

A **brief** introduction to Dynamic Time Warping

May 2020

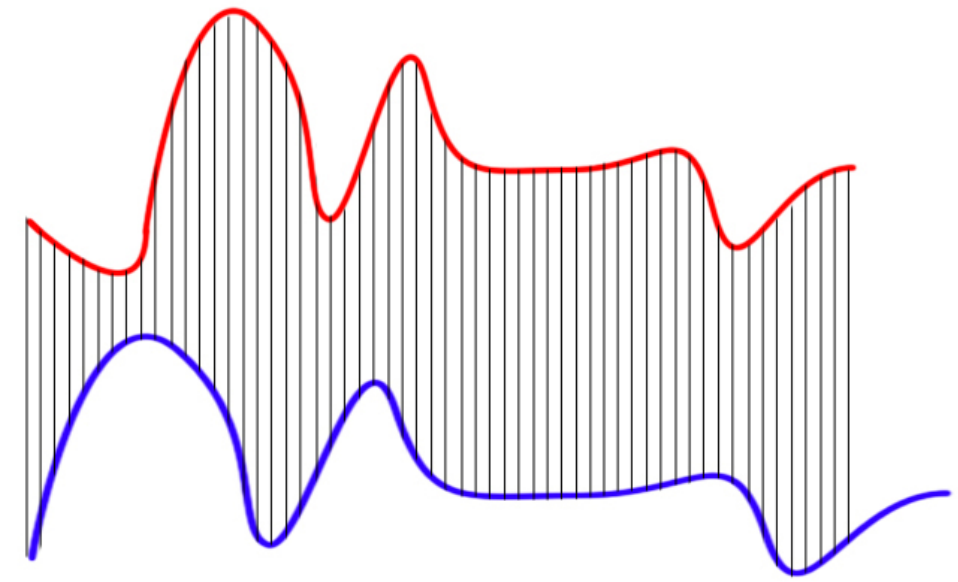
QS Update

When will this be useful?

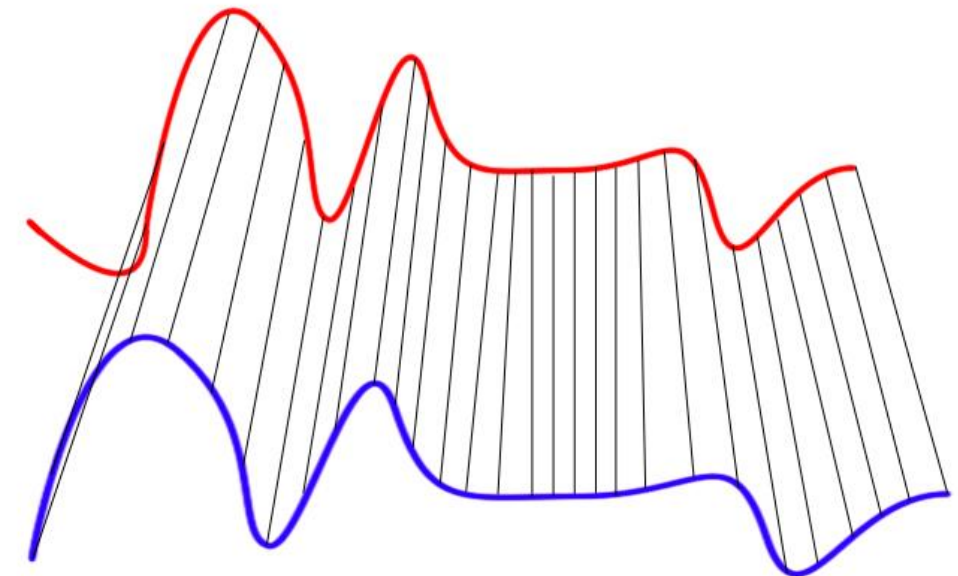
- When you need to dynamically compare time series data when time indices between comparison points do not sync up perfectly.
 - *Speech Recognition* – determining whether one phrase matches another, even if the phrase is spoken faster or slower in comparison.
 - *Financial markets* – comparing stock trading data over similar time frames, even if they do not match up perfectly. For example, comparing monthly trading data for February (28 days) and March (31 days).
 - *Income trajectory* – cluster individuals based on their patterns of income. As they retire, do we see their income sharply decrease, or gradually decline?

The objective of time series comparison is to produce a distance metric between two input time series

- Similarity or dis-similarity is typically calculated by converting data into vectors and calculating the Euclidean distance between those points in a vector space.
 - This is, however, extremely restrictive.
- Dynamic time warping allows the two curves to match up evenly even though the X-axes (i.e. time) are not necessarily in sync.

