CSE 594: Spatial Data Science & Engineering

Lecture 7

Spatial Query Processing and Join Optimization

Spatial Query Processing

Nearest Neighbor Search

Branching and Bounding Method

- Total set of feasible solutions are partitioned into smaller subsets of solutions
- Smaller subsets are evaluated systematically until the best solution is found

    - Candidate solutions are thought of as a tree with full set at the root
    - Explores branches of the tree representing solution subsets
    - Before enumerating candidate solutions of a branch, the branch is pruned with upper and lower bounds of optimal solutions

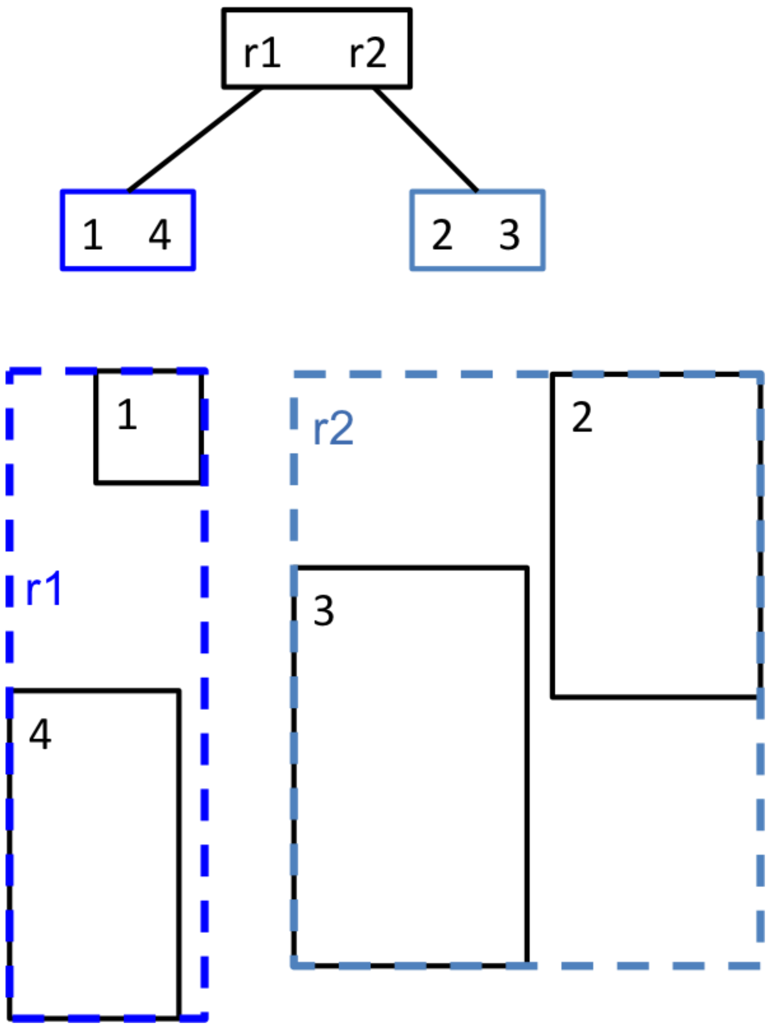
Branching and Bounding with R-tree

R-tree Properties

- Each face of an MBR must touch at least one of the objects it encloses, because it tightly encapsulates the spatial objects within it

