



**Universität
Zürich^{UZH}**

Indexing Videos Containing Human Motion in the Form of Dance

Master Thesis

Institut für Informatik
der
Universität Zürich

Nicoletta Farabullini

Supervisor
Sven Helmer

Zürich
10.01.2022

Contents

1	Introduction	2
2	Previous work - MSc Project	2
3	Similarity measures	4
3.1	Proportionality of Euclidean Distance and Cosine Similarity . . .	5
4	Dynamic Time Warping (DTW)	6
4.1	DTW with Cosine Similarity	9
5	Multi-dimensional DTW	9
5.1	Accounting for missing values	11
5.2	Evaluation of similarity measure accuracy	12
5.3	DTW vertical for a multi-dimensional space	14
5.4	DTW with Euclidean distance vertical for a multi-dimensional space	14
5.5	Cosine Similarity vertical for a multi-dimensional space	17
5.6	Past and future vectors combinations	20
6	Horizontal analysis of angles vectors matrix	22
6.1	DTW horizontal for a multi-dimensional space	22
6.2	Cosine Similarity horizontal for a multi-dimensional space	25
7	LB_Keough for 1D	27
8	LB_Keough for a multi-dimensional space	30
8.1	Filtering dimensions for LB_Keough	32
9	Set Query video and create a candidate set	34
9.1	Candidate set with DTW only	34
9.2	Candidate set with DTW and LB_Keough	35
9.3	Candidate set with LB_Keough only	36
10	Vitrivr	37
11	Conclusions and Future Work	37

Master Thesis

Nicoletta Farabullini

January 19, 2022

1 Introduction

Videos have different lengths, hence number of frames. The longest amount is 133 frames whereas the shortest is 21.

2 Previous work - MSc Project

This thesis develops on a previously completed Master project. In this previous work, dance videos were divided into frames (25 per second), which were inputted in the Interpose library in Python to extract positions in space of joints. For each position in time, the angles between joints were then calculated. The output is a vector of angle for each frame:

1. angle from nose to neck to left shoulder,
2. angle from nose to neck to right shoulder,
3. angle from left shoulder to right shoulder,
4. angle from left shoulder to left upper arm,
5. angle from left lower arm to left upper arm,
6. angle from right upper arm to right shoulder,
7. angle from right upper arm to right lower arm,
8. angle from left eye to nose to left ear to eye,
9. angle from left eye to nose to neck,
10. angle from nose to neck to right eye to nose,
11. angle from left eye to nose to right eye to nose,
12. angle from right eye to nose to right ear to eye,
13. angle from right hip to right upper leg,

14. angle from right upper leg to right lower leg,
15. angle from left hip to left upper leg,
16. angle from left upper leg to left lower leg,
17. angle from left lower leg left ankle to heel,
18. angle from right lower leg to right ankle to heel,
19. angle from right foot to right toes,
20. angle from right foot to right lower leg,
21. angle from right foot to right ankle to heel,
22. angle from left foot to left lower leg,
23. angle from left foot to left ankle to heel,
24. angle from left foot to left toes,
25. angle from torso to right shoulder,
26. angle from torso to left shoulder,
27. angle from torso to nose to neck,
28. angle from torso to right hip,
29. angle from torso to left hip

Each vector of angles comprises of 29 entries. By combining all vectors, a data frame is obtained where each row is a vector of angles for a specific body part:

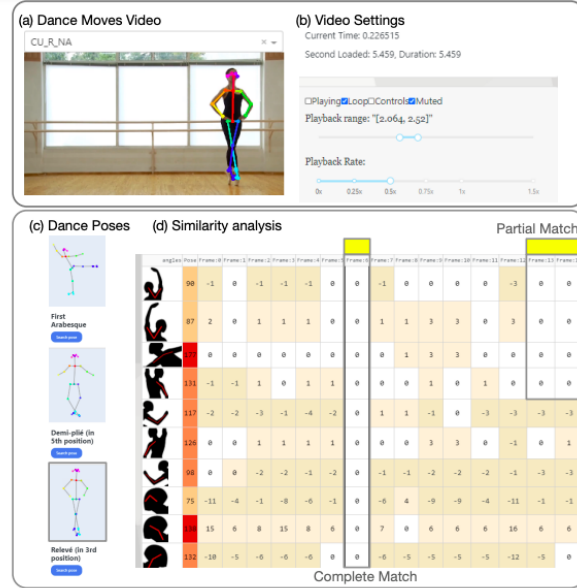


Figure 1: Dance poses matrix

Two data frames were compared against each other by first creating a DTW path comparing same body parts vectors of angles. Then, looping over the path, corresponding values in two data frames of two videos were used to compute cosine similarity and the result was appended to a list. The final similarity output was given by the mean of values in said list. Finally, a UI was created to view and compare videos.