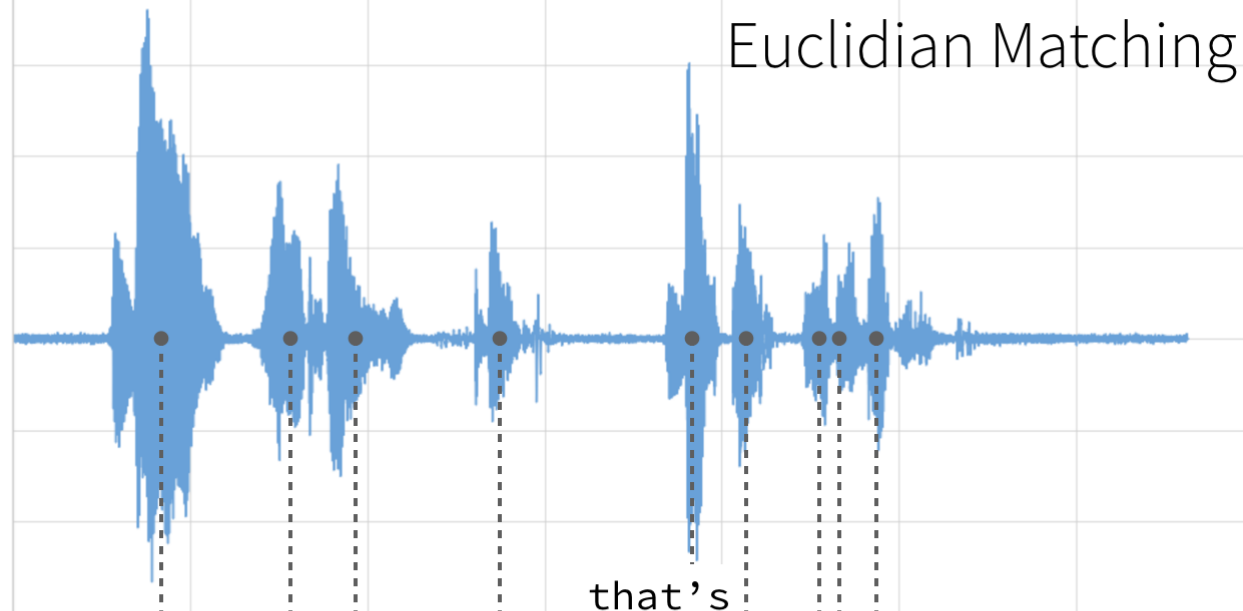


Euclidean Matching



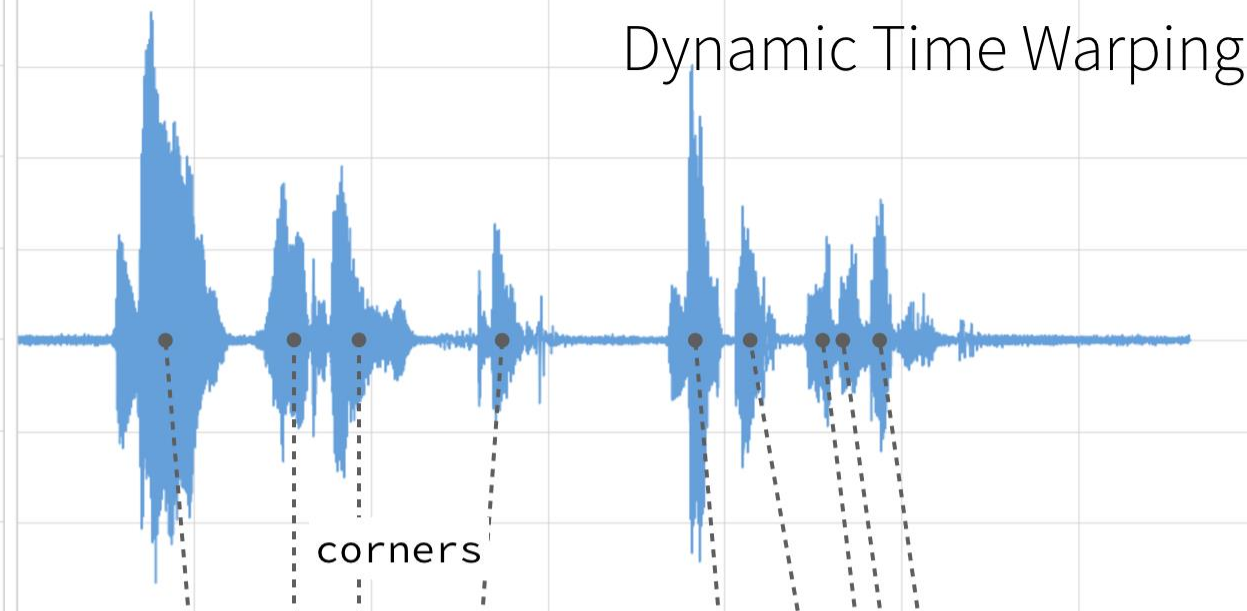
Dynamic Time Warping Matching

Euclidian Matching



doors and kid
corners
that's where they get
you

Dynamic Time Warping

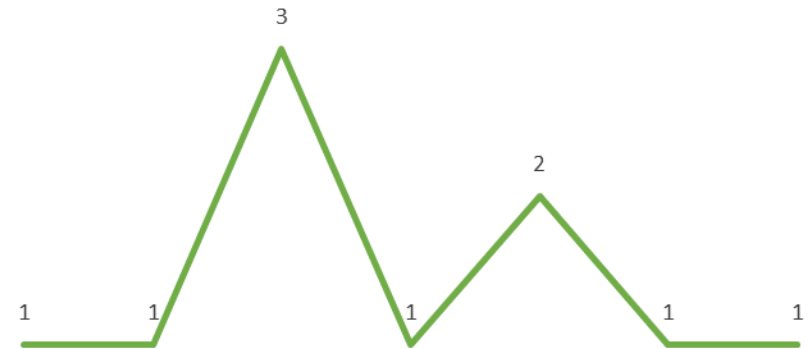
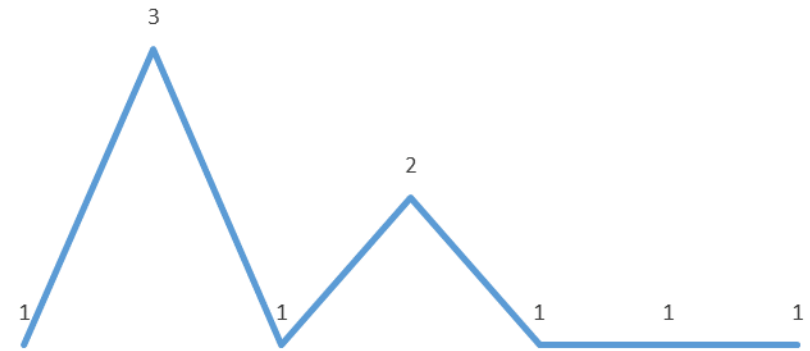


doors and kid
corners
that's where they get
you

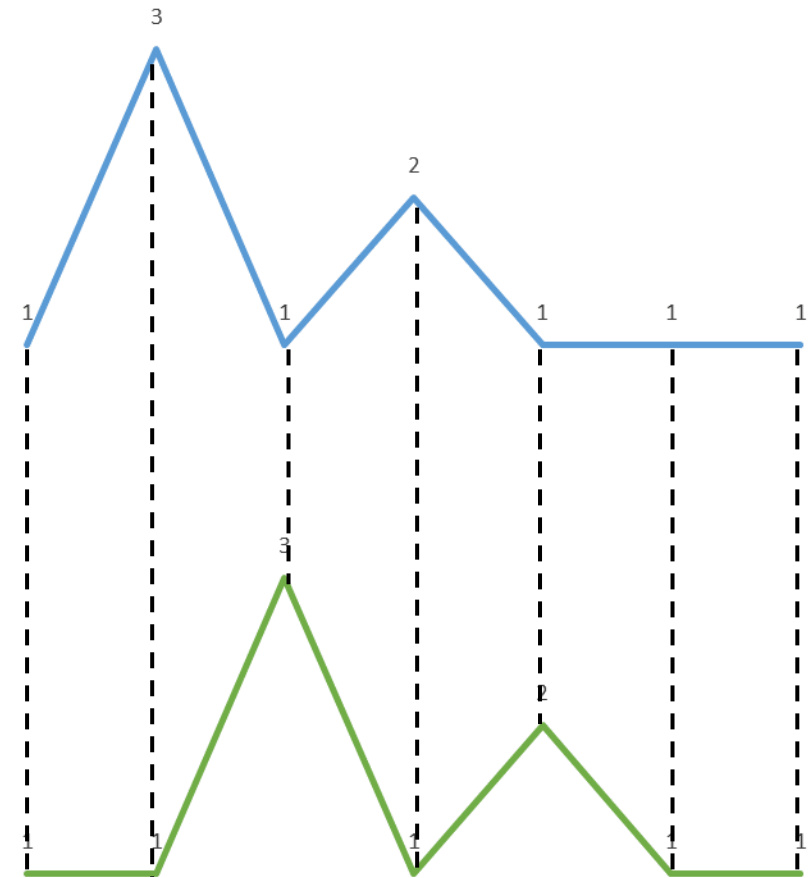
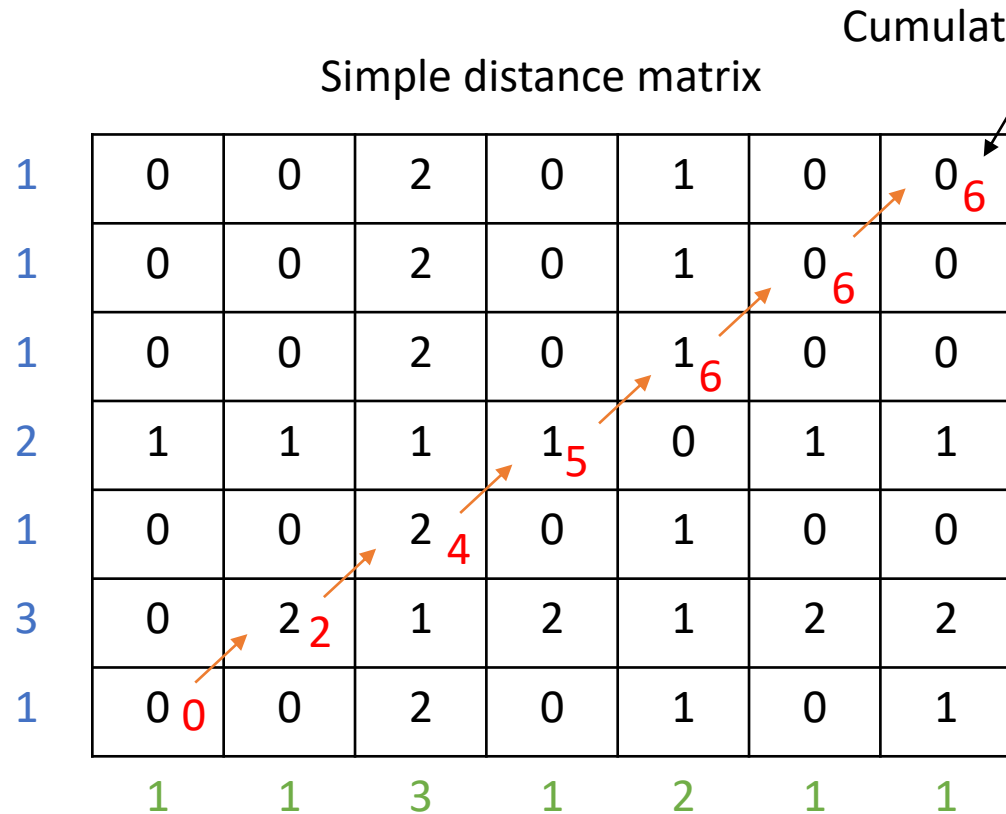
We start with a simplified example for clarity

