R-trees

Buffer Paradigm

Heuristics

Computational Geometry
Spatial data: points, lines, polygons/polyhedra in \( \mathbb{R}^p \).

Example: Windowing in GIS.

For simplicity: assume \( p = 2 \).

Typical queries:

- **Spatial Join**: All intersections between stored objects and query set of objects.
- **Enclosure**: All objects containing query object.
- **Intersection**: All objects intersecting query object.
- **Point**: All objects containing query point.

Spatial data: points, lines, polygons/polyhedra in \( \mathbb{R}^p \).
R-trees

Approximate objects by Bounding Box:

[Guttman, 84]

...
Searching in R-trees

Example: Point Query

Search recursively in all subtrees of node where BB contains the query point.

In leaves:

- Average case performance depends heavily on distribution of objects
- Worst case query time: \( \Theta(N) \)

Observe:

Heuristic data structure:

4
Build R-Trees

1. Distribute objects in leaves (assign linear order).
2. Build R-tree bottom up.

Query efficiency: depends heavily on the linear order.

Example: random distribution in leaves most leaves has BB close to
BB of entire set of objects \implies\ queries inefficient.

Goal: linear order where objects close in order \implies close in Euclidean distance.

Number of I/O's:

1. \( O(\text{Sort}(N)) \)
2. \( O(\text{Scan}(N)) \)

\( O(\text{Sort}(N)) \) vs.

\( O(\text{Scan}(N)) \)

Build ("bulk load" in database community):
Space Filling Curves

The z-curve:

Comparing \((x_0; y_0)\) and \((x_1; y_1)\) having binary expansions \((x_1^1x_1^2x_1^3\cdots x_1^k; y_1^1y_1^2y_1^3\cdots y_1^k)\) and \((x_0^1x_0^2x_0^3\cdots x_0^k; y_0^1y_0^2y_0^3\cdots y_0^k)\):

Compare \(y_1^1\) and then \(y_1^2\) and then \(y_1^3\), then \(x_1^1\) and then \(x_1^2\).
The Hilbert-curve:

Note: subpattern = pattern flipped along a diagonal. Generates four possibilities (as Hilb² = \( \frac{1}{2} \) and as Hilb¹ = \( \frac{1}{2} \)).

Comparing \((x, y)\) and \((x, y)\) having binary expansions:

\[(x_1, x_2, \ldots, x_k) \text{ and } (y_1, y_2, \ldots, y_k)\]

Top level: Compare \(x^1\) and \(x^1\), then \(y^1\) and \(y^1\). Advance to next bit, choosing proper variant of this scheme.

Space Filling Curves

The Hilbert-curve:
A static R-tree build this way has good query time in practice. Empirically:

Choose ordering in leaves by sorting midpoints of bounding boxes of objects according to position on some space filling curve.

Space Filling Curves and R-Trees
Dynamic R-trees

Insert:
We need two subroutines:

Route() Given a node \( v \) and a new object \( x \), decide which subtree of \( v \) that \( x \) should belong to.

Split() Given a node \( v \) that is overflowing, decide how to split the subtrees in two groups.

Various heuristics for these have been proposed, based on minimizing various properties of changed BBS (total area, area not used by objects, distance to borders of BBS, ...).

Bottom line: Fast heuristics giving good query times in practice.
Buffered R-Trees

Principle and Analysis: exactly as for buffer trees (using Route and Split)

Advantage: Efficient bulk updates and queries. For $\Theta(N)$ size bulk:

Number of I/Os is $O(\log_B \Theta + \log(\frac{B}{1}))$ per insert instead of

$O(\log_B N)$.
Empirically: Faster bulk insertions than for previous proposals. Query time in resulting structure competitive with repeated insertions.

Note that spatial join is a bulk query.

Similar statements for queries and deletions (see paper).

**Buffered R-Trees**