Search Optimization Heuristics

- Visit an MBR only when it is necessary

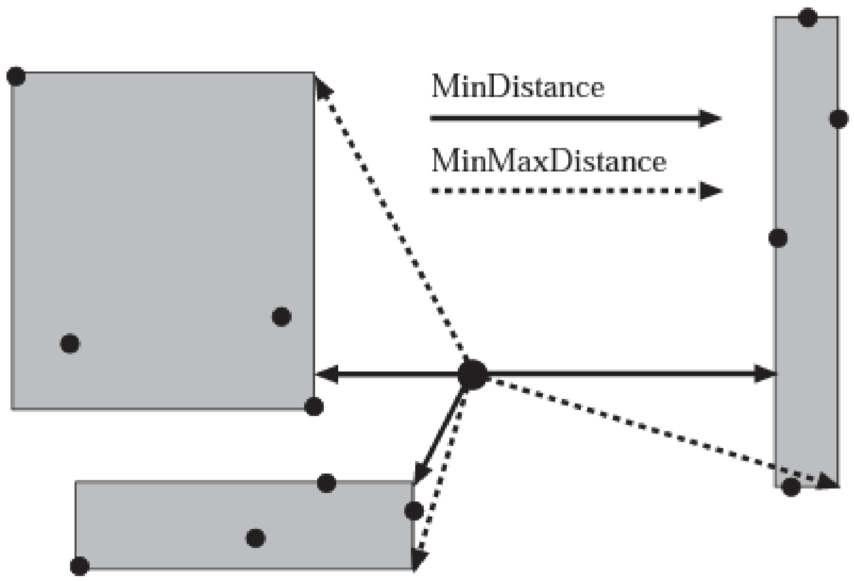


MINDIST

- If P is a query point and R is a bounding rectangle, then MINDIST(P, R) is the minimum distance between the query point P and rectangle R
- If P is inside R, MINDIST = 0, otherwise MINDIST is the distance between P and the closest point in R

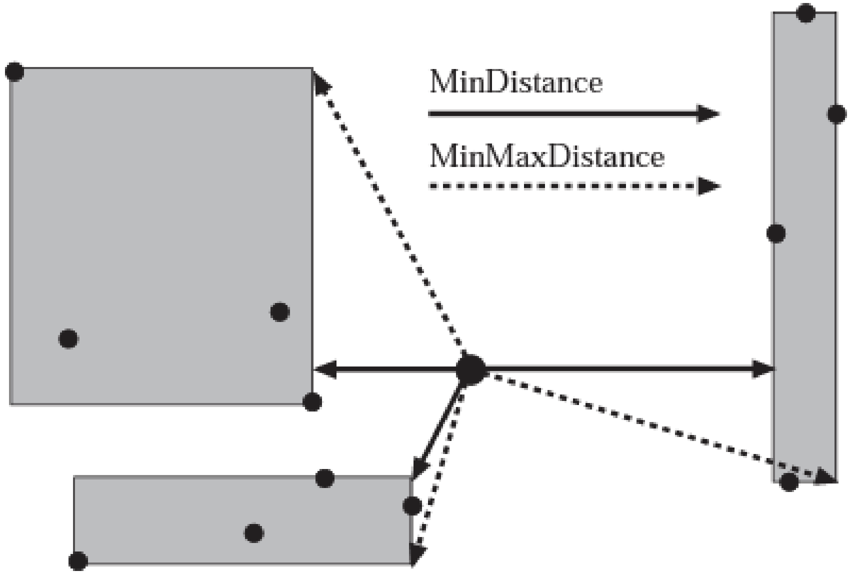
Observation

- Actual closest point is at least MINDIST distance away



MINMAXDIST

- Minimal distance from a set of maximal distances
- Calculate the distance of farthest point in each face
- MINMAXDIST is the minimum of all distances to farthest points
- Denotes there is at least one object in MBR whose distance to P is smaller than or equal to MINMAXDIST



Observation

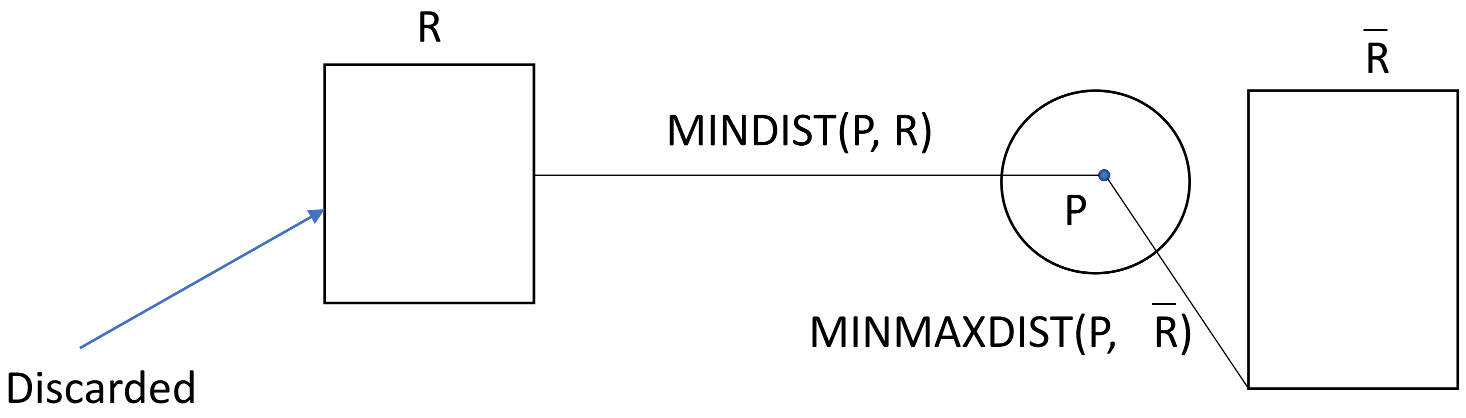
- Actual closest point is less than MINMAXDIST distance away

