INFS4205/7205 Advanced Techniques for High Dimensional Data

Tutorial 8: High Dimensional Indexing and Search

Semester 1, 2020

Question 1: What is the dimensionality curse? Why we still can do the similarity search in high dimensional space?

Sample Solution:
The dimensionality curse are the following observations:

1. The number of partitions 2^d grows exponentially as the dimensional number grows. When d becomes large enough, we have more partitions than data points.
2. When data is uniformly distributed, most of the data are in the boundary regions, leaving the central space a very hollow space. Suppose we take the range of $[0.05, 0.95]$ of each dimension, then the interior region's volume is 0.9^d. When $d = 50$, that takes only 0.005 of the entire volume.
3. It would be hard to distinguish the distance between the objects. The distance between the nearest neighbor and the farthest neighbor becomes nearly the same.

Because the real data distribution is nearly never uniformly distributed, there are always some clusters and skewed distribution. Therefore, we still can find the nearest neighbor within each cluster.

Question 2: X-Tree contains three kinds of nodes: *Data Node, Normal Directory Node*, and *Supernode*. Describe what causes the following two special cases of the X-Tree and analyze their performance: (1) None of the directory nodes is a Supernode; (2) The directory consists of only one large Supernode (root).

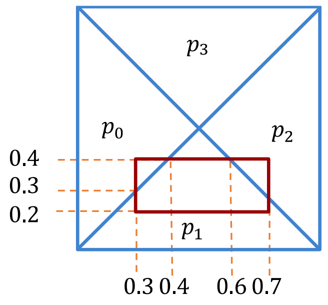
Sample Solution:

(1) The X-Tree has a completely hierarchical organization of the directory and is therefore similar to an R-Tree. It could be caused by the low

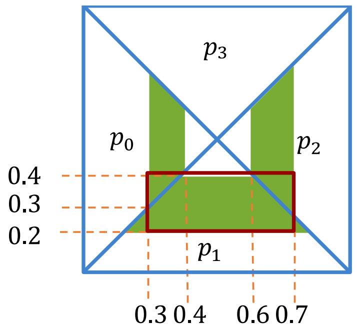
dimensional and non-overlapping data. The performance corresponds to the tree height. It has the extreme height of the X-Tree.

(2) The directory of the X-Tree is basically one root: one Supernode that contains the lowest directory level of the corresponding R-Tree. It could be caused by high-dimensional and highly overlapping data. The performance corresponds to the performance of a linear directory scan. It has the extreme size of a Supernode.

Question 3: Given a 2D pyramid as shown below, what are the query ranges to search in the B-Tree?



Sample Solution:



The query region covers three pyramids: p_0, p_1 and p_2. The actual query regions are labeled in the green. The height ranges in these three pyramids are computed as follow:

1. For the height in p_0, its range is [0.5-0.4, 0.5-0.3] = [0.1, 0.2].
2. For the height in p_1, its range is [0.5-0.4, 0.5-0.2] = [0.1, 0.3].
3. For the height in p_2, its range is [0.6-0.5, 0.7-0.5] = [0.1, 0.2].

Then we add each of these values with their corresponding pyramid ID to generate the final search range in the B-Tree:

1. For Pyramid p_0: [0.1, 0.2]
2. For Pyramid p_1: [1.1, 1.3]
3. For Pyramid p_2: [2.1, 2.2]

Question 4: VA-File bears a close resemblance to the grid file as we discussed in the spatial index module. They both impose a grid on the underlying dataspace, and queries are processed by inspecting data that falls into relevant grid cells. What are the differences between them?

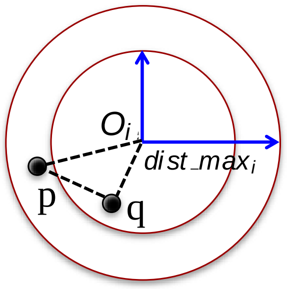
Sample Solution:

1. In VA-File, the relevant grid cells are obtained by a sequential scan of all the grid cells, while in grid file, the relevant grid cells are obtained by random access via the aid of the d linear scales.
2. In the grid file, the boundaries of the slices are modified as the underlying data changes. Thus, the dynamic behaviour of the VA-File may be bad, which means that, at times, the VA-File may have to be rebuilt.
3. Moving points in a grid file is random access, while the same action in VA-File needs a scan of the entire VA-File. Therefore, the grid file has a good dynamic behavior.

