Computational complexity: Discuss the computational complexity of the algorithm implemented in Part F (i.e., DTW with total path length constraint). Is it different from the algorithm in Part A (i.e., standard DTW)? Explain the recurrence relation.

Notations used in this report:

A: time sequence A

B: time sequence B

A[i]: the ith element in time sequence A

B[j]: the jth element in time sequence B

n: the size/length of time sequence A

m: the size/length of time sequence B

l: the Total Path Length Constraint, represents maximum allowable total steps in the alignment

i: the row index of the cumulative cost matrix, also represent index of sequence A, it takes value 1 to n

j: the column index of the cumulative cost matrix, also represent index of sequence B

k: the height index of the cumulative cost matrix, also represent path length in which restricted by the total path length constraint

Different from part 1a where each cell in a 2-dimensions matrix (costMatrix[i][j]) represents the minimum cumulative cost among all possible path lengths for first ith elements in sequence A and the first jth elements in sequence B.

In part 1f, I had used a 3-dimensions cost matrix, and each cell in the 3D matrix (costMatrix[i][j][k]) represents the minimum cumulative cost for first ith elements in sequence A and the first jth elements in sequence B of a particular path length k where k is bounded by the total path length constraint.

The recurrence relation in part 1a is shown below:

costMatrix[i][j] = costOf(A[i-1], B[j-1]) + min(costMatrix[i -1][j], costMatrix[i, j-1], costMatrix[i - 1][j - 1])

In part A, the time complexity of initialising the cost matrix is (n+1)*(m+1) = Θ (n*m), and the time complexity of computing cost matrix in an unconstrained DTW is Θ (n*m) since we visit every cell in the matrix except the first row and column. The space complexity of part 1a is (n+1)*(m+1) =

Θ (n*m) as (n*m+n+m+1) has same order of growth as (n*m).

Since the representation of cost matrix changed in part 1f, the way we fill the cost matrix would changed as well, which means the recurrence relation we used in part 1f would changed, the recurrence relation in part 1f is shown below:

costMatrix[i][j][k] = costOf(A[i-1], B[j-1]) + min(costMatrix[i -1][j][k-1], costMatrix[i, j-1][k-1], costMatrix[i - 1][j - 1][k-1])

Comparing with the standard DTW algorithm we used in part 1a that has a space complexity of

Θ (n*m), since I had used a 3D matrix for part 1f in which the new dimension is the step/path length, the new space complexity would be:

$$((n+1)*(m+1)*(l+1)) = \Theta\ (n*m*l)$$

As ((n+1)*(m+1)*(l+1)) has same order of growth as (n*m*l)

Moreover, the time complexity for part 1f will be different from the time complexity n*m = Θ(n*m) for part 1a, since we increase the dimension of the cost matrix while still scanning every cell of the 3-dimensions cost matrix, it is natural to see the increase in time complexity. Using the new recurrence relation shown below:

costMatrix[i][j][k] = costOf(A[i-1], B[j-1]) + min(costMatrix[i -1][j][k-1], costMatrix[i, j-1][k-1], costMatrix[i - 1][j - 1][k-1])

We can see that the time complexity for part 1f = (n*m*l) = Θ (n*m*l)

Therefore, the space complexity and time complexity for part 1f are both greater than the standard DTW in part 1a, however, in considering the total path length constraint in its algorithm, part 1f's modified DTW algorithm is a more accurate and flexible algorithm in aligning two time series. It should be used when accuracy is an important aspect of time series alignment and calculation power and space are adequate.