MSBD6000J Spring 2021

HW2 Report

XiangYu Wang
20711306
xwanggb@connect.ust.hk

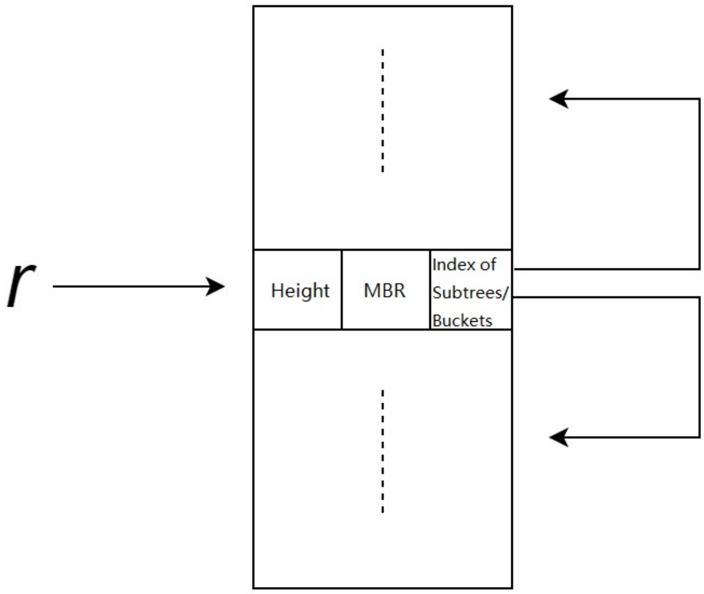
1. Introduce

This work implements a R-Tree for multidimensional data to accelerate the nearest neighbor queries.

2. Methodology

2.1. Data Structure (Task1)

In this work, A R-tree contains a list (regarded as RAM) *RTree* and an integer r. Each item of *RTree* is a node of R tree, which contains height, MBR and index of buckets/subtrees. r is the index of the root, which is used to starts the query:



2.2. The Splitting of Node (Task 1)

To make the covered area of two groups of MBRs as small as possible, a heuristic algorithm is used to split the node. For a leaf node:

```
split(bucket):
    find two points farthest apart in bucket(denoted as a and b)
    G_1 = {a}, G_2 = {b}
    for p in bucket \a \b:
        if d(a, p) > d(b, p):
            add p to G_2
        else:
            add p to G_1
    return G_1, G_2
```

For a non-leaf node:

```
split(subtrees):
    c = [center(subtree.MBR) for subtree in subtrees]
    find two points farthest apart in c (denoted as a and b)
```

```
    G_1 = {The subtree corresponding to a}, G_2 = {The subtree corresponding to b}
    for p in c \a \b:
        if d(a, p) > d(b, p):
            add the subtree corresponding to p to G_2
        else:
            add the subtree corresponding to p to G_1
    return G_1, G_2
```

2.3.Insert a Point (Task 1)

To reduce the cost of querying, a point is inserted to the subtree with the minimum MBR after expansion:

```
Given: n: the capacity of bucket, d: the maximum number of children that a non-leaf node has.
insert(RTree, index, point, root):
    v = RTree[index]
    if v is a leaf:
        add point to v and update MBR
        if there are more than n points in v:
            v, v' = split(v)
            add v' to RTree
            if v is root:
                add new root to RTree
                return the index of v' and new root
            return the index of v' and root
        else:
            return Null, root
    else:
        t = the index of the subtree with the minimum MBR after expansion
        update MBR
        v', root = insert(RTree, t, point, root)
        if v' != Null:
            add v' to v.subtree and update MBR
            if there are more than d subtrees in v':
                v, v' = split(v)
                add v' to RTree
                if v is root:
                    add new root to RTree
                    return the index of v' and new root
                return the index of v' and root
        return Null, root
```

2.4.Construct a R Tree (Task2)

First, *RTree* and *r* are respectively initialized to a tree that has only one empty leaf node and 0. Then, insert all points to *Rtree*:

```
Rtree = a tree that has only one empty leaf node, r = 0
for p in POIs:
    _, r = insert(Rtree, r, p, r)
```

2.5.Nearest Neighbor(NN) Query via Pruning (Task3)

```
Nearest Neighbor(RTree, index, point, nearest, distance, minmaxdist_min):
```

```
    v = RTree[index]
    if v is a leaf:
        linear scan and update nearest, distance
        return nearest, distance, minmaxdist_min
    else:
        for t in v.subtree:
            if mindist(t, point) < minmaxdist_min:
                calculate minmaxdist(t, point) and update minmaxdist_min
                if mindist(t, point) < distance:
                    nearest, distance, minmaxdist_min = Nearest Neighbor(RTree, t, point, nearest, distance, minmaxdist_min)
                else:
                    prune t
            else:
                prune t
        return nearest, distance, minmaxdist_min
```