Question 5: How is the triangle inequality is used to determine the search space in the iDistance?

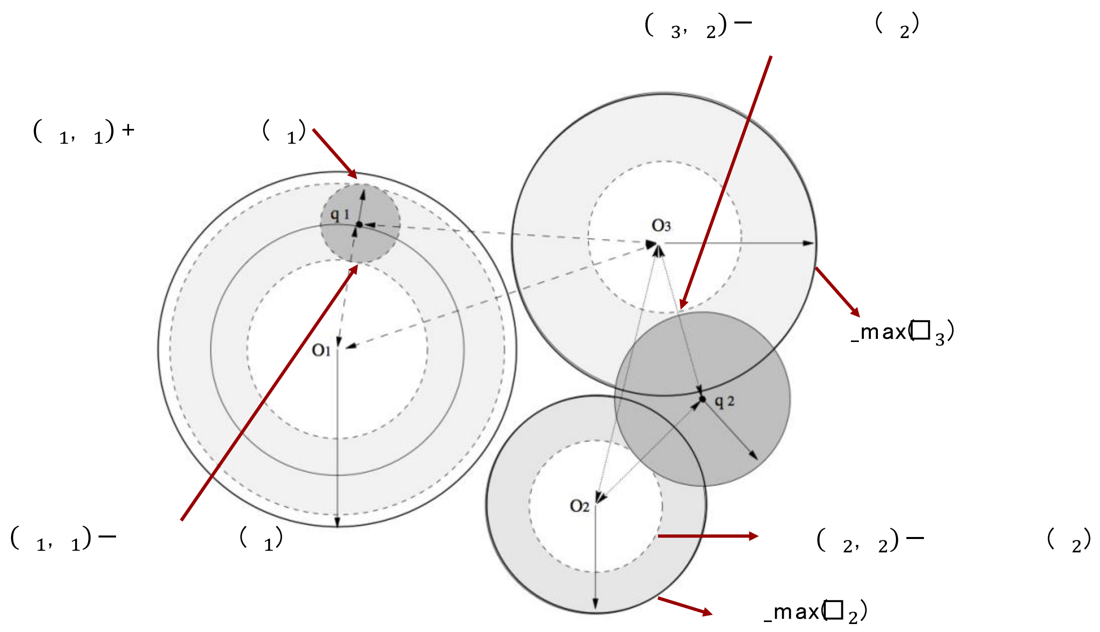
Sample Solution:

As we can see from the following example, the cluster reference point O, the query point q, and the result p form a triangle. Because the data are organized with their distance to O, so we have to know range of $dist(O_i, p)$ to determine the search space. The lower bound of the search space (the inner radius) is $dist(O_i, q) - querydist(q)$, and the upper bound of the search space (the outer radius) is the smaller one of $dist\_\max(O_i)$ and $dist(O_i, q) + querydist(q)$.



As we can see in this example, for query point q_1 with query range $querydist(q_1)$, the lower bound the query result is $dist(O_1, q_1) - querydist(q_1)$, and the upper bound the query result is $dist(O_1, q_1) + querydist(q_1)$. The result search space is a ring around O_1.

q_1 does not need to test visit cluster 2 and cluster 3 because its lower bound to O_2 (similar to O_3) is $dist(O_2, q_1) - querydist(q_1) > dist\_\max(q_2)$. Therefore, O_2 and O_3 are pruned.



For query point q_2, its lower bounds to O_2 and O_3 are $dist(O_2, q_2) - querydist(q_2)$ and $dist(O_3, q_2) - querydist(q_2)$. Their upper bounds are all their maximum distance because their maximum distances are all smaller than $dist(O_i, q_1) + querydist(q_1)$.

q_1 does not need to visit cluster 1 because its distance lower bound to O_1 is $dist(O_1, q_2) - querydist(q_2) > dist\_\max(O_1)$.