Pruning Heuristics

MINDIST(P, R) <= NN(P) <= MINMAXDIST(P, R)

Pruning Heuristic 1

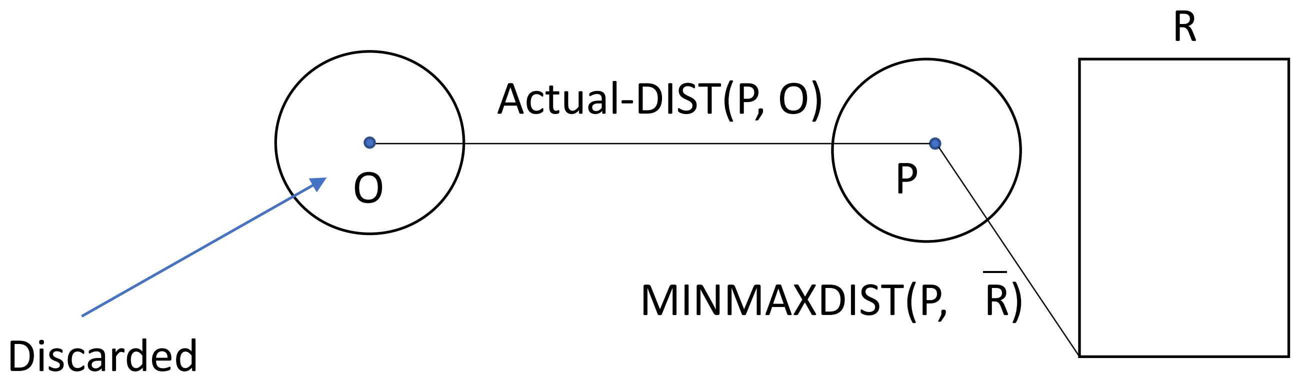
An MBR R is discarded if there exists another MBR $\bar{R}$ such that MINDIST(P, R) > MINMAXDIST(P, $\bar{R}$)



Pruning Heuristics

Pruning Heuristic 2

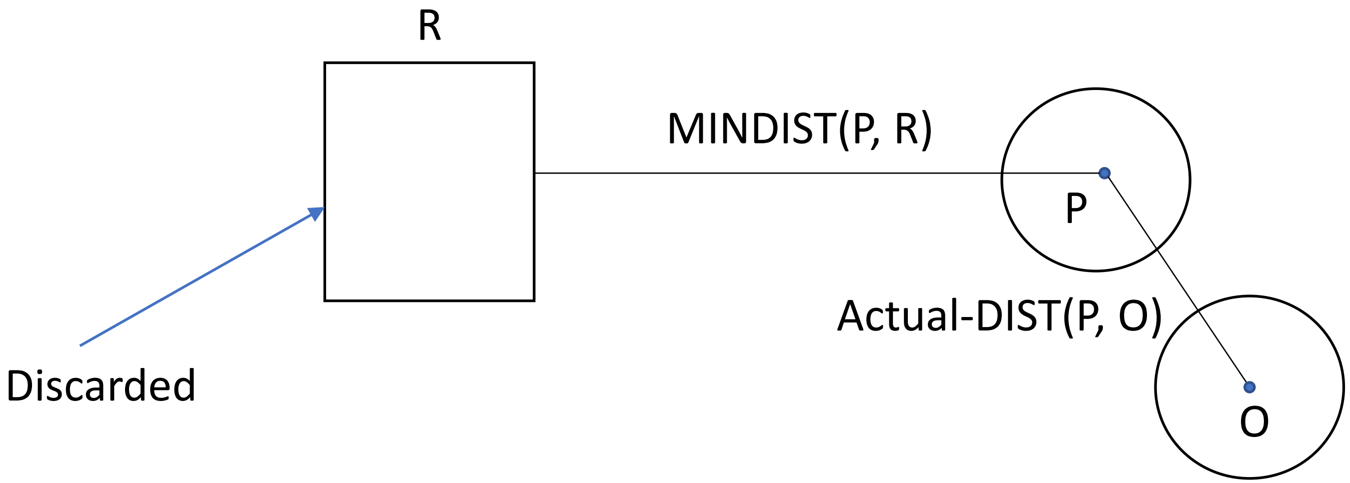
An object O is discarded if there exists an MBR R such that Actual-DIST(P, O) > MINMAXDIST(P, R)