Simple distance matrix

1	0	0	2	0	1	0	0
1	0	0	2	0	1	0	0
1	0	0	2	0	1	0	0
2	1	1	1	1	0	1	1
1	0	0	2	0	1	0	0
3	0	2	1	2	1	2	2
1	0	0	2	0	1	0	1
	1	1	3	1	2	1	1



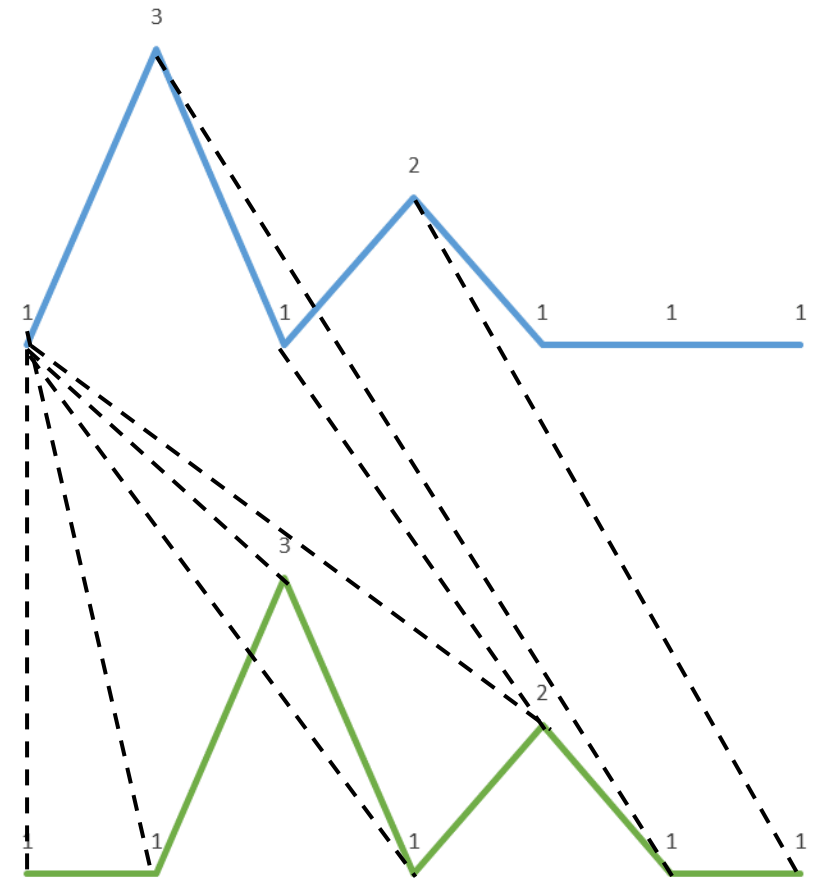
Simple distance measures do not cut it



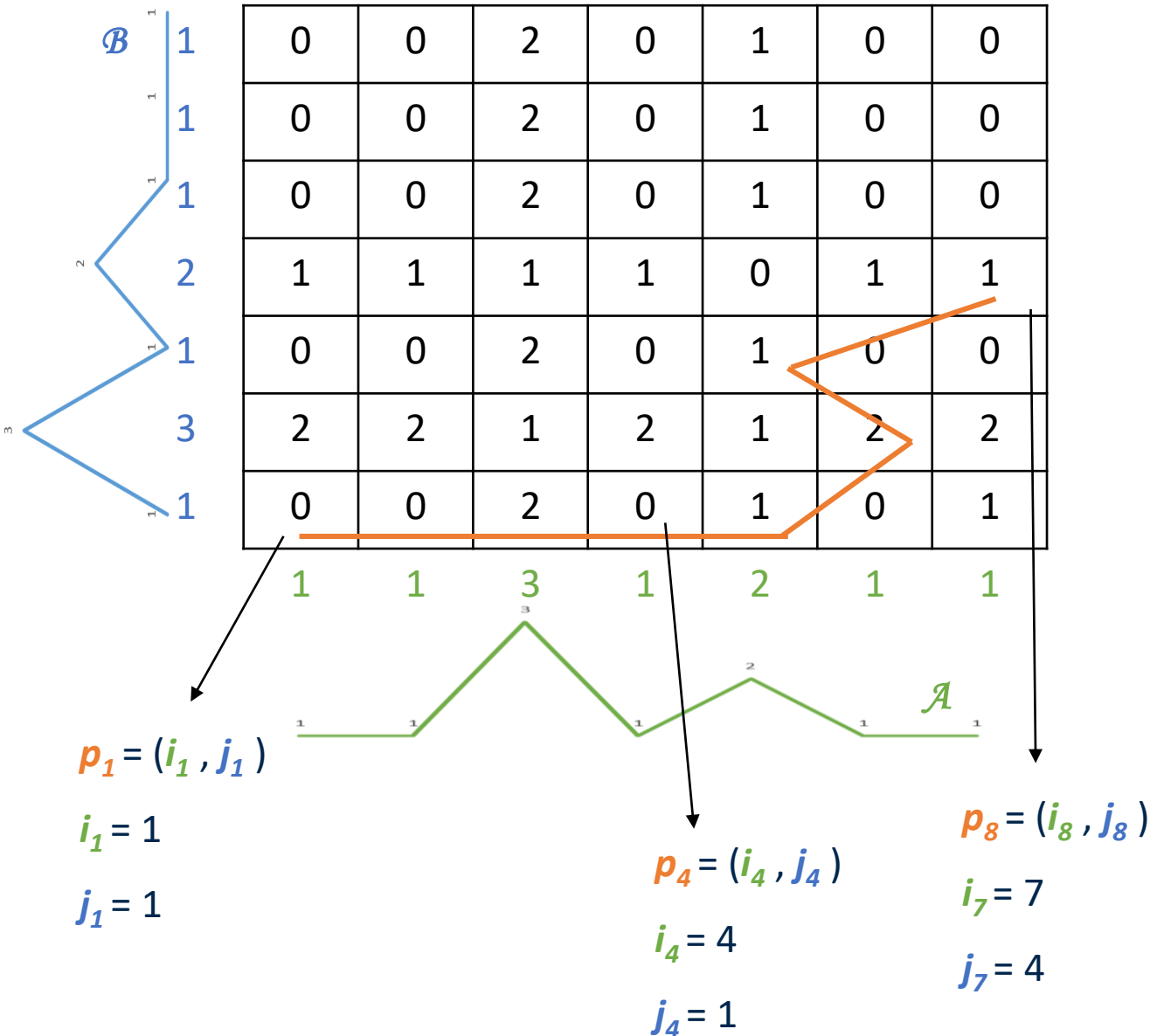
What if we try another path?

