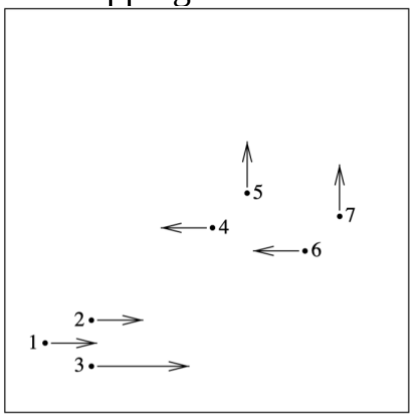
INFS4205/7205 Advanced Techniques for High Dimensional Data

Tutorial 7: Spatiotemporal Data Management

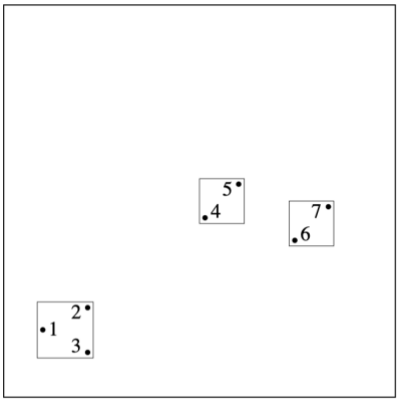
Semester 1, 2020

Question 1: What is the benefit of using the TPR-Tree than the other R-Tree-based structures to index the moving objects?

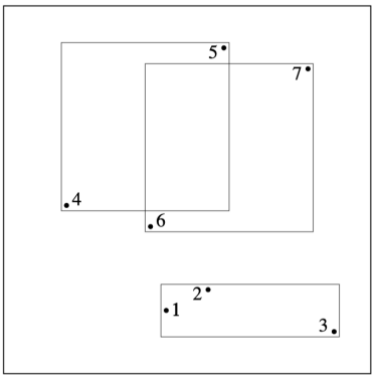
Sample Solution:
The TPR-Tree is more suitable for queries with a time-interval and has smaller overlapping ratio by considering the current location and the future location. An example of the lower overlapping is shown below:



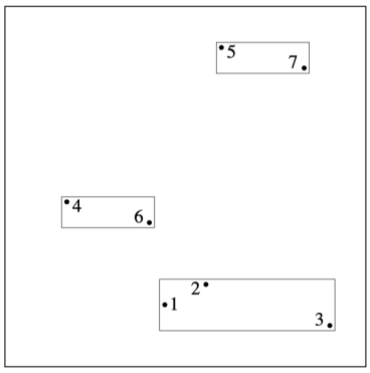
Suppose we have seven points with their current locations at this timestamp as shown in the above figure, and they are moving directions are labeled as the arrows. If only considering the locations of the current timestamp, the optimal MBR should be like this:



However, if we use these three MBRs to approximate the points in the next timestamp, they will have a very large overlapping as shown below:



Therefore, if we consider both the locations and the prediction of the future locations (with moving direction and velocity), we could have better choices, like put 5 and 7 together, put 4 and 6 together.



Question 2: 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 3: Edit Distance is another widely used string similarity measurement that can be used to compare two trajectories. To convert from one string to another, for each character, we have four choices: let it be, insert another one, delete it, and replace it. The insertion, deletion, and substitution are call edit operations, and the minimum number of edit operations it takes to transform one string to another is call the edit distance between them. Similarly, in the trajectory distance, we can regard two points within a distance threshold as the same one. In this way, the trajectory similarity is converted to the string similarity. Please use the example in Question 1 to show what is the edit distance between S and T.