Pruning Heuristics

Pruning Heuristic 3

An MBR R is discarded if there exists an object O such that MINDIST(P, R) > Actual-DIST(P, O)



Nearest Neighbor Search with Branching and Bounding Method

1. Initialize nearest distance as infinite distance and traverse the tree in dept first search order
2. If current node is leaf node
3. Compute distance to all objects within the leaf node, compare with the current best NN, and update if necessary
4. Else
5. Sort child nodes based on MINDIST and put them in ABL (active branch list)
6. Visit MBRs in ABL until ABL is empty
7. Prune ABL by applying heuristic 1
8. Update nearest distance applying heuristic 2 on MBRs in ABL
9. Apply heuristic 3 on each MBR in ABL, discard which can be discarded and recursively call step 2 for remaining MBRs
10. Return the last updated nearest distance

K-Nearest Neighbor Search

- Keep the sorted buffer of at most k current nearest neighbors
- Perform the pruning based on k-th distance

Best Bin First Method

- Bins are looked in an increasing order of distance from the query point
- The distance to a bin is defined as a minimal distance to any point of its boundary
- Implemented with a priority queue

    - Maintain distance to all entries in a priority queue based on MINDIST
    - Repeat
        - Inspect the next MBR in the priority queue
        - Add the children to the list and reorder
    - Until all remaining MBRs are pruned

Nearest Neighbor Search with Best Bin First Method

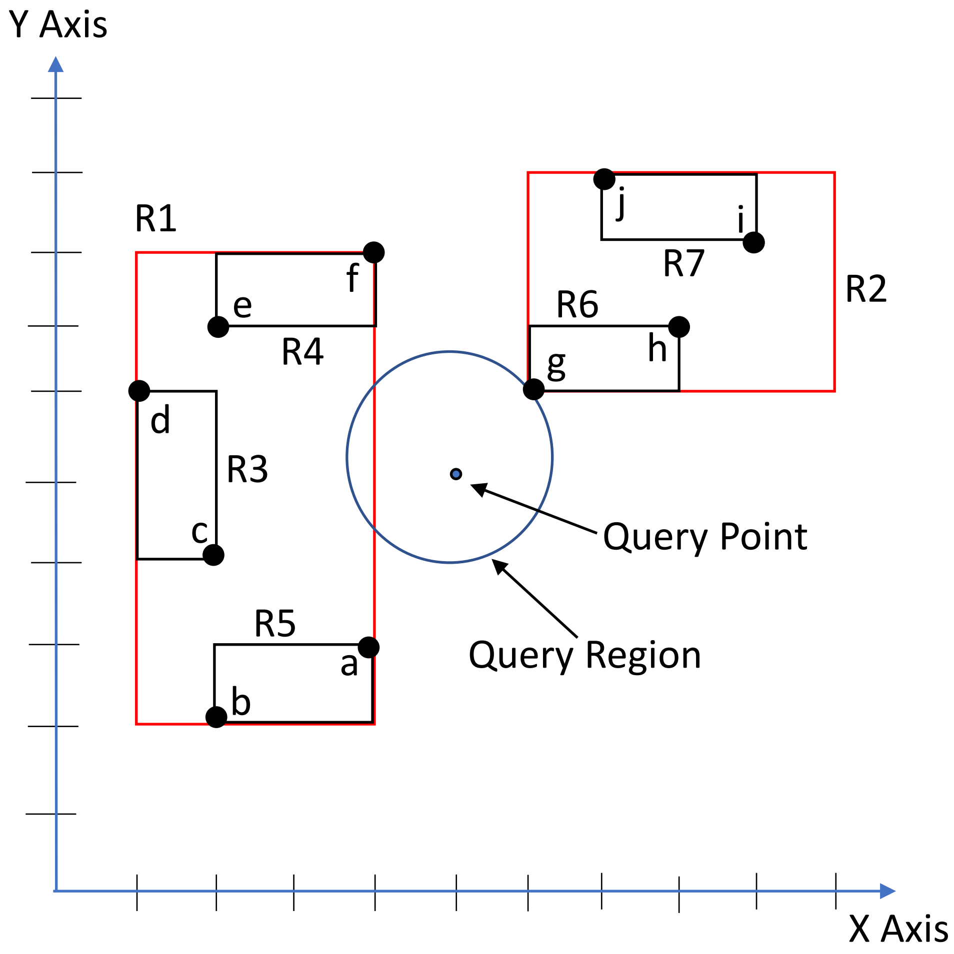
1. Initialize priority queue, Q
2. Insert root into Q
3. While not isEmpty(Q)
4.   S = dequeue(Q)
5.   If S is an object
6.   report S and exit
7.   If S is a leaf page node
8.   For each object O in S, calculate actual distance between O and query point P, and insert O into Q in the order of distance