Simple distance matrix

1	0	0	2	0	1	0	0
1	0	0	2	0	1	0	0
1	0	0	2	0	1	0	0
2	1	1	1	1	0	1	17
1	0	0	2	0	1	6	0
3	2	2	1	2	1	2	5
1	0	0	2	0	1	3	0
	1	1	3	1	2	1	1



Simple distance matrix



To find the *best alignment* between \mathcal{A} and \mathcal{B} one needs to find the path through the grid

$$P = p_1, \dots, p_s, \dots, p_k$$

$$p_s = (i_s, j_s)$$

which *minimizes* the total distance between them.

P is called a warping function.

Time-normalized distance between \mathcal{A} and \mathcal{B} :

$$D(\mathcal{A}, \mathcal{B}) = \left[\frac{\sum_{s=1}^k d(p_s) \cdot w_s}{\sum_{s=1}^k w_s} \right]$$

$d(p_s)$: distance between i_s and j_s

$w_s > 0$: weighting coefficient.

Best alignment path

between \mathcal{A} and \mathcal{B} :

$$P_0 = \arg \min_P (D(\mathcal{A}, \mathcal{B}))$$

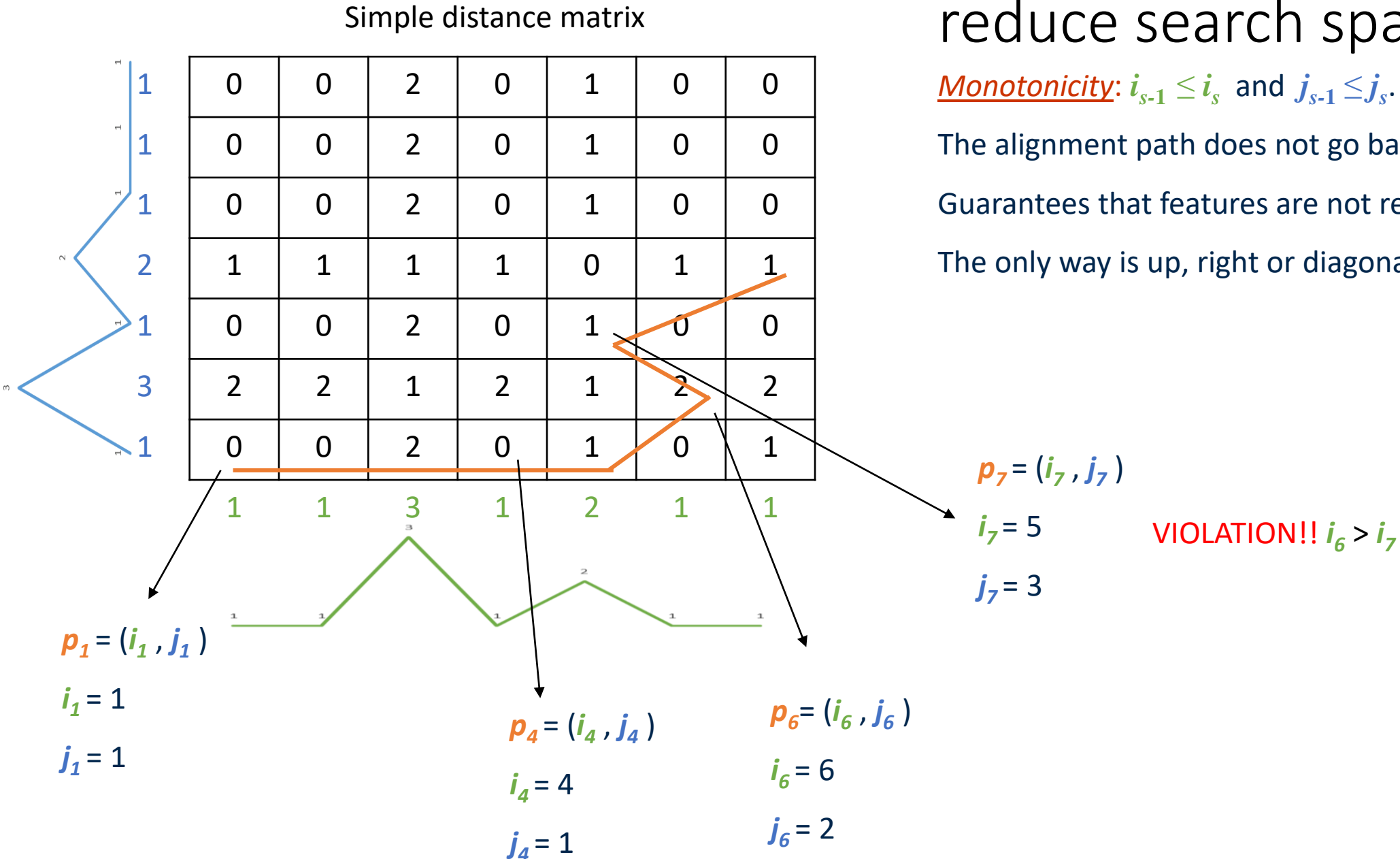
We set some constraints to reduce search space

Monotonicity: $i_{s-1} \leq i_s$ and $j_{s-1} \leq j_s$.

The alignment path does not go back in “time” index.

Guarantees that features are not repeated in the alignment.

The only way is up, right or diagonally up-right.



We set some constraints to reduce search space

Monotonicity: $i_{s-1} \leq i_s$ and $j_{s-1} \leq j_s$.

The alignment path does not go back in “time” index.

Guarantees that features are not repeated in the alignment.

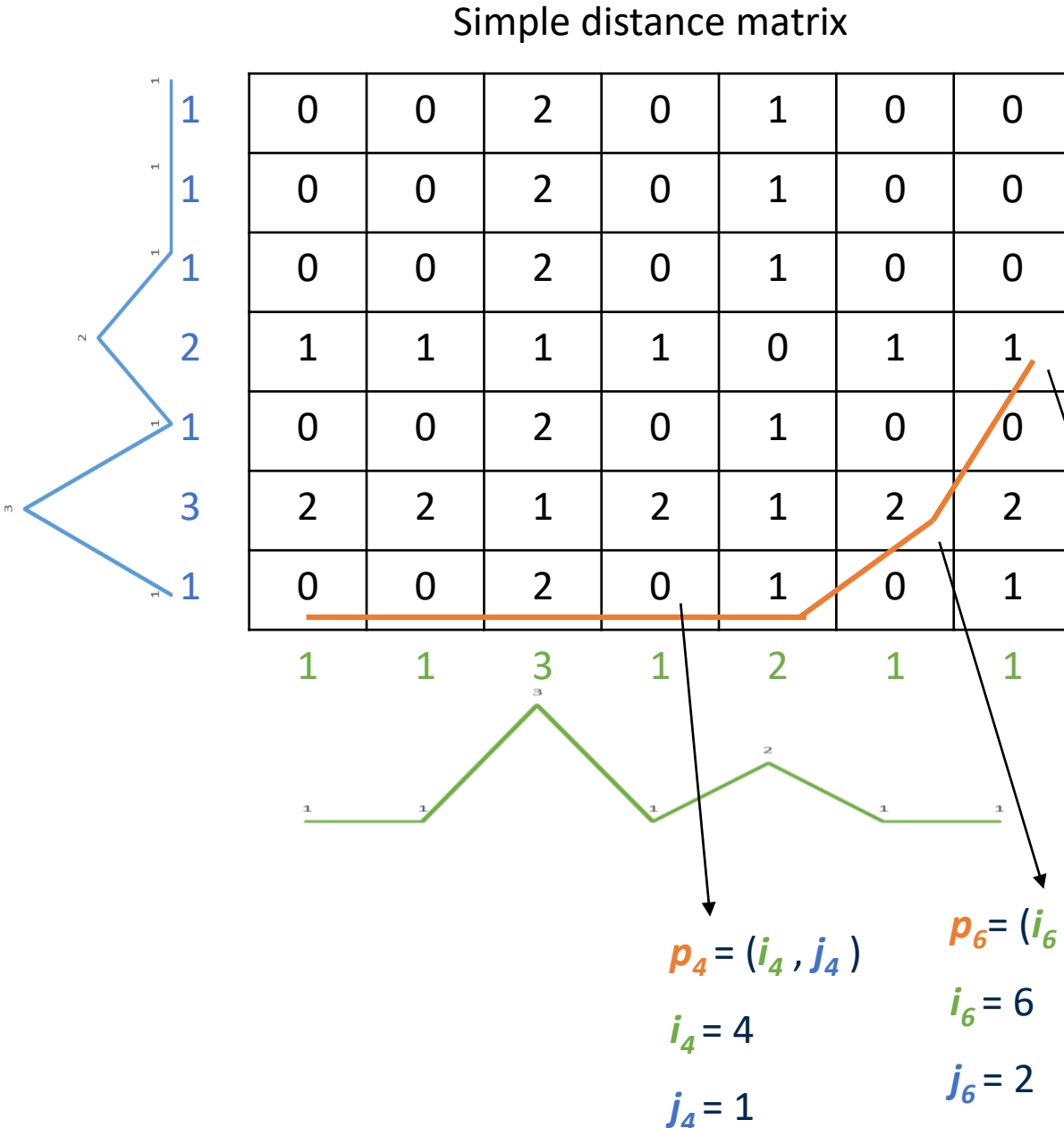
The only way is up, right or diagonally up-right.

