Talk Outline

• Spatial join and issues

• Background: R-tree

• The seeded tree method

• Seeded tree construction

• Experiment results

• Conclusions
Objective

- **Spatial joins with**
  no pre-computed spatial indices.

- No existing solutions.

- Approach: construct spatial index structure at join time.
Previous Spatial Joins

- Cannot use relational join alg.
  - Spatial data lack total order.
  - Spatial joins are more than natural joins.

- Use spatial indices designed for spatial selections.
  - E.g. R-trees, R*-trees, ...

- Spatial indices must exist for datasets at time of join
  [Beckmann et al. 90, Brinkhoff et al. 93, Faloutsos et al. 87, Gunther 93, Guttman 84, Sellis et al. 87].

- Expensive to construct dynamically.
Spatial Join

- Spatial data:
  Data with spatial extent.
  E.g.: points, lines, regions ....

- Spatial Join:
  Given spatial data sets A and B, find all 
  \((a, b), \ a \in A \text{ and } b \in B\), such that

  \[
  \text{spatial\_predicate}(a, b) = \text{ TRUE}
  \]

- Commonly encountered predicate:
  \(\text{overlap}(a, b)\).
Background: Tree Matching

- R-tree join algorithm [Brinkhoff et al. 93]: Between two pre-computed R-trees

1. Two nodes match iff their mbrs overlap.

2. Recursively descend both trees finding pairs of matching nodes.

3. Report results at the leaf level.
Background: R-Trees

- B-tree-like data structure.

- Node contains array of \((\text{mbr}, \text{cp})\).
  - Minimal bounding box:

- Expensive to construct when large.
  - Possibility of memory thrashing.
Problem with Pre-computed Indices

- Spatial indices not exist for all datasets.
- Queries with non-spatial selections.
  - E.g. Find all government-owned buildings that overlap residential areas.
- Queries with multiple spatial joins.
  - Input many be intermediate results.
Our Approach:
Seeded Trees

- Dynamically build indices at join time.

- Principles:
  1. Index optimized for join, not selection.
  2. Exploit information about join.
  3. Low construction costs.

- Working assumption:
  - R-tree exists for one dataset.
  - Construct a seeded tree to join R-tree.
Joins vs. Selections

Dataset 1 with bounding box

Dataset 2

Data of set 2 and Bounding box of set 1

Optimized for selection

Optimized for join

Bounding box in tree 1
Bounding box in tree 2
Data object in set 1
Data object in set 2
Seeded Tree Structure

- Tailored for join with a given R-tree.
- Upper levels: Seed levels.
- Grown level: grown subtrees are R-trees.
- Slots
Seeded Tree Life Cycle

- Tree construction
  - Seeding phase
  - growing phase
  - cleanup phase

- Tree matching
Seeding Phase

- Copy upper $n$ levels of R-tree.

- Copied nodes may be transformed.
Growing Phase:
Build Linked Lists

- Insert each object through seed levels, choosing appropriate slot.

- Build linked lists at the slots.

- When buffer full, **batch-write** linked lists to disk.

![Diagram showing the process of inserting objects into a linked list structure.]
Growing Phase:
Build Grown Subtrees

- Convert linked lists into grown subtrees.
- Construct one subtree at a time.
  - Avoids memory thrashing.
- Construct subtree under \( i \) using linked lists attached to \( i \).
Tree Construction
Buffer & I/O Management

- Building linked lists: write batches.

- building grown subtrees:
  - Read units: linked lists
  - Write units: subtrees
  - At most needs 1 linked list & 1 subtree in buffer.
Cleanup Phase and Tree Matching

- Cleanup Phase: housing keeping.
  - Final adjustment of mbrs if necessary.
  - Delete empty slots.

- Tree matching: produce join result.
Seed-Level Filtering

- Object overlaps some leaf of tree
  ⇒ overlaps some node in each level.

- With copy-seeding if object overlaps no seed level
  - It overlaps no leaf of the seeding tree.
  - Don’t consider it anymore.

- Reduces seeded tree size.
Experiments

- **STJ** (Seeded Tree Join): construct a seeded tree, and match with existing R-tree.

- **RTJ** (R-Tree Join): construct an R-tree, and match with existing R-tree. (variation of [Brinkhoff et al. 93])

- **BFJ** (Brute Force Join): perform a series of window queries (i.e. spatial selections).

  - Experiment series 1:
    - vary data set size.

  - Experiment series 2:
    - vary degree of spatial clustering.
Experiment Series 1:
Total Costs

Cardinality of data set S

Number of disk I/O
Experiment Series 1: Construction Cost

Cardinality of data set S

No. of disk I/O

- RTJ
- STJ1-2N
- STJ1-2F
- STJ1-3F
Experiment Series 1:
Matching Cost

Cardinality of data set S

No. of disk I/O

BFJ
RTJ
STJ1-2N
STJ1-2F
STJ1-3F
Experiment Series 2:

Total Costs

No. of disk I/O

35000
30000
25000
20000
15000
10000
5000
0

Fraction of area used in data clustering

0.2 0.4 0.6 0.8 1

BFJ
RTJ
STJ1-2N
STJ1-2F
STJ1-3F
Experiment Series 2: Construction Cost

![Bar Chart]

- RTJ
- STJ1-2N
- STJ1-2F
- STJ1-3F

No. of disk I/O

Fraction of area used in data clustering

0 1000 2000 3000 4000 5000 6000 7000 8000 9000

0.2 0.4 0.6 0.8 1
Experiment Series 2: Matching Cost

![Bar chart showing the number of disk I/O operations for different fractions of area used in data clustering. The categories are BFJ, RTJ, STJ1-2N, STJ1-2F, and STJ1-3F.]}
Conclusions

• New method to address situations where existing spatial indices are not applicable.

• Dynamic index construction at very low cost.

• Significant performance win over other methods.

• Dynamically constructing indices for joins ⇒ Doable for spatial databases.

• Extensions:
  – 2-seeded tree joins.
  – Reducing matching costs – spatial hash joins.
New Problem

2-Seeded-Tree Join

- No spatial index for either dataset.
- Must dynamically construct 2 seeded trees.
- Difficulty:
  No R-tree to copy seed levels from.
- Solution: _Don't copy!_
  - Bootstrap seeding: determine topology and contents of seed levels from dataset.
Costs of 2-ST Joins

- Comparison unfair to seeded tree joins.
  - Two indices: given 2 pre-computed RT.
  - One index: given 1 pre-computed RT, build 1 ST.
  - No indices: build 2 ST.
Spatial Hash Join

- Relational hash join paradigm.

- **Bootstrap Seeding** to produce hash function.

- Solve multiple overlap problem.

- Good performance in our experiment.
Copying Strategies

$C_1$: Copy mbrs.

$C_2$: Copy the center points of mbrs.

$C_3$: Copy center points of mbrs at slot level. At other levels, mbr fields contain the true minimum bounding boxes of its children.
Update Strategies

$U_1$: No updates after insertions.

$U_2$: Update mbrs after each insertion to enclose inserted data objects and original seed mbrs.

$U_3$: Same as $U_2$, but updated mbrs enclose only inserted data, not seed mbrs.

$U_4$: Update mbrs at slot level as in $U_2$. Other mbrs untouched.

$U_5$: Update slot level mbrs as in $U_3$. Other mbrs untouched.