Even though this work provided a solid beginning to the analysis of body movements in videos, further investigations need to be performed in regards to similarity measures and comparison of a query video with the dataset. To evaluate the validity of each similarity method, outputs were compared against a ground truth.

3 Similarity measures

There are different techniques that can be used to perform similarity measures among vectors of angles, e.g. $\vec{x} = x_1, x_2, \dots, x_n$ and $\vec{y} = y_1, y_2, \dots, y_n$. In particular, two methods were investigated:

Euclidean distance

$$ed(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)^2} \quad (1)$$

Cosine similarity

$$\text{cosSim}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| |\vec{y}|} = \frac{\sum_i \vec{x}_i \cdot \sum_i \vec{y}_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}} \quad (2)$$

These two methods can bring similar results as they can be made proportional to each other

3.1 Proportionality of Euclidean Distance and Cosine Similarity

Cosine Similarity and Euclidean distance can be re-arranged to be proportional. In fact, if we look at the squared Euclidean distance:

$$\text{ed}(x, y)^2 = (\sqrt{(x-y)^2})^2 = (x-y)^2 = \sum_i (x_i - y_i)^2 = \sum_i x_i^2 + \sum_i y_i^2 - \sum_i 2 * x_i \cdot y_i \quad (3)$$

If we now apply the same concept to with \hat{x} and \hat{y} being divided by their respective sums:

$$\text{ed}(\hat{x}, \hat{y})^2 = \sum_i \left(\frac{x_i}{\sqrt{\sum_i x_i^2}} - \frac{y_i}{\sqrt{\sum_i y_i^2}} \right)^2 \quad (4)$$

Folding it out:

$$\text{ed}(\hat{x}, \hat{y})^2 = \sum_i \left(\frac{x_i}{\sqrt{\sum_i x_i^2}} \right)^2 + \sum_i \left(\frac{y_i}{\sqrt{\sum_i y_i^2}} \right)^2 - \sum_i \frac{2 * x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} \quad (5)$$

Re-positioning summations:

$$\text{ed}(\hat{x}, \hat{y})^2 = \frac{\sum_i x_i^2}{\sum_i x_i^2} + \frac{\sum_i y_i^2}{\sum_i y_i^2} - 2 * \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} \quad (6)$$

The first two components of the sum in Equation(6) can be simplified to 1:

$$\text{ed}(\hat{x}, \hat{y})^2 = 2 - 2 * \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} = 2 * \left(1 - \frac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2} * \sqrt{\sum_i y_i^2}} \right) \quad (7)$$

Equation(6) is directly proportional to Equation(1):

$$\text{ed}(\hat{x}, \hat{y})^2 = 2 * (1 - \text{cosSim}(x, y)) \quad (8)$$

In other words, the Euclidean distance can be expressed in terms of cosine similarity if the two vectors are normalized to unit length.

These two techniques were then implemented in the Dynamic Time Warping (DTW) matrix to account for similar movements that might have not occurred at exactly the same time.

4 Dynamic Time Warping (DTW)

The Dynamic Time Warping (DTW) is a powerful technique when comparing time series that are out of sync, i.e where a one-to-one comparison of data points would not output the correct similarity measure. The classical DTW makes use of the Euclidean distance to compare entries of a time series against another. The resulting time complexity of $O(n^2)$.

For example, two time series $x = x_1, x_2, \dots, x_n$ and $y = y_1, y_2, \dots, y_n$ need to be compared against each other. They both have peaks of the same height occurring at different times:

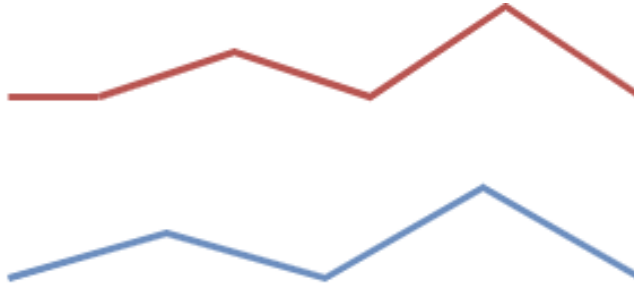


Figure 2: Time series

If only data points occurring at the same point in time were compared, low values for the Euclidean distance measures would be observed only in a few spots (in this case, the last three):

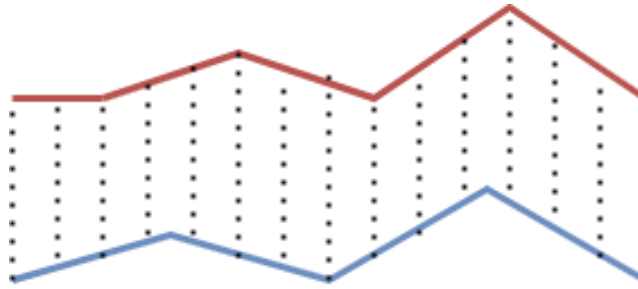


Figure 3: Time series comparison with basic Euclidean distance

Under these circumstances, the two series will not show much resemblance. However, a comparison using the DTW technique would highlight a higher number of similar data points even if happening at different points in time:

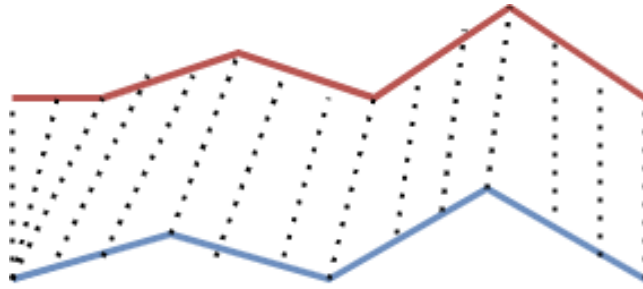


Figure 4: Time series comparison with DTW

This scenario brings a far more accurate similarity measure.

On a mathematical level, the DTW technique compares each point of the two time series, thus constructing a path of minimum distances.

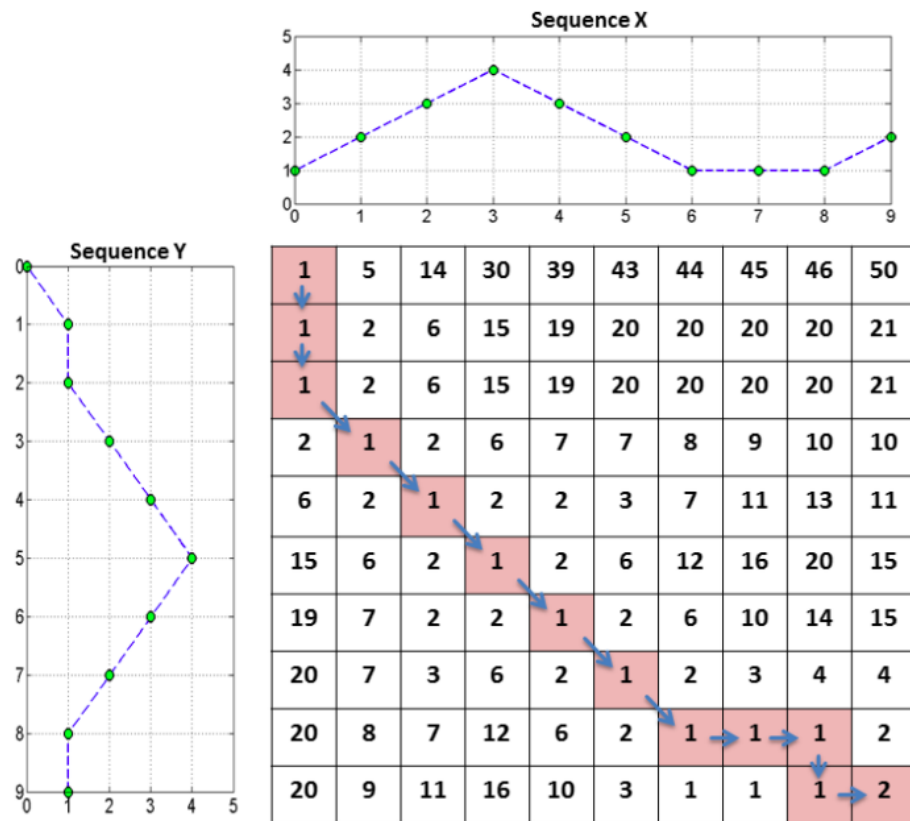
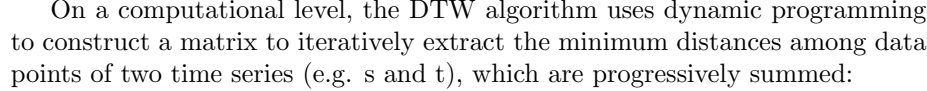


Figure 5: DTW matrix example

(<https://www.researchgate.net/figure/An-example-of-DTW->

shows the similarity between the sequences X and Y both of length 10


On a computational level, the DTW algorithm uses dynamic programming to construct a matrix to iteratively extract the minimum distances among data points of two time series (e.g. s and t), which are progressively summed:

Listing 1: DTW_for_1D_time_series

```
def dtw(s, t):
    n, m = len(s), len(t)
    # initialize DTW matrix
    dtw_matrix = np.zeros((n + 1, m + 1))
    for i in range(n + 1):
        for j in range(m + 1):
            dtw_matrix[i, j] = np.inf

    # set first value of matrix to 0
    dtw_matrix[0, 0] = 0

    # fill in matrix
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            cost = abs(s[i - 1] - t[j - 1])
            # take last min from a square box
            last_min = np.min([dtw_matrix[i - 1, j],
                               dtw_matrix[i, j - 1],
                               dtw_matrix[i - 1, j - 1]])
            dtw_matrix[i, j] = cost + last_min

    return dtw_matrix[n, m]
```

(https://en.wikipedia.org/wiki/Dynamic_time_warping)

The last entry of the DTW matrix is the final similarity measure between the two time series. If two time series are identical, this measure will output 0. On a more general level, the higher the measure the higher the dissimilarity of the two series.