Continuity: $i_s - i_{s-1} \leq 1$ and $j_s - j_{s-1} \leq 1$.

The alignment path does not jump in “time” index.

The only way up, right or diagonally up-right by one-unit.

Guarantees that the alignment does not omit important features.



We set some constraints to reduce search space

Monotonicity: $i_{s-1} \leq i_s$ and $j_{s-1} \leq j_s$.

The alignment path does not go back in “time” index.

Guarantees that features are not repeated in the alignment.

The only way is up, right or diagonally up-right.

Continuity: $i_s - i_{s-1} \leq 1$ and $j_s - j_{s-1} \leq 1$.

The alignment path does not jump in “time” index.

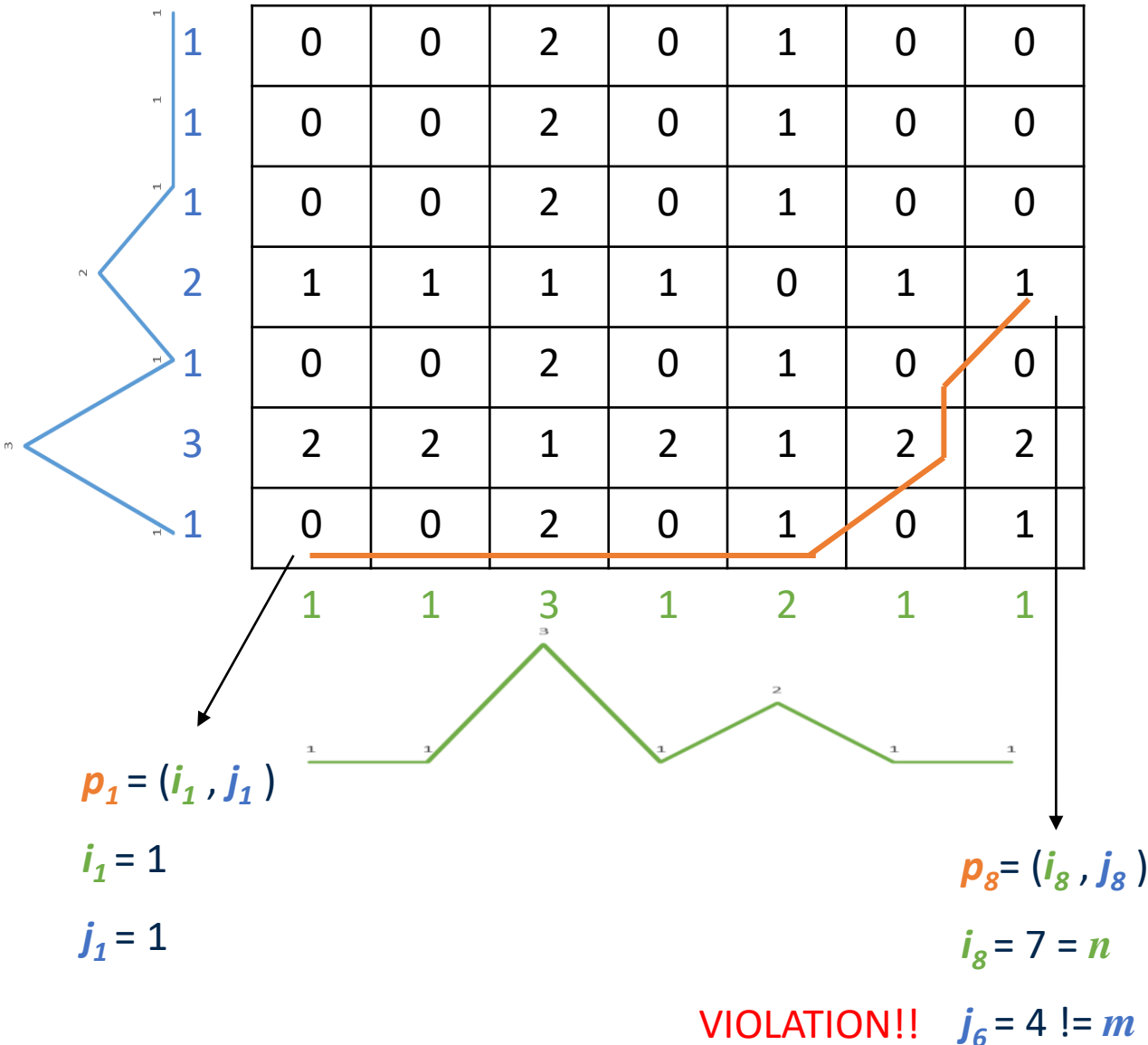
The only way up, right or diagonally up-right by one-unit.

Guarantees that the alignment does not omit important features.

Boundary Conditions: $i_1 = 1, i_k = n$ and $j_1 = 1, j_k = m$.

The alignment path starts at the bottom left and ends at the top right.

Simple distance matrix



We set some constraints to reduce search space

Monotonicity: $i_{s-1} \leq i_s$ and $j_{s-1} \leq j_s$.

The alignment path does not go back in “time” index.

Guarantees that features are not repeated in the alignment.

The only way is up, right or diagonally up-right.

Continuity: $i_s - i_{s-1} \leq 1$ and $j_s - j_{s-1} \leq 1$.

The alignment path does not jump in “time” index.

The only way up, right or diagonally up-right by one-unit.