1.   If S is an internal node
2.   For each MBR R in S, calculate MINDIST between R and query point P, and insert R into Q in the order of distance

Nearest Neighbor Search with Best Bin First Method

Steps	Priority Queue	Results
Visit Root	R1, R2	{}
Visit R1	R2, R5, R4, R3	{}
Visit R2	R6, R5, R4, R3, R7	{}
Visit R6	g, R5, R4, R3, R7, h	{}
Visit g	R5, R4, R3, R7, h	{g}

Report g and terminate



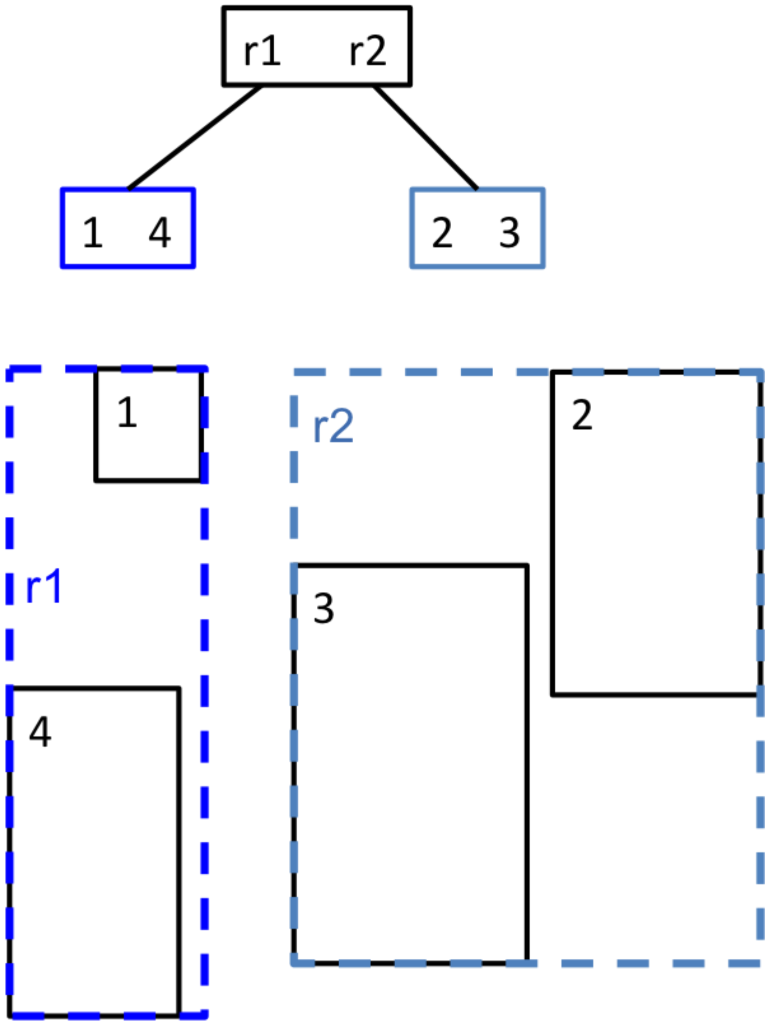
Branching and Bounding vs Best Bin First