This concept can be applied to this work as similar videos will have similar angles, e.g. 90 and 89 will still be evaluated as similar. However other methods can be used to calculate similarity of angles, such as Cosine Similarity. While the classical DTW implementation makes use of the Euclidean distance, other similarity methods can be used to calculate the DTW matrix. Because angles are an essential component of this project, it is paramount to investigate both similarity measures.

4.1 DTW with Cosine Similarity

Cosine similarity is a widely used technique to measure similarity between vectors of angles. Unlike Euclidean distance where values range from 0 to infinity, this method is bound to a range between 0 and 1. The more similar the two vectors are, the closer to 1 the similarity output will be. Consequently, to compare this method with Euclidean distance, it is paramount to subtract the final output of the former by 1.

To implement Cosine Similarity in the DTW algorithm, it only necessary to substitute the Euclidean distance portion:

Listing 2: DTW_cosSim_for_1D_time_series

```
def dtwcosSim(s, t):
    n, m = len(s), len(t)
    # initialize DTW matrix
    # in the same way as for Euclidean Distance

    # fill in matrix
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            cosSim = (s[i - 1] * t[j - 1]) / (n * m)
            cost = 1 - cosSim

    # fill in matrix
    # in the same way as for Euclidean Distance


    return dtw_matrix[n, m]
```

These algorithms are explicated for a case where only two time sequences need to be compared. However, the dance videos comprise of many of such sequences, thus requiring a shift to a multi-dimensional scenario.

5 Multi-dimensional DTW

The one-dimensional case is straight forward, however the dance videos comprise of 29 vectors of angles for each frame and a number of frames ranging between 21 and 133. This scenario requires an adaptation of the similarity measures for a multi-dimensional case. Two approaches were examined:


- Evaluating similarity measures for each body pose: each data point in the 1D time series is now a vector of 29 angles in the vertical direction of the data frame:



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100
Body Part 2	80	82	80	80	78	76	78	79
Body Part 3	176	177	179	179	179	177	179	179
Body Part 4	103	105	110	113	117	120	122	126
Body Part 5	137	141	142	145	147	150	158	162
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 6: Video 1 - body pose - vertical approach

- Evaluating similarity measures for angles of individual body parts: each data point in the 1D time series is now a vector of n angles (corresponding to the number of frames) in the horizontal direction of the data frame:



	P1	P2	P3	P4	P5	P6	P7	P8	
Body Part 1	95	94	99	99	100	101	100	100
Body Part 2	80	82	80	80	78	76	78	79
Body Part 3	176	177	179	179	179	177	179	179
Body Part 4	103	105	110	113	117	120	122	126
Body Part 5	137	141	142	145	147	150	158	162
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Figure 7: Video 1 - body parts - horizontal approach

5.1 Accounting for missing values

It is important to note that angles vectors might include missing values (NaNs). This can be for example due to a body part being hidden behind the dancer's body. Any mathematical operation involving NaNs in Python outputs nan as a result; thus if one entry in a vector is not a number, the computation for that iteration has to be skipped. For example, when evaluating the similarity between \vec{x} and \vec{y} :

i	\vec{x}	\vec{y}
1	91	16
2	88	160
3	179	177
4	112	93
5	143	177
6	108	94
7	136	172
8	94	NaN
9	133	NaN
10	130	NaN
11	96	NaN
12	87	NaN
13	76	92
14	172	179
15	78	92
16	163	175
17	161	175
18	151	171
19	32	165
20	127	113
21	74	25
22	109	84
23	71	18
24	35	168
25	90	89
26	90	92
27	178	109
28	85	93
29	90	93

Here, the 8th, 9th, 10th, 11th, and 12th iterations cannot be evaluated. Consequently, the final sum will include computations from 24 entries instead of 29. Even though this is not an ideal scenario, an approximate similarity measure can still be extracted. However, this is not the case if no entry can be evaluated because missing values are present in either index of the two vectors, for

example:

i	\vec{x}	\vec{y}
1	NaN	16
2	NaN	160
3	NaN	177
4	112	NaN
5	143	NaN
6	108	NaN
7	NaN	172
8	94	NaN
9	133	NaN
10	130	NaN
11	96	NaN
12	87	NaN
13	NaN	92
14	NaN	179
15	78	NaN
16	NaN	NaN
17	NaN	NaN
18	151	NaN
19	32	NaN
20	127	NaN
21	NaN	25
22	109	NaN
23	NaN	18
24	NaN	168
25	NaN	89
26	90	NaN
27	NaN	109
28	85	NaN
29	NaN	93

In this case, the DTW matrix cell is filled with the value corresponding to the same column but previous row.