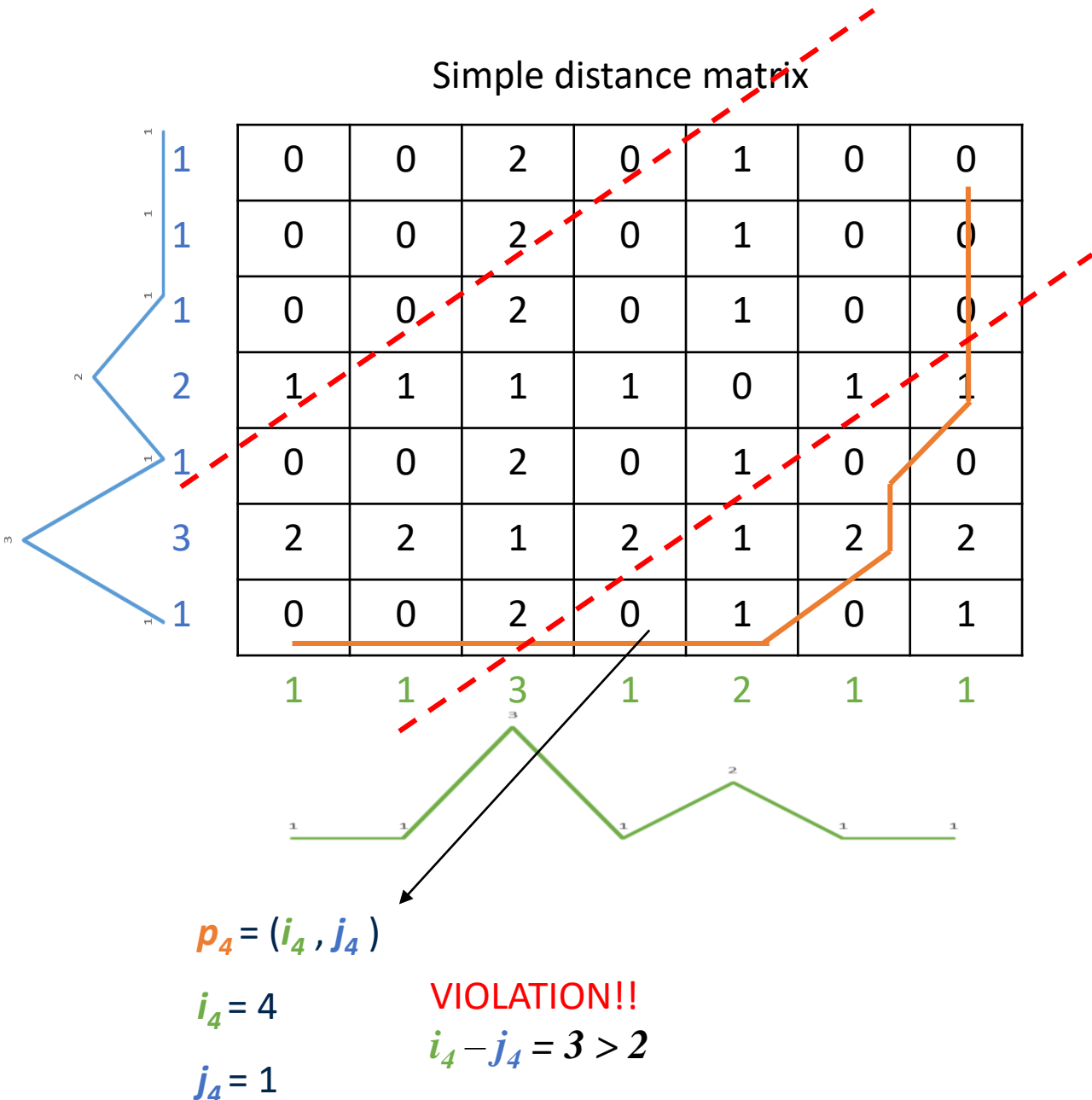
Guarantees that the alignment does not omit important features.

Boundary Conditions: $i_1 = 1, i_k = n$ and $j_1 = 1, j_k = m$.

The alignment path starts at the bottom left and ends at the top right.

Warping Window: $|i_s - j_s| \leq r$, where $r > 0$ is the window length.

Example: $r = 2$



We need to seek a weighting coefficient which is independent of warping function

Recall that time-normalized distance between \mathcal{A} and \mathcal{B} :

$$D(\mathcal{A}, \mathcal{B}) = \min_P \left[\frac{\sum_{s=1}^k d(p_s) \cdot w_s}{\sum_{s=1}^k w_s} \right].$$

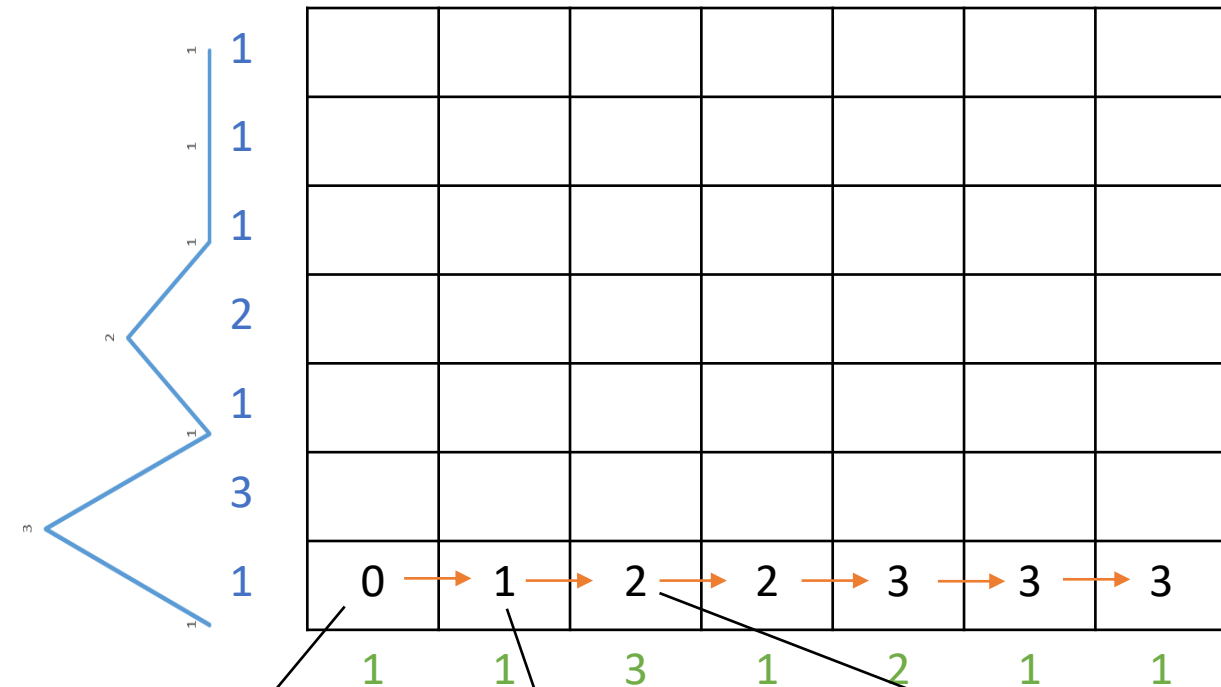
$$C = \sum_{s=1}^k w_s$$

$$D(\mathcal{A}, \mathcal{B}) = \frac{1}{C} \min_P \left[\sum_{s=1}^k d(p_s) \cdot w_s \right]$$

For simplicity sake, we take w_s to be the number of steps taken in the right, left, or upwards direction (quasi-symmetric). There are other ways to weigh the distance, but we will not discuss them here today 😊

Back to the simple example – we use dynamic programming to solve for the lowest distance

TOTAL distance matrix



$$g(1, 1) =$$

$$d(1, 1) = |1 - 1|$$

$$g(2, 1) = g(1, 1) + d(2, 1) = 0 + 0 = 0$$

$$g(3, 1) = g(2, 1) + d(3, 1) = 0 + |3 - 1| = 2$$

We ignore warping window for now:

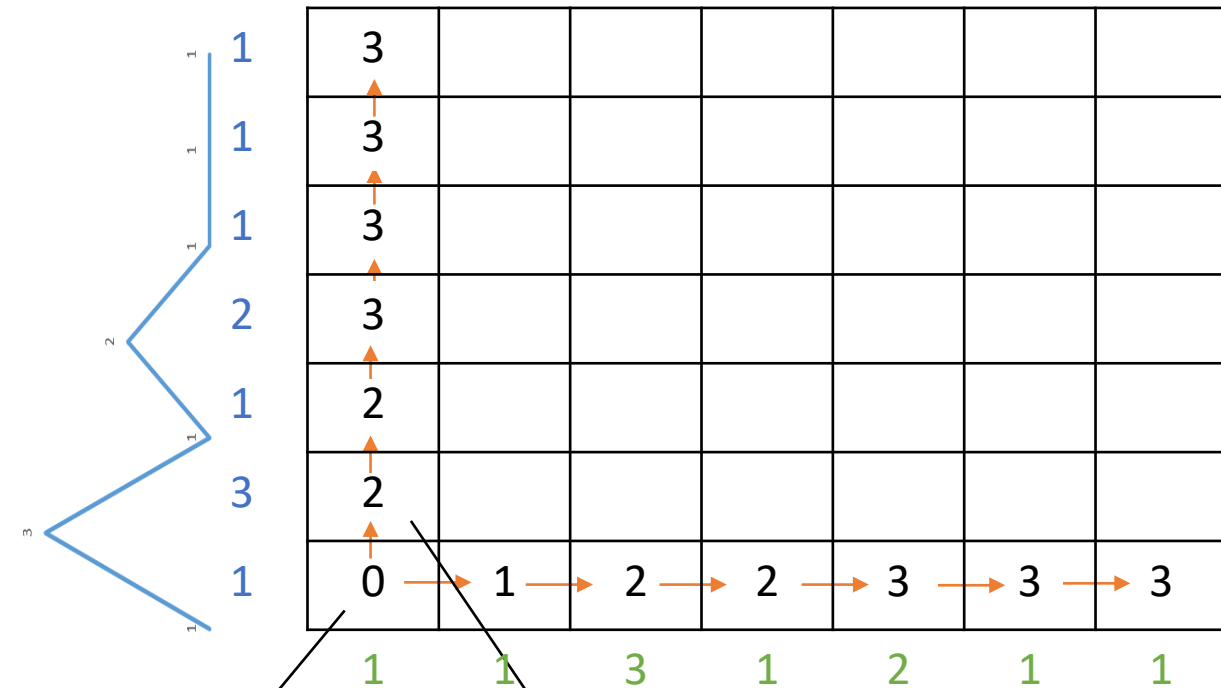
Initial condition: $g(1, 1) = d(1, 1)$.

DP-equation:

$$g(i, j) = \min \begin{pmatrix} g(i, j-1) + d(i, j) \\ g(i-1, j-1) + d(i, j) \\ g(i-1, j) + d(i, j) \end{pmatrix}$$

Back to the simple example – we use dynamic programming to solve for the lowest distance

TOTAL distance matrix



$$g(1, 1) =$$

$$d(1, 1) = |1 - 1|$$

$$g(1, 2) = g(1, 1) + d(1, 2) = 0 + |3 - 1| = 2$$

We ignore warping window for now:

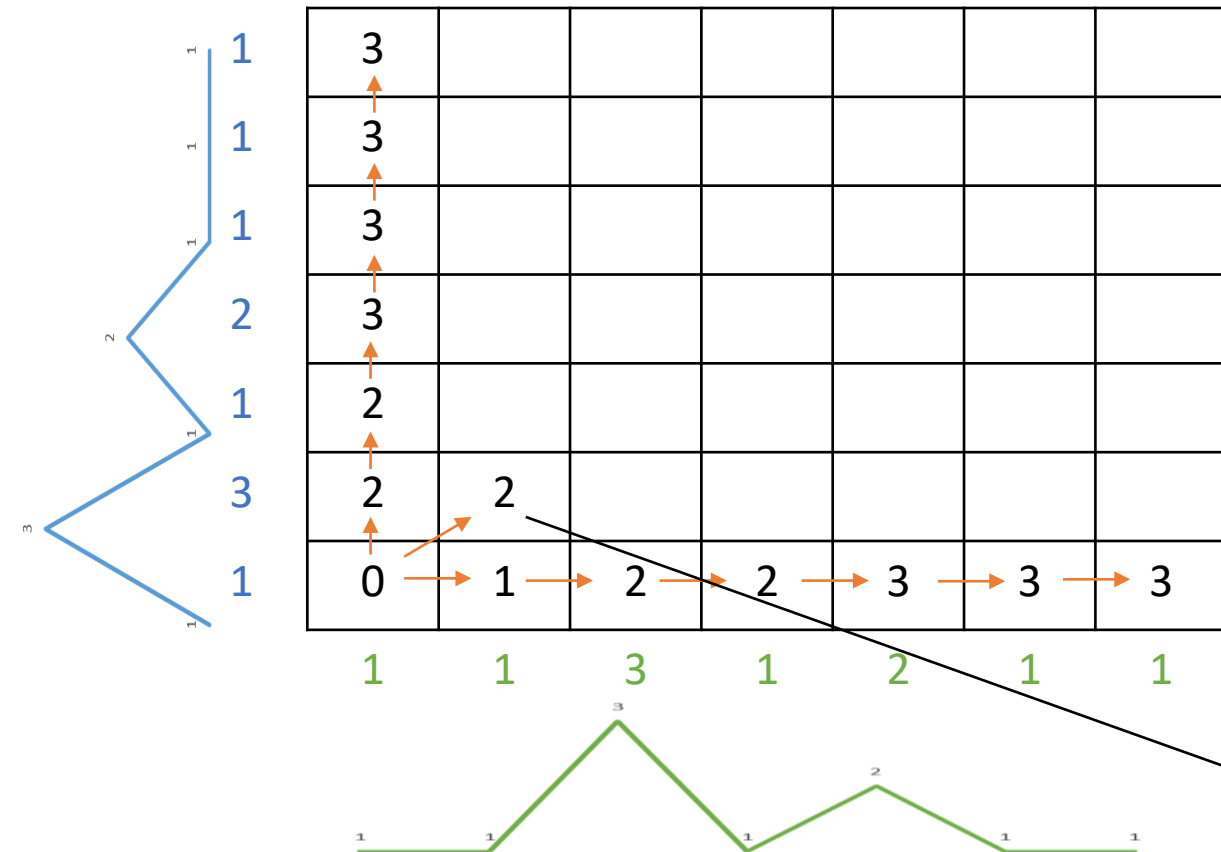
Initial condition: $g(1, 1) = d(1, 1)$.

DP-equation:

$$g(i, j) = \min \begin{pmatrix} g(i, j-1) + d(i, j) \\ g(i-1, j-1) + d(i, j) \\ g(i-1, j) + d(i, j) \end{pmatrix}$$

Back to the simple example – we use dynamic programming to solve for the lowest distance

TOTAL distance matrix



We ignore warping window for now:

Initial condition: $g(1,1) = d(1,1)$.

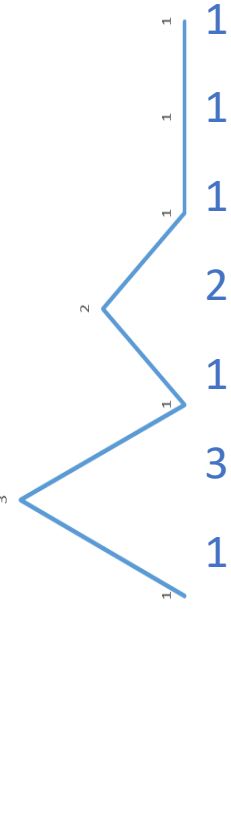
DP-equation:

$$g(i,j) = \min \begin{pmatrix} g(i,j-1) + d(i,j) \\ g(i-1,j-1) + d(i,j) \\ g(i-1,j) + d(i,j) \end{pmatrix}$$

$$\begin{aligned} g(2,2) &= \min[g(1,1), g(1,2), g(2,1)] + d(2,2) \\ &= \min(0, 2, 1) + |3-1| = 2 \end{aligned}$$

Back to the simple example – we use dynamic programming to solve for the lowest distance

TOTAL distance matrix



3	3	5	2	3	3	1
3	3	5	2	3	3	1
3	3	5	2	2	1	1
3	3	3	2	1	2	3
2	2	3	1	2	2	2
2	2	1	3	3	5	5
0	1	2	2	3	3	3

1 1 3 1 2 1 1

We ignore warping window for now:

Initial condition: $g(1,1) = d(1,1)$.