- Best bin first search is optimal as it visits only necessary nodes, but needs to store a very large priority queue in memory
- Branching and bounding uses small lists for each node by pruning with both MINDIST and MINMAXDIST

Spatial Join Optimization

R-tree Properties for Spatial Join

- Internal nodes contain MBRs of child nodes
- Leaf nodes contain MBRs of data objects
- If there is no intersection between MBRs of internal nodes, they can be pruned for join
- Otherwise, a join is required between them



Basic Spatial Join with R-tree

Algorithm SpatialJoin1 (R , S) (both are R-tree nodes)

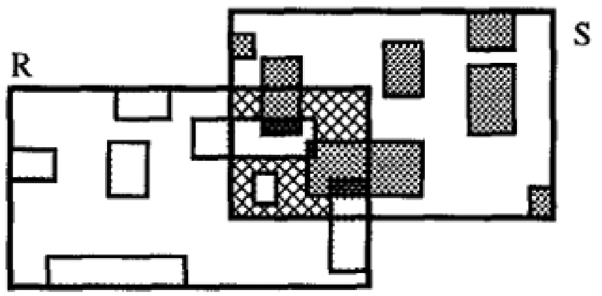
1. For each child E_s in node S do
2.   For each child E_r in node R which has an overlap with the child E_s do
3.   If E_r and E_s are leaf nodes containing data objects do
4.   Output (E_r, E_s)
5.   Else
6.   Call **SpatialJoin1(E_r, E_s)**
7.   End If
8.   End For
9. End For

Limitations of R-tree Based Basic Spatial Join

- Each entry of one node is checked against all entries of the other node, search space is still high
- Pages are selected for the next call of spatial join without considering the I/O cost for reading these pages
- Does not compute the best sequence of pairs of pages required for a join

Optimization 1

- Reduce search space further
- Recursively call SpatialJoin1 only for those pairs of child nodes which intersect with the overlapped rectangle



Optimized Spatial Join with R-tree

Algorithm SpatialJoin2 (R , S, rect) (both are R-tree nodes and rect is the intersected rectangle between R and S)

1. For each child E_s in node S which intersects with rect do
2. For each child E_r in node R which intersects with rect do
3. call **SpatialJoin1a** (E_r, E_s)
4. End For
5. End For

Optimized Spatial Join with R-tree

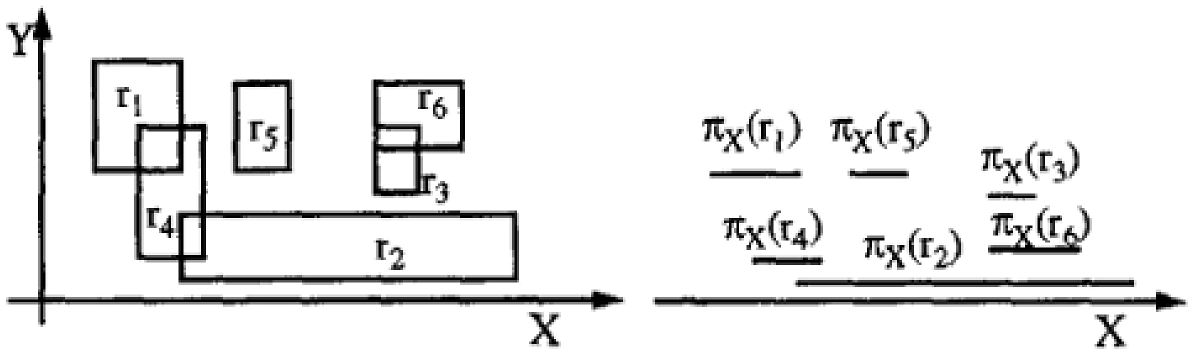
Algorithm SpatialJoin1a (R , S) (both are R-tree nodes)

