Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching

INFO-H509 - Geo-Spatial and web technologies

Gaspard Merten, Adel Nehili, Allan Noubissi Kamgang, Gregory Coolen

Université Libre de Bruxelles

May 11, 2024

Table of contents

Introduction

Université Libre de Bruxelles

Context

- Searching for similar time series in databases:
    - Identifying moving objects $\rightarrow$ Searching for black holes
    - Medicine $\rightarrow$ Finding time series discords for anomaly detection
    - Data mining $\rightarrow$ Speach recognition
- Measures of (dis)similarity
    - Non-elastic *metrics* from the L_p space: **Euclidian Distance** (ED) but also Manhattan, Minkowski, Infinite Norm, ...
    - An elastic similarity *measure* (that is not a metric) supporting time shifts: **Dynamic Time Warping** (DTW)
    - An elastic *metric* supporting time shifts: **Edit distance with Real Penalty** (ERP)

Time series matching: main algorithms

DTW

- About 60 y.o
- Adapted for time shifts
- Not a metric !

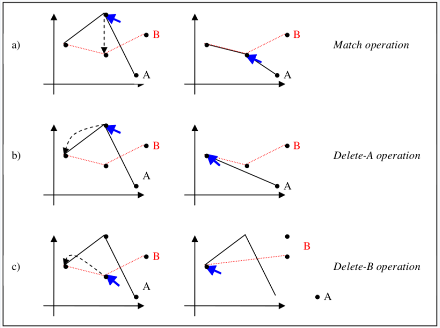
ERP

- About 20 y.o
- Uses edit distance while incorporating real costs proportional to edit
- Adapted for time shifts
- Is a metric !

Introducing TWED

Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching
Pierre-François Marteau (2007)

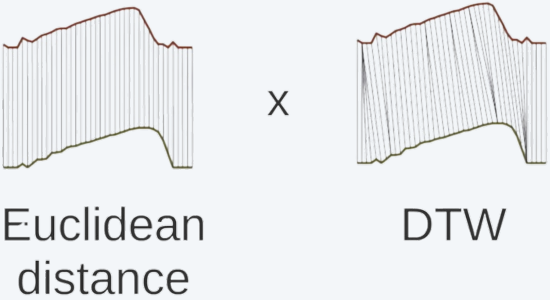
- Graphical analogy (cf. figure)
- Built on ERP
  - Combining L_p norms and Edit Distance
  - Supports time shifts
- Introducing *stiffness* parameter:
  - Drives *elasticity* of TWED
    $\rightarrow$ Trades off between DTW and ED



Background on Dynamic Time Warping (DTW)

Université Libre de Bruxelles

Background on Dynamic Time Warping (DTW)



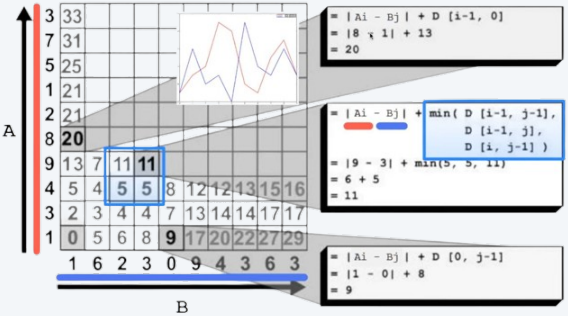
DTW Algo : All steps

1. **Initialization**
2. **Matrix Population**
3. **Path Finding**
4. **Output**

DTW Algo : Initialization & Matrix Population

1. **Initialization**: Given two sequences, X and Y, of lengths n and m respectively, DTW starts by creating an $n \times m$ matrix where the cell (i, j) **represents the distance between X_i and Y_j.**
2. **Matrix Population**: The DTW algorithm then populates this matrix by computing the **cumulative distance between all pairs of points**. The cumulative distance at (i, j) is calculated as the distance at (i, j) plus the minimum of the cumulative distances at $(i - 1, j)$ (previous point in X), $(i, j - 1)$, and $(i - 1, j - 1)$. By construction, this step ensures that the algorithm finds the path through the matrix that minimizes the total distance between the sequences.