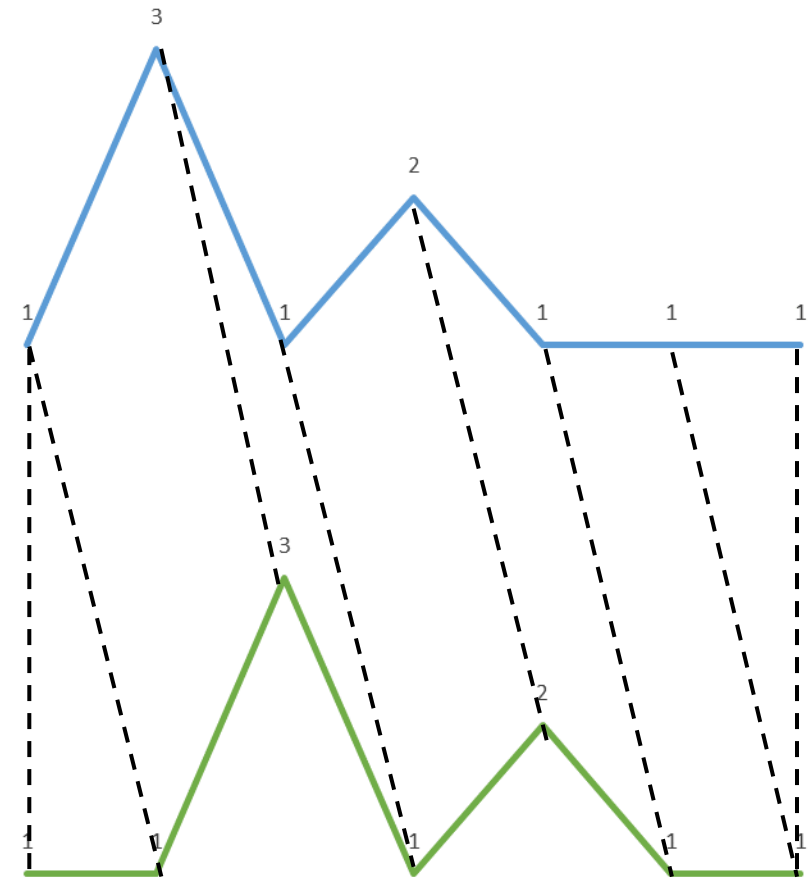
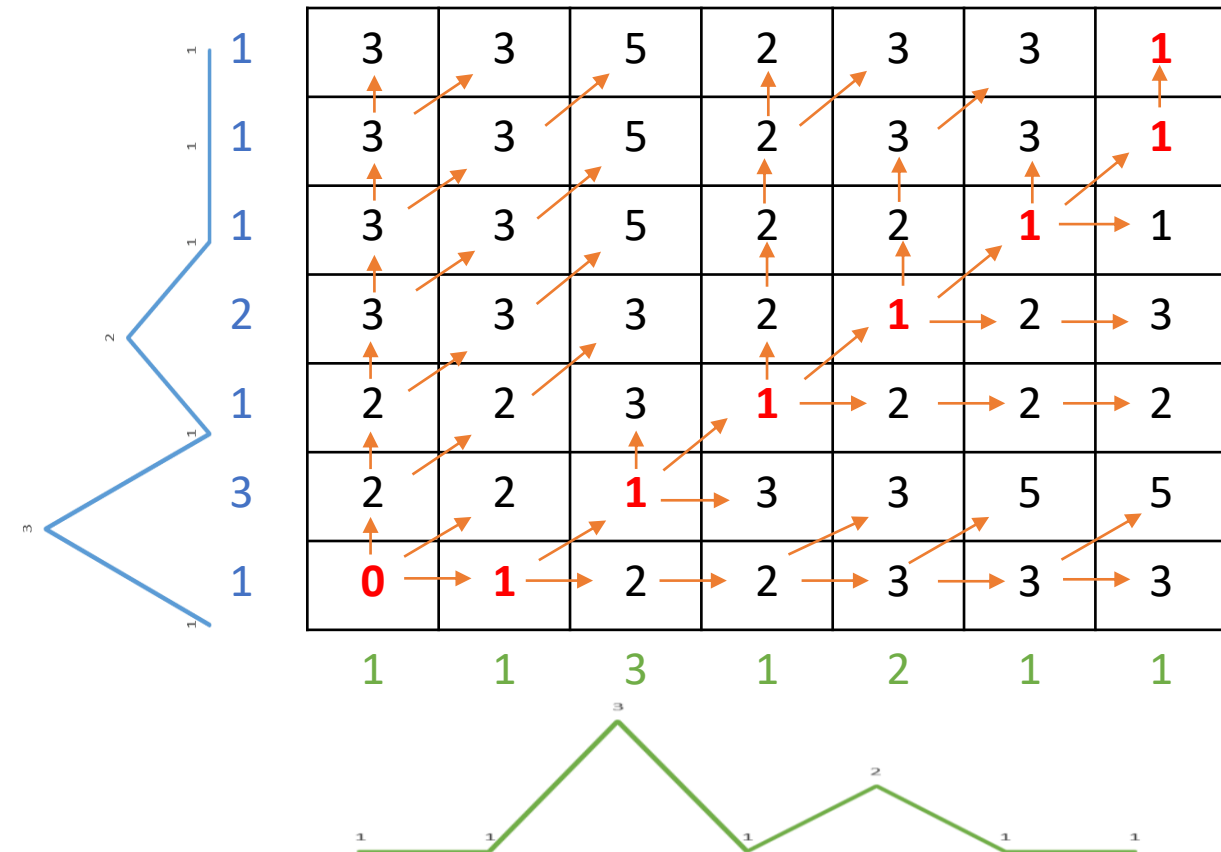
DP-equation:

$$g(i,j) = \min \begin{pmatrix} g(i,j-1) + d(i,j) \\ g(i-1,j-1) + d(i,j) \\ g(i-1,j) + d(i,j) \end{pmatrix}$$

$$\begin{aligned} g(4,3) &= \min[g(3,2), g(4,2), g(3,3)] + d(4,3) \\ &= \min(1, 3, 3) + |1-1| = 1 \end{aligned}$$

Back to the simple example – we use dynamic programming to solve for the lowest distance

TOTAL distance matrix



DTW is costly. Time and space complexity are both $O(n^2)$

- Recently, Gold and Shafir (2018) broke this quadratic barrier by presenting a deterministic algorithm that calculates DTW distance in $O(n^2 \log \log \log n / \log \log n)$ time.
- There are other libraries that we can use such as FastDTW, which provides an approximate Dynamic Time Warping (DTW) algorithm that provides optimal or near-optimal alignments with an $O(n)$ time and memory complexity.
- And that is only comparing 2 time series. What if we have $m = 50K$ time series to compare?
- That is equals to $50000C2 = 1249975000 =$ quite a lot of time series to compare.

In my previous project, I wanted to use DTW distance to cluster 100K time series. But SWG kept breaking because the memory required is too high

- So I did the one thing I knew how to
- I turned to the smartest statistician I know in the team and asked him for advice.
- His advice was to calculate DTW distance for a subset (~40K) time series, cluster them, find the centroid for every cluster.
- For the other 60k, put them in the cluster in which the centroid has the lowest DTW distance (1-NN).
 - Which means that for this 60k, I am only comparing 60k X (no of centroids) time series.