5.2 Evaluation of similarity measure accuracy

In order to establish whether similarity measures are accurate, it is paramount to compare them with a ground truth. Included in the Master project there was the ground truth for dance videos clustering. Ideally, this list should have clearly stated which videos were most similar to each other. However, a couple of mistakes were found. Additionally, this list had to be filtered from 73 to 52 videos due to confusing labeling or absence of videos in the data set. The final list of videos is as following:

Groups	Videos Labels
1	BS_F_BK, BS_F_LT, BS_F_RT BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S
2	LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_dis, LU_S_dis
3	BJ_FT ,SJ_FT
4	AR, TA, LU_F_big, LU_S_big
5	BJ_RT, SJ_RT, BJ_LT, SJ_LT
6	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
7	TB_F_FB, TB_S, TB_S_FB, TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S, TR_F, TR_S
8	SYN_K
9	BJ_BK, SJ_BK

The similarity measures accuracy in this work were evaluated based on how close they were to this list.

Practically speaking, while videos are being compared against each other, a similarity matrix is filled with the final similarity measures. This similarity matrix is then used to create the clustering with the **AgglomerativeClustering** function from the sklearn package by setting the number of clusters to 9, the affinity as 'precomputed' (since the similarity matrix was created without using any of the predefined function methods), and the linkage as 'average'.

The clustering groups that we obtain for the combinations may be ordered differently from the ground truth, for this reason the following algorithm was developed to make the comparison automated:

- Merge clustering groups labels with list of video names in a data frame
- sort data frame with respect to clustering labels
- create a list with sublists, each sublist is a different clustering group
- do the same for the ground truth (candidate list)
- compare ground truth with candidate, not the other way around
- extract list with highest number of elements from ground truth (reference list)
- check what candidate list the reference list matches the most, i.e. the lists that have the highest match in number of elements
- remove both sublists from lists
- iterate over all of the sublists and count the number of matches, in case of perfect match this will be equivalent to the number of videos

Additionally, the similarity matrix is used to create dendrograms using the **linkage**, **squareform**, and **dendrogram** functions from the `scipy` package by setting the method to 'average'.

5.3 DTW vertical for a multi-dimensional space

One approach to evaluate Euclidean Distance and Cosine Similarity for multiple dimensions is to extract the 29 angles for each frame (hence for each body position) for two videos as inputs. In other words, the 1D time series are now replaced with angle vectors of 29 dimensions each, i.e. one data point in the 1D series is now a vector of 29 angles.

- Each entry of two single angles vectors are extracted and their similarity evaluated through either Euclidean distance or Cosine Similarity
- The similarities of all entries of the two vectors are progressively summed
- The final sum is divided by the number of entries accounted for (normally 29 by less in case of missing values)
- The summation result is used as one data point in the DTW algorithm

5.4 DTW with Euclidean distance vertical for a multi-dimensional space

Adapting the original DTW algorithm to a multi-dimensional scenario involves adding a loop before the DTW matrix computation. In this loop, angles from the vectors corresponding to the same index are subtracted from each other and the result is squared. All results are iteratively summed and the square root of the final result is taken. To account for missing values, the square root is divided by the number of entries that did not contain missing values. Additionally, because different videos have different lengths, the final DTW output is divided by the number of steps of the path:

Listing 3: DTW_ED_for_MD

```

def dtw_ed_MD(s, t):
    # number of columns in each dataframe
    # corresponding to the number of frames for each video
    n, m = len(s.columns), len(t.columns)

    # initialize DIW matrix
    # initialize vector to account for missing values

    # fill in matrix
    for i in range(1, n + 1):
        for j in range(1, m + 1):

            # set count and sum_entries to 0

            # multi-dimensional case
            for k in range(0, len(s[i - 1])):
                # check that operation does not output NA
                if (s[i - 1][k] * t[j - 1][k]) != NA:
                    sum_entries += (s[i - 1][k] - t[j - 1][k])^2
                    count += 1
            if count > 0:
                cost = sqrt(sum_entries)/count

            # fill in matrix
            # in the same way as for 1D case

            # fill in vector of values to account for missing data
            vec_vals[j] = cost + last_min
        else:
            dtw_matrix[i, j] = vec_vals[j]

    # divide by length of DIW path, aka the number of steps
    dtw_final = dtw_matrix[n, m]/n_steps
    return dtw_final

```

The resulting clustering groups and dendrogram are as follows:

Groups	Videos Labels
1	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
2	AR, BJ_BK, BJ_LT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT, SJ_LT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S, TA
3	TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S
4	BJ_RT, SJ_RT
5	BJ_FT
6	TB_F_FB, TB_S_FB
7	TR_F, TR_S
8	SYN_K
9	TB_S

