Spatiotemporal Data Management and Analytics

Question Set Day 3: Managing Spatiotemporal and High Dimensional Data

Semester 3, 2020

Question 1: LCSS (Longest Common Sub-Sequence) is a way to compare the similarity between two sequences. Given two sequences, LCSS finds the length of the longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous (if we require the contiguous, it is the longest common sub-string). For example, "abc", "agb", "bdf", "aeg", "acefg" are all sub-sequences of "abcdefg". Such a meter can also be used in to evaluate how similar two trajectories are. Given two points p and q, a distance threshold ϵ, we regard p and q as the same point if $dist(p,q) \leq \epsilon$. Please discuss the dynamic programming technique that can be used to compute LCSS.

Sample Solution:
We use two sequence to illustrate how LCSS can be computed. Suppose one string is S=AGGCAB, another one is T=GXCXAYB. We first divide this problem into a set of subproblem: we will look at the LCSS of a prefix of S and a prefix of T, running over all pairs of prefixes. For simplicity, let's worry first about finding the length of the LCSS and then we can modify the algorithm to produce the full result.

Suppose we use $LCSS[i,j]$ to denote the result of LCSS of sub-sequence $S[1,\dots,i]$ and $T[1,\dots,j]$ (If $i=2$, and $j=3$, we are comparing AG and GXC). And suppose we already know all the results of shorter prefix pairs ($LCSS[i-,j]$, $LCSS[i,j-1]$, $LCSS[i-1,j-1]$). How can we solve the $LCSS[i,j]$ with those smaller and already solved problems' results? Here are two cases:

1. When $S[i] \neq T[i]$, then the result has to ignore one of $S[i]$ or $T[j]$, so we have
    - $LCSS[i,j] = \max(LCSS[i-1,j], LCSS[i,j-1])$
    - Maximum value of the upper and left neighbor cells

2. When $S[i] = T[i]$, then we increase the already matched value of $LCSS[i-1, j-1]$ by 1:
    - $LCSS[i,j] = LCSS[i-1, j-1] + 1$
    - Top left neighbor value plus one

Therefore, we need only two loops to compute these LCSS. Initially, the first row and the first column are set to 0.

	ϕ	G	X	C	X	A	Y	B
ϕ	0	0	0	0	0	0	0	0
A	0							
G	0							
G	0							
C	0							
A	0							
B	0							

Then we fill this table row by row with the previous two rules:

	ϕ	G	X	C	X	A	Y	B
ϕ	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
C	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0	1	1	2	2	2	2	4

Therefore, the LCSS is |GCAB|=4.

Question 2: R-tree is one of the most popular indexing methods for 2D spatial data. For spatiotemporal data, where points can move over time and polygons can have changing locations and shapes, it is inefficient to have one R-tree to index such data at every time snapshot.

1. One proposal is to use 3D R-tree to index spatiotemporal data, as a spatiotemporal point can be represented in the form of (x, y, t) indicating that the point is at location (x, y) at time t. Give an example to show that this approach can be highly inefficient.
2. Historical R-tree (HR-tree) is proposed to index such data. Please briefly describe the key ideas and benefits of HR-Tree.

3. Give an example to show that HR-tree can also be highly inefficient for some datasets or queries.

Sample Solution:

1. The temporal dimension is unbounded / Box is inefficient to approximate lines / Only efficient time slice and range query, not efficient for trajectory-based (similarity) query
2. One R-tree for each time snapshot. If the objects in one MBR do not change, then the $t + 1$ snapshot can share t's node. So it takes less space.
3. When the objects keeps moving and the objects in MBRs change

Question 3: 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 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.