Sample Solution:
All the notations are the same as Question 1. For each cell $[i, j]$ in the table, it means the minimum number of edit operations it takes to convert string $S[1, i]$ to $T[1, j]$. The edit operations have the follow actual effects on the value computation. Suppose we are now at $[i, j]$, we have three choices to compute its edit distance:

1. We can use the edit distance between $S[1, i-1]$ and $T[1, j]$ (the left neighbor), so we only need insert the value of $T[j]$ at $S[i]$ to make $S[1, i]$ and $T[1, j]$ the same. Therefore, the edit distance at $ED[i, j] = ED[i, j-1]+1$.
    1. For example, in the first line, from the empty string ϕ to G, we have to insertion G, so the edit distance is 1; From G to GX, we have insert X, so the edit distance is 1+1… And eventually, from ϕ to GXCXAYB, we have to insert 7 characters in total.
    2. Whenever we use the edit distance from the left neighbor, it means we do the insertion, so the edit distance plus 1.
2. We can also use the edit distance between $S[1, i]$ and $T[1, j-1]$ (the upper neighbor), so we only need delete the value of $S[i]$ to make $S[1, i]$ and $T[1, j]$ the same. Therefore, the edit distance at $ED[i, j] = ED[i, j-1]+1$.

ED[i-1,j]

    1. For example, in the first column, it means the edit distance from A to ϕ, so we delete A, and its edit distance is 1; Then if we convert AG to ϕ, the edit distance is 2… Finally, the edit distance from AGGCAB to ϕ is 6, because we have to delete all six characters.

2. Whenever we use the edit distance from the upper neighbor, it means we do the deletion, so the distance plus 1.

3. We can also use the edit distance between $S[1, i-1]$ and $T[1, j-1]$ (the upper-left neighbor). This time we have two cases to consider
    1. If $S[i] = T[j]$, we can do not need any editing, so $ED[i,j] = ED[i-1, j-1]$
    2. If $S[i] \neq T[j]$, we can replace $S[i]$ with $T[j]$, so $ED[i,j] = ED[i-1, j-1]$+1
4. Now for each $ED[i,j]$, we have the following three choices. We use the minimum of these three as the edit distance:
    1. From the left neighbor, $ED[i,j] = ED[i, j-1]$+1
    2. From the upper neighbor, $ED[i,j] = ED[i-1, j]$+1.
    3. From the upper-left neighbor:
        i. $ED[i,j] = ED[i-1, j-1]$, if $S[i] = T[j]$
        ii. $ED[i,j] = ED[i-1, j-1]$+1, if $S[i] \neq T[j]$

In summary, $ED[i,j] = \min(ED[i,j-1]+1, ED[i-1,j]+1, ED[i-1,j-1] + 0/1)$. So only three computations and taking the minimum is enough.

	ϕ	G	X	C	X	A	Y	B
ϕ	0	1	2	3	4	5	6	7
A	1	1	2	3	4	4	5	6
G	2	1	2	3	4	5	5	6
G	3	2	2	3	4	5	6	6
C	4	3	3	2	3	4	5	6
A	5	4	4	3	3	3	4	5
B	6	5	5	4	4	4	4	4

The edit distance between AGGCAB and GXCXAYB is 4. The actual edit steps are: Delete the first A, replace the second G with X, insert X between C and A, insert Y between A and B.

Question 4: How to compute the DTW with the dynamic programming?

Sample Solution:
Unlike LCSS and ED, the DTW is not about how many same characters or how many operations, but the actual distance between two trajectories. Still, using the points are not handy to illustrate the example, we use two one dimensional numbers here: $S = \{1,2,3,5,5,5,6\}, T = \{1,1,2,2,3,5\}$. Therefore,

the distance between to numbers are just their difference. When working in the 2D point, we can simply replace the distance with Euclidean distance or Network distance.

Like the dynamic programming in LCSS and ED, for each cell $[i, j]$, we also have three sources: the left neighbor $DTW[i-1, j]$, the upper neighbor $DTW[i, j-1]$ and the upper-left neighbor $DTW[i-1, j-1]$. All we need to do is take the minimum of them and add the distance between $S[i]$ and $T[j]$.

	0	1	1	2	2	3	5
0	0	INF	INF	INF	INF	INF	INF
1	INF	0	0	1	2	4	8
2	INF	1	1	0	0	1	4
3	INF	3	3	1	1	0	2
5	INF	7	7	4	4	2	0
5	INF	11	11	7	7	4	0
5	INF	15	15	10	10	6	0
6	INF	20	20	14	14	9	1