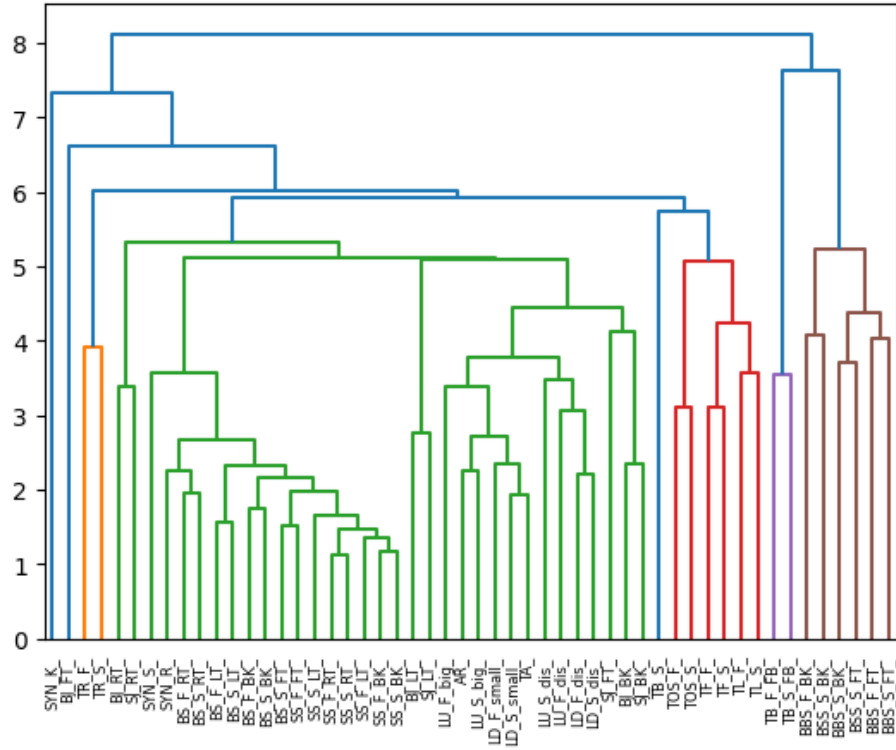


Figure 8: DTW ED dendrogram Vertical

Using this method, the total number of correctly clustered videos is 30.

5.5 Cosine Similarity vertical for a multi-dimensional space

Adapting the original DTW algorithm to a multi-dimensional scenario with Cosine Similarity involves adding a loop before the DTW matrix computation. In this loop, angles from the vectors corresponding to the same index are multiplied. All results and the square of all used angles are iteratively summed. The final results summation is divided by the square root of the multiplication of the sums of the squared angles. This computation already accounts for missing values, hence no further operations are needed. Additionally, because different videos have different lengths, the final DTW output is divided by the number of steps of the path:

Listing 4: DTW_CosSim_for_MD

```

def dtw_cosSim(s, t):
    # number of columns in each dataframe
    # corresponding to the number of frames for each video
    n, m = len(s.columns), len(t.columns)

    # initialize DIW matrix
    # initialize vector to account for missing values

    for i in range(1, n + 1):
        for j in range(1, m + 1):

            # set count, s_squared_sum, t_squared_sum, sum_entries to 0

            # multi-dimensional case
            for k in range(0, len(s[i - 1])):
                if (s[i - 1][k] * t[j - 1][k]) != NA:
                    sum_entries += s[i - 1][k] * t[j - 1][k]
                    s_squared_sum += s[i - 1][k]^2
                    t_squared_sum += t[j - 1][k]^2
                    count += 1

            if count > 0:
                denominator = sqrt(s_squared_sum)*sqrt(t_squared_sum)
                cost = sum_entries/denominator
                cost = 1 - cost

                # fill in matrix
                # in the same way as for 1D case

                # fill in vector of values to account for missing data
                vec_vals[j] = cost + last_min
            else:
                dtw_matrix[i, j] = vec_vals[j]

    # divide by length of DIW path, aka the number of steps
    dtw_final = dtw_matrix[n, m]/n_steps
    return dtw_final

```

The resulting clustering groups and dendrogram are as follows:

Groups	Videos Labels
1	AR, BJ_BK, BJ_LT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT, SJ_LT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S, TA
2	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
3	TB_S, TF_F, TF_S, TL_F, TL_S
4	TB_F_FB, TB_S_FB
5	BJ_FT
6	SYN_K
7	TR_F, TR_S
8	BJ_RT, SJ_RT
9	TOS_F, TOS_S