1. For each child E_s in node S do
2.   For each child E_r in node R which has an overlap with the child E_s do
3.   If E_r and E_s are leaf nodes containing data objects do
4.   Output (E_r, E_s)
5.   Else
6.   Call **SpatialJoin2 (E_r, E_s, Overlap between E_r.MBR and E_s.MBR)**
7.   End If
8.   End For
9. End For

Optimizing R-tree Based Spatial Join

Optimization 2: Spatial Sorting and Plane Sweep

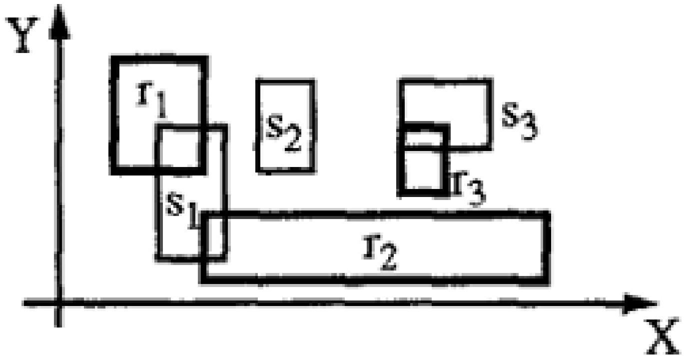
- Sort the entries in a node according to the spatial location of the corresponding rectangles
- Rectangles are two dimensional
- Perform sorting according to X-axis using plane sweep, move a line perpendicular to X-axis from left to right



Sorted Insertion Algorithm

Algorithm SortedInsertion (Rseq , Sseq, Output)

1. Set Output to empty, i to 1, j to 1
2. While i <= len(**Rseq**) and j <= len(**Sseq**) do
3. if r_i.xl < s_j.xl Then
4. **InternalLoop (r_i, j, Sseq, Output);** i = i + 1
5. Else
6. **InternalLoop (s_j, i, Rseq, Output);** j = j + 1
7. End If
8. End While



$t = r_1 : r_1 \leftrightarrow s_1$

$t = s_1 : s_1 \leftrightarrow r_2$

$t = r_2 : r_2 \leftrightarrow s_2\ ,\ r_2 \leftrightarrow s_3$

$t = s_2 :$ -

$t = r_3 : r_3 \leftrightarrow s_3$

** Pairs tested for insertion by InternalLoop algorithm. All pairs may not be inserted based on satisfaction of conditions **

Algorithm InternalLoop (t, k, Seq, Output)

- Traverse Seq starting from index k until s_k.xl < t.xu and

  append (t, s_k) into Output if t.yl < s_k.yu and t.yu > s_k.yl

Optimized Spatial Join with R-tree and Sorted Insertion

Algorithm SpatialJoin3 (R , S, rect) (both are R-tree nodes and rect is the intersected rectangle between R and S)

R = Set of child E_r in node R which intersects with rect

S = Set of each child E_s in node S which intersects with rect

Sort(R), Sort(S)

Call **SortedInsertion** (R, S, Seq)

For i = 1 to len(Seq) do

    (E_r, E_s) = Seq[i]

    If E_r and E_s are leaf nodes containing data objects do

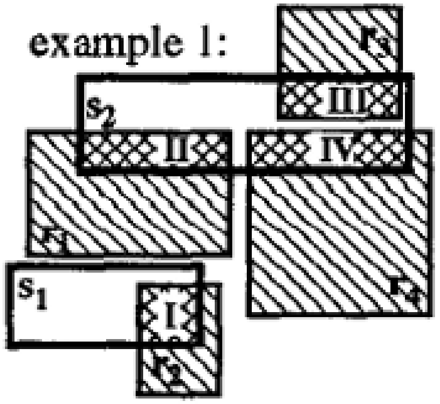
        Output (E_r, E_s)

    Else

        Call **SpatialJoin3 (E_r, E_s, Overlap between E_r.MBR and E_s.MBR)**

    End If

End For



Read Schedule: <s_1, r_2, r_1, s_2, r_4, r_3>