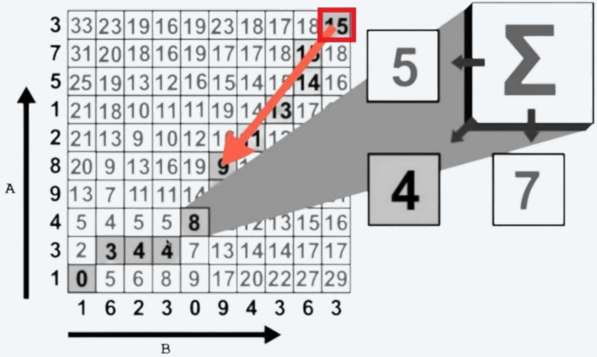
DTW Algo : Initialization & Matrix Population



DTW Algo : Path Finding & Output

1. **Path Finding**: After the matrix is fully populated, DTW finds the path **from** (n, m) **back to** $(1, 1)$ **that minimizes the total cumulative distance**. This path represents the optimal alignment between the two sequences.
2. **Output**: The total cumulative distance along this path is the DTW distance between the sequences, indicating how similar they are. The **path itself shows which elements in** X **are aligned with elements in** Y.

DTW Algo : Path Finding & Output



DTW properties

- Reflexivity : $d(X, X) = 0$
- Symmetry : $d(X, Y) = d(Y, X)$
- Non-negativity : $d(X, Y) \geq 0$
- **No** Triangle inequality : $d(X, Z) + d(Z, Y) \; ! \geq (X, Y)$

Background on Edit Distance with Real Penalty (ERP)

Université Libre de Bruxelles

Background on Edit Distance with Real Penalty (ERP)

		a	b	c	d	e	f
	$\boxed{0}$	97	196	297	400	505	612
a	97	$\boxed{0}$	98	197	297	398	500
z	244	122	$\boxed{24}$	121	219	318	418
c	297	221	123	$\boxed{24}$	122	221	321
e	404	301	224	125	$\boxed{25}$	122	222
d	500	401	303	225	125	$\boxed{26}$	$\boxed{124}$

Minimum Edit Distance = ERP – ”Real Penalty”

To make things **easier**, let’s start with **Minimum Edit Distance (arbitrary shorted to MED)**.
The Minimum Edit Distance algorithm calculates the **smallest number of operations (edits) required to transform one string into another**.

Minimum Edit Distance = ERP - ”Real Penalty”

1. **Initialization**
2. **Matrix Setup**
3. **Matrix Population**
4. **Determining the Minimum Edit Distance**

MED Algo : Initialization & Matrix Setup

- **Initialization**:
  - Define two sequences, A and B, of lengths n and m respectively.
  - Create a matrix with dimensions $(n+1) \times (m+1)$ where the element at position (i,j) will represent the minimum edit distance between the first i characters of A and the first j characters of B.
- **Matrix Setup : First row/column of the matrix**:
  - First row is initialized to $0, 1, 2, \ldots, m$ representing the cost of **deleting characters from A to match B (the empty string ϵ)**.
  - First column is initialized to $0, 1, 2, \ldots, n$ representing the cost of **adding characters from B to the empty string ϵ to match B**.

MED Algo : Initialization & Matrix Setup

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	0	0	0	0	0	0
z	2	0	0	0	0	0	0
c	3	0	0	0	0	0	0
e	4	0	0	0	0	0	0
d	5	0	0	0	0	0	0

MED Algo : Matrix Population & Output

- **Matrix Population**:
  - For **each operation** (deleting, substituting, adding), assign a **uniform cost of 1**.
  - Do not consider the actual distance between the symbols during operations.
  - **If** $A_i = B_j$:
    - $M(i,j) = M(i-1,j-1)$
  - **Else**:
    - $M(i,j) = 1 + \min(M(i-1,j-1), M(i-1,j), M(i,j-1))$
- **Determining the Minimum Edit Distance**:
  - The value at matrix position (n, m) gives the minimum edit distance needed to transform A into B, reflecting the least operations required.

