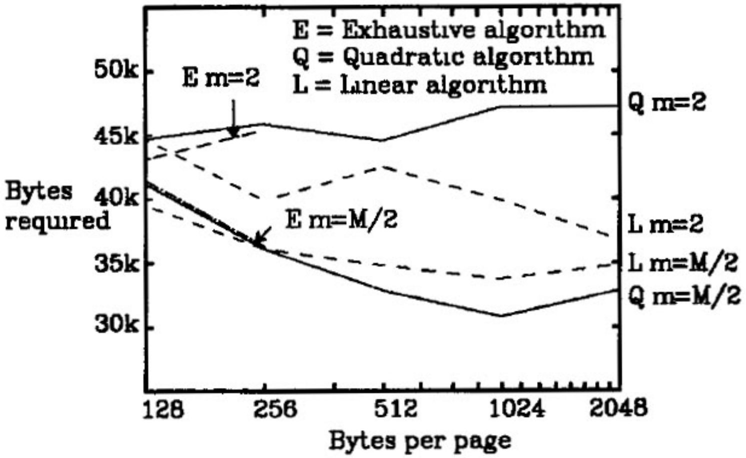




- Linear INSERT fastest
- Linear INSERT quite insensitive to M and m
- Quadratic INSERT depends on M as well as m
- DELETE extremely sensitive to m

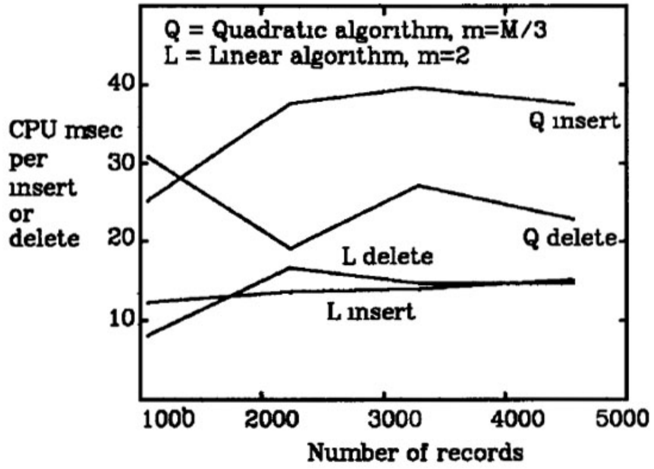
Performance of R-trees

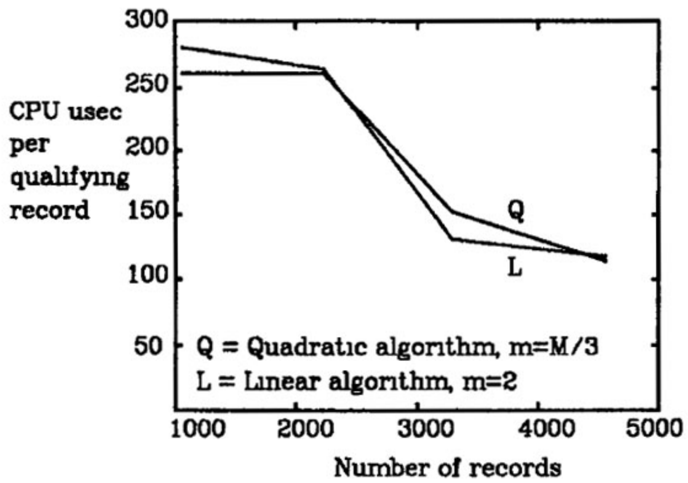




- Same page hit / miss performance for linear and quadratic split
- Slight advantage for exhaustive version ;-)
- Space usage strongly depending on m

Performance of R-trees





- Quadratic-cost splitting with jumps at INSERT operations for increasing amount of data
- Linear INSERT extremely constant
- R-tree structure very effective in directing search to small subtrees

Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. **Performance, Benchmarks**
5. R-tree Modifications
6. Conclusion

R-tree Modifications

Why?

- Improve performance
- Optimize space usage

Different forms of R-trees:

- R^+-trees: Reduce overlapping MBRs → Increase search performance
- R*-trees: Take overlapping and circumference in to consideration when performing INSERT or DELETE

R-tree Modifications

- TV-trees: Allow more complex objects than MBRs
- X-trees: Avoid / reduce overlapping by dynamically adjusting node capacities
- QR-trees: Hybrid concept, combining R-trees with quad trees

Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. **Performance, Benchmarks**
5. R-tree Modifications
6. Conclusion

Conclusions

Reasons for R-trees:

- Importance of being able to store and effectively search spatial data
- Existing structures with many limitations

Basic Idea:

- Structure similar to B-tree
- Represent objects by their MBR
- Allow overlapping

Conclusions

Algorithms:

- Search and insert similar to B-tree operations
- Delete performing re-insert instead of merge for under-full nodes (incrementally refines spatial structure)
- Node splitting: Benchmarks show that linear version performs quite as well as quadratic-cost implementation

Modifications:

- Structure is easy to adapt for special applications and their needs
- Performance advantages are usually paid for with price of a structure that gets harder to maintain

R-trees are the spatial correspondent of B-trees!

Overview – Where are we?

1. Introduction, Motivation
2. R-tree Index Structure – Overview
3. Algorithms on R-trees
   - Searching
   - Insert / Delete
   - Node Splitting
   - Updates & other Operations
4. **Performance, Benchmarks**
5. R-tree Modifications
6. Conclusion