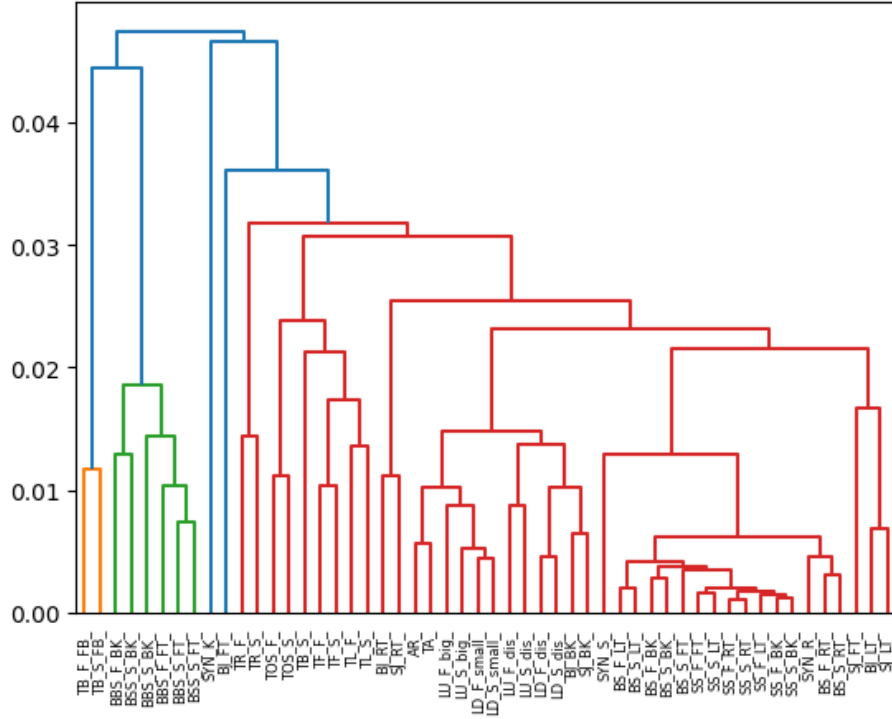


Figure 9: DTW cosSim vertical

Using this method, the total number of correctly clustered videos is 28.
The clustering between DTW Ed and cosSim is very similar: only 2 videos

are clustered differently. However, the clustering with the ground truth is too far off.

5.6 Past and future vectors combinations

One last approach to evaluate dance videos similarities was to combine vectors of angles. Given an position vector that needs to be analysed, the approach consists of combining a set amount of vectors preceding and an equivalent following said vector into a larger vector. The same action is performed for both videos and the entries of the resulting vectors are compared one-to-one.

For example 2 videos are being compared, setting the number of preceding and following frames to 2, it is possible to analyse P4 by combining it P2, P3, P5, and P6:

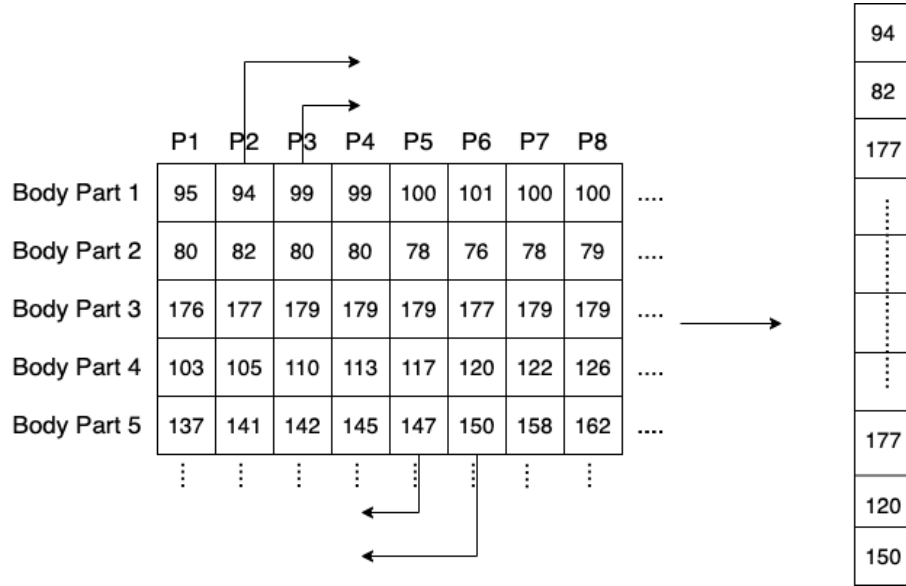


Figure 10: Past and Future vectors

The same process is performed for the second video. These two new vectors are then compared by taking the Euclidean distance of same index entries (i.e. one-to-one comparison of the angles). The resulting clustering groups and dendrograms are as follows:

Groups	Videos Labels
1	AR, BJ_BK, BJ_FT, BJ_LT, BJ_RT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT, SJ_LT, SJ_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S, TA, TF_S, TL_S, TOS_S, TR_S
2	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
3	TF_F, TL_F
4	TOS_F
5	SYN_K
6	TB_F_FB
7	TR_F
8	TB_S
9	TB_S_FB