MED Algo : Matrix Population & Output

		a	b	c	d	e	f
	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
z	2	1	1	2	3	4	5
c	3	2	2	1	2	3	4
e	4	3	3	2	2	2	3
d	5	4	4	3	2	3	3

ERP Algorithm

Since we developped the MED Algo, let's finish with **ERP**.
The ERP Algo take into account the **distance between the current symbols** but also the **distance of a pontential symbol to delete/add to the "gap value"**.
The "gap value" in ERP is a **predefined constant that represents the cost of inserting or deleting a character relative to some baseline or neutral value**.

```
# Populate the distance matrix
for i in range(1, n + 1):
    for j in range(1, m + 1):
        cost_sub = abs(ord(source[i - 1]) - ord(target[j - 1]))
        cost_del = abs(ord(source[i - 1]) - gap_value)
        cost_ins = abs(ord(target[j - 1]) - gap_value)


        dist[i][j] = min(dist[i - 1][j] + cost_del,    # Deletion, We use the horizontal path
                         dist[i][j - 1] + cost_ins,    # Insertion, We use the vertical path
                         dist[i - 1][j - 1] + cost_sub) # Substitution, We use the diaganole path
```

ERP Algo : Initialization & Matrix Setup

		a	b	c	d	e	f
	0	97	195	294	394	495	597
a	97	0	0	0	0	0	0
z	219	0	0	0	0	0	0
c	318	0	0	0	0	0	0
e	419	0	0	0	0	0	0
d	519	0	0	0	0	0	0

Figure: ERP with UTF-16

ERP Algo : Matrix Population & Output

		a	b	c	d	e	f
	0	97	195	294	394	495	597
a	97	0	98	197	297	398	500
z	219	122	24	121	219	318	418
c	318	221	123	24	122	221	321
e	419	322	224	125	25	122	222
d	519	422	324	225	125	26	124

Figure: ERP with UTF-16

Time Warp Edit Distance

Université Libre de Bruxelles

Time Warp Edit Distance (TWED)

- Previous methods for time series distance computation were no metrics *(DTW)*, or did not take into account the **Temporal dimension** in the *cost* computation.
- TWED answers that by **revisiting** the **edit operations and their associated cost** of *Levenshtein distance*:
  - *Delete_A* operation, *Delete_B* operation, *Match* operation, operations can thus be applied to both time series.
  - Cost functions integrating time and spatial distances, in part provider by a new *Stiffness parameter*
- For the next few slides, assume that we investigate TWED using two timseries A and B, of which each element is denoted a_i/b_i.

TWED: Delete A/B

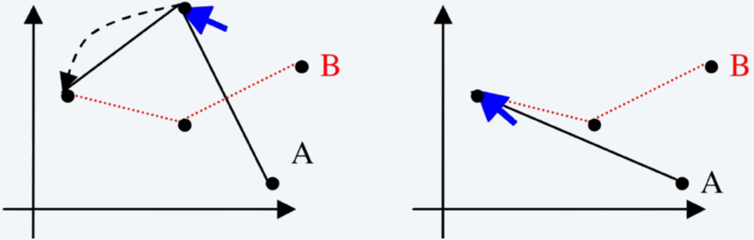


Figure: The Delete operation applied to time serie A

- The operation concists in dragging point a_i to $a_i - 1$
- The cost is the length of the vector (a_i, a_{i-1}) increased by an extra penalty $\lambda > 0$