3. Experiment

In this experiment, 10 random points are:

ID	Location	Nearest Neighbor	Distance
1	(115.9453, 40.1405)	(115.9508, 40.1400)	0.0055421
2	(117.0836, 40.5817)	(117.0844, 40.5887)	0.0070366
3	(117.4573, 40.9662)	(117.4405, 40.6608)	0.30583
4	(116.9097, 39.6134)	(116.8848, 39.6793)	0.070371
5	(116.8198, 40.7601)	(116.8006, 40.7664)	0.020156
6	(116.6941, 40.9932)	(116.6288, 41.0075)	0.066864
7	(115.5113, 41.0180)	(115.7757, 40.5198)	0.56405
8	(116.1137, 40.9221)	(116.3553, 40.9122)	0.24189
9	(117.2728, 40.3985)	(117.2298, 40.4219)	0.048959
10	(116.0585, 40.8211)	(116.1742, 40.6529)	0.20416

3.1. Case 1: $n = 100$, $d = 6$

3.1.1. Construction of R-Tree

Non-Leaf	Overlapped	Leaf	Height	Time
946	946	3027	6	8.323s

Utilization	0-25%	25-50%	50-75%	75%-100%
Confidence	10.935%	24.149%	29.765%	35.15%

3.1.2. Nearest Neighbor Query

ID	Visited	Calculated	Pruned
1	2945	46819	1465
2	1327	10145	794
3	471	1034	337
4	2619	23619	1649
5	612	3423	403
6	158	806	103
7	1701	7350	1169
8	1103	7538	703
9	1140	7923	708
10	1499	8831	989

3.2. Case 2: $n = 100$, $d = 2$

3.2.1. Construction of R-Tree

Non-Leaf	Overlapped	Leaf	Height	Time
4579	4579	3009	16	11.2260s

Utilization	0-25%	25-50%	50-75%	75%-100%
Confidence	10.402%	24.194%	29.678%	35.726%

3.2.2. Nearest Neighbor Query

ID	Visited	Calculated	Pruned
1	4176	45078	1005
2	1695	11400	441
3	489	1374	165
4	3237	23826	971
5	391	1408	126
6	219	907	64
7	2708	14758	866
8	1356	7831	405
9	1492	10223	396
10	2048	12552	609

3.3. Case 3: $n = 10$, $d = 6$

3.3.1. Construction of R-Tree

Non-Leaf	Overlapped	Leaf	Height	Time
7952	7952	26164	7	10.6116s

Utilization	0-25%	25-50%	50-75%	75%-100%
Confidence	4.743%	13.247%	34.311%	47.699%

3.3.2. Nearest Neighbor Query

ID	Visited	Calculated	Pruned
1	7709	2819	5526
2	2514	1194	1715
3	458	64	332
4	5170	653	3931
5	786	168	556
6	270	106	187
7	2462	124	1865
8	1746	205	1298
9	1612	584	1130
10	2543	235	1903

3.4. Case 4: n = 10, d = 2

3.4.1. Construction of R-Tree

Non-Leaf	Overlapped	Leaf	Height	Time
40554	40554	26849	19	15.386s

Utilization	0-25%	25-50%	50-75%	75%-100%
Confidence	5.900%	14.086%	35.126%	44.888%

3.4.2. Nearest Neighbor Query

ID	Visited	Calculated	Pruned
1	12280	4930	4429
2	3982	1693	1330
3	1014	127	387
4	7714	1480	3168
5	1676	451	606
6	438	129	157

7	3657	474	1457
8	2885	934	1009
9	3037	1030	1071
10	3654	903	1349

4. Discussion and Potential Improvement

According to the experiment results, I guess:

a. As n decreases, the number of calculations decreases and that of visits increases.

b. As d increases, the number of visits decreases.

To find the optimal parameters, more experiment is required. Also, a window queries-based solution should be attempted in the future:

```
Nearest Neighbor via window query(point, RTree, root):
    v = root
    while v is not a leaf:
        v = the subtree with the minimum MBR after expansion in v
    Linear scan to find the nearest neighbor in v and the corresponding distance d.
    Query all points in: (point – d, point + d)
    Linear scan to find the nearest neighbor p and the corresponding distance d.
    return p, d
```