TWED: Match A to B

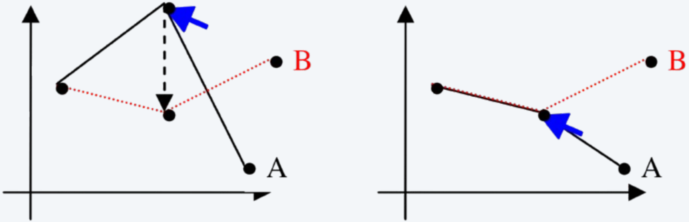


Figure: The Match operation applied to both time series

- The operation consists in dragging point a_i to b_j
- The associated cost is the length of the vector (a_i, b_j)

TWED: The game point-of-view

- TWED can be seen as a game where the goal is to fully superimpose time series A and B.
- Given the three previously defined operations, one must edit both A and B in order for them to be equals.
- Once an operation has been applied, the point to which it has been applied, as well as previous ones, can no longer be used.
- The winner of the game is the one who minimizes the sum of the cost of all the applied operations.
- There are $m.n$ (where $m = |A|$ and $n = |B|$) possibilities, thankfully we have computers.

TWED: Recursion algorithm

$$\delta_{TWED}(A_1^p, B_1^q) = Min\begin{cases} \delta_{TWED}(A_1^{p-1}, B_1^q) + d(a'_p, a'_{p-1}) + \lambda \\ \delta_{TWED}(A_1^{p-1}, B_1^{q-1}) + d(a'_p, b'_q) \\ \delta_{TWED}(A_1^p, B_1^{q-1}) + d(b'_{q-1}, b'_q) + \lambda \end{cases}$$

Figure: The proposed recursion algorithm to compute TWED

- At each step, we pick the operation that minimizes the most the distance between A and B
- Notice that switching A with B does not seem to affect the measure

TWED: The missing part

- Until now, we did not define the distance ($d(a_i, b_j)$) that we used in our algorithm.
- In order to take both time and space into account, one must conciliate both in one measure.
- The approach taken by TWED is to apply euclidian distance to both and to sum them together with a proportionality factor γ for the time part.

$$d(a',b') = d_{LP}(a,b) + \gamma . d_{Lp}(t_a, t_b)$$

Figure: The distance formula with stiffness as γ

TWED: Stiffness ?

- The notion of **Stiffness** can appear a bit obscure at first.
- In fact it should be seen as a measure of how much the distance measure is deformed by time difference between both time series.
- An euclidian distance would have an infinite Stiffness while DTW would have a null-stiffness.
- γ does provide a way to configure the Stiffness of TWED.
- Note that γ shall be greater than 0 for TWED to be a distance on *SxT*

TWED: Putting it all together

- We defined a distance measure between two time series.
- It is sensitive to both **Spatial and Temporal** differences thanks to the introduction of *Stiffness*.
- It can be demonstrated that TWED is a distance under the usual definition.
- TWED can be **parameterized** by λ and γ.
- TWED has a **time complexity** of $O(m * n)$, just like ERP and DTW.

Experimental analysis & results

Université Libre de Bruxelles

Experimentation Setup

The authors used datasets from the **UCR** repository.

- **Goal**:Assess with a simple classification task the performance and practicality of TWED in handling time series data.
- **Data Split:** Each dataset is divided into a training subset and a testing subset.
- **Classification Rule:** The classification is conducted using a nearest neighbor approach; the category of an unknown time series from the test set is determined by the category of its closest neighbor in the train set.
- **Comparative methods:** Euclidean Distance (ED), Dynamic Time Warping (DTW), Optmized Dynamic Time Warping (ODTW) and Edit Distance with Real Penalty (ERP).
- **Adjustment for the Experiment:** Given the absence of explicit time stamps in **UCR** datasets, sample indices were used to simulate time values.

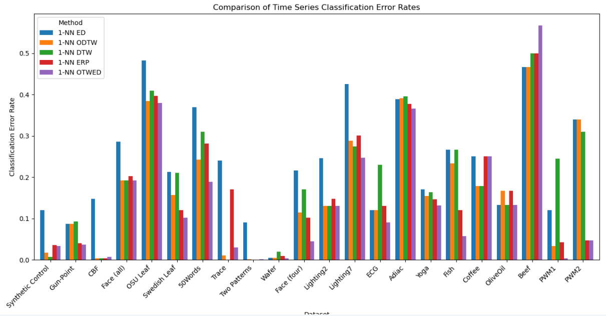
Parameters Adjustments

$$\delta_{TWED}(A_1^p, B_1^q) = Min\begin{cases} \delta_{TWED}(A_1^{p-1}, B_1^q) + \left|a_p, a_{p-1}\right| + \gamma + \lambda \\ \delta_{TWED}(A_1^{p-1}, B_1^{q-1}) + \left|a_p, b_q\right| + \gamma.\left|p-q\right| \\ \delta_{TWED}(A_1^p, B_1^{q-1}) + \left|b_{q-1}, b_q\right| + \gamma + \lambda \end{cases}$$

Figure: The proposed implementation of TWED

- **Parameter Tuning:** Optimization of TWED parameters γ and λ was achieved using a leave-one-out cross-validation on the training data to minimize classification errors. 'stiffness values' (γ) are selected into $\{1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}, 1e^{-1}, 1\}$ and λ is selected into $\{1.0, 0.75, 0.5, 0.25, 0\}$
- **OTWED:** Is the optimized version of TWED used with optimal γ and λ parameters.

Results



- TWED generally outperforms or matches the performance of DTW and ERP across most datasets.
- **Limitations**: TWED requires careful tuning of parameters to be reliable.

Conclusion

- **Advantages Over ERP and DTW:** TWED bridges the gap between Euclidean distances and Dynamic Time Warping (DTW)
- **Performance:** At least one version of TWED outperforms in average ERP and DTW.
- **Sensitivity to parameters:** Classification error rates show significant sensitivity to parameters.
- **Potential Application:** Domains requiring robustness to time shifts, such as audio processing, financial data analysis, and biomedical signal analysis.
- **Future Directions:** Explore automated methods for selecting optimal parameters values, potentially through machine learning techniques.

Appendix

TWED: Distance proof sketch

Theorem 1: δ_{TWED} is a distance on the set of finite discrete time series U:

P1: $\delta_{TWED}(A, B) \geq 0$ for any finite discrete time series A and B,

P2: $\delta_{TWED}(A, B) = 0$ iff $A = B$ for any finite discrete time series A and B,

P3: $\delta_{TWED}(A, B) = \delta_{TWED}(B, A)$ for any finite discrete time series A and B,

P4: $\delta_{TWED}(A, B) \leq \delta_{TWED}(A, C) + \delta_{TWED}(C, B)$ for any finite discrete time series A, B and C.

TWED: Lambda - Deletion penalty

- Lambda is an additional constant that is added to the cost of a delete operation
- It is added to compensate for the inherent absence of an obvious distance measurement as in the case of the Match operation
- It should thus be seen as a way to adequately penalize the Delete operation in order to be faire to the match operation.

TWED: Recursion algorithm initialization

$$\delta_{TWED}(A_1^0, B_1^0) = 0$$

$$\delta_{TWED}(A_1^0, B_1^j) = \sum_{k=1}^{j} d(b'_k, b'_{k-1}), j \in \{1,..,q\}$$

$$\delta_{TWED}(A_1^i, B_1^0) = \sum_{k=1}^{i} d(a'_k, a'_{k-1}), i \in \{1,..,p\}$$

with $a'_0 = b'_0 = 0$ by convention.

Figure: The initialization values for the TWED algorithm

Gaspard Merten, Adel Nehili, Allan Noubissi Kamgang, Gregory Coolen

Université Libre de Bruxelles

May 11, 2024