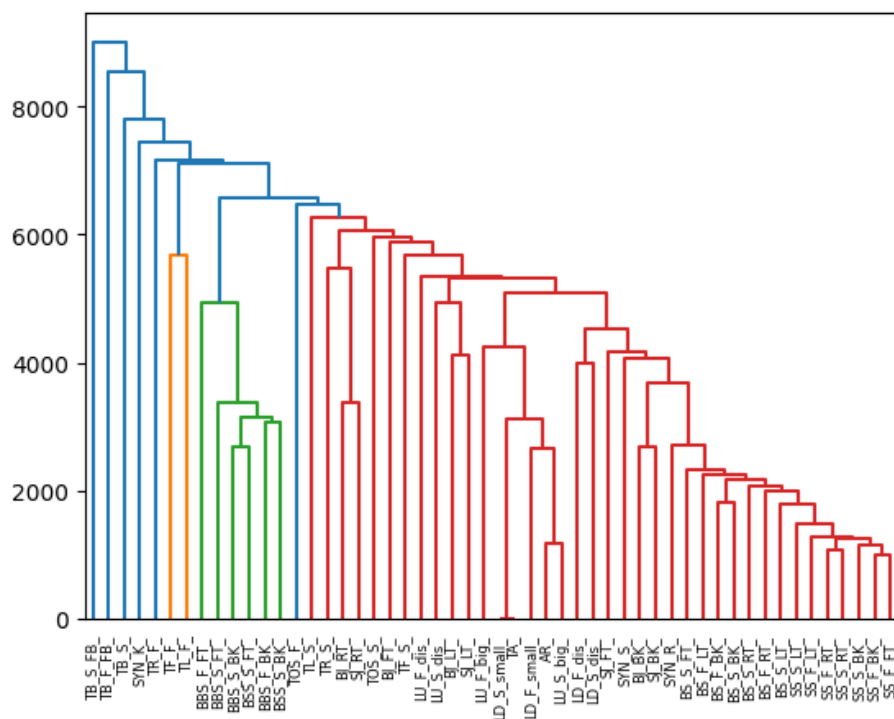


Figure 11: Past and future vectors

Using this method, the total number of correctly clustered videos is 24.

6 Horizontal analysis of angles vectors matrix

Performing similarity measures of DTW with Euclidean distance and Cosine similarity vertically, i.e. taking the vector of body parts for each frame, outputs quite different results compared to the ground truth. A better approach involved performing the similarity measures horizontally, i.e. the vectors of angles do not include the 29 body parts but the angles for each individual body part from start until the end of the video. The DTW matrix now takes as inputs the vectors of angles for the same body parts. The results of each matrix are then divided by the number of steps and summed iteratively.

6.1 DTW horizontal for a multi-dimensional space

The set up to evaluate two videos by angles of vectors of body parts instead of angles of vectors of positions is quite different. In fact, a DTW matrix is created for each body part. Then, individual entries of each vector of angles of two videos are iterated, similarity measures are evaluated with Euclidean Distance and used for the DTW matrix. The result is divided by the number of steps to account for different lengths in videos. All body parts outputs are summed together.

Listing 5: DTW_ED.h_for_MD

```

def dtw_horizontal(s, t):
    # number of rows in each dataframe
    # corresponding to the number of body parts
    n, m = len(s), len(t)

    sum_dtw_values = 0
    for i in range(n):

        # initialize DIW matrix
        # initialize vector to account for missing values
        # both wrt the number of frames for each body part

        for j in range(1, len(s_frames[i]) + 1):
            for k in range(1, len(t_frames[i]) + 1):

                # set count and sum_entries to 0

                # multi-dimensional case
                if (s_frames[i][j - 1] - t_frames[i][k - 1]) != NA:
                    diff_entries = abs(s_frames[i][j - 1] -
                                       t_frames[i][k - 1])

                    count += 1
                if count > 0:
                    cost = diff_entries

                # fill in matrix
                # in the same way as for 1D case

                # fill in vector of values to account for missing data
            else:
                dtw_matrix[j, k] = vec_vals[k]

        # divide by length of DIW path, aka the number of steps
        dtw_final = dtw_matrix[len(s_frames[i]), len(t_frames[i])] / n_steps
        sum_dtw_values += dtw_final
    return sum_dtw_values

```

The resulting clustering groups and dendrogram are as follows:

Groups	Videos Labels
1	BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, SYN_S
2	BJ_BK, BJ_FT, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, SJ_BK, SJ_FT
3	BJ_RT, SJ_RT
4	AR, TA
5	BJ_LT, SJ_LT
6	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
7	TB_S, TB_S_FB, TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S, TR_F, TR_S
8	SYN_K
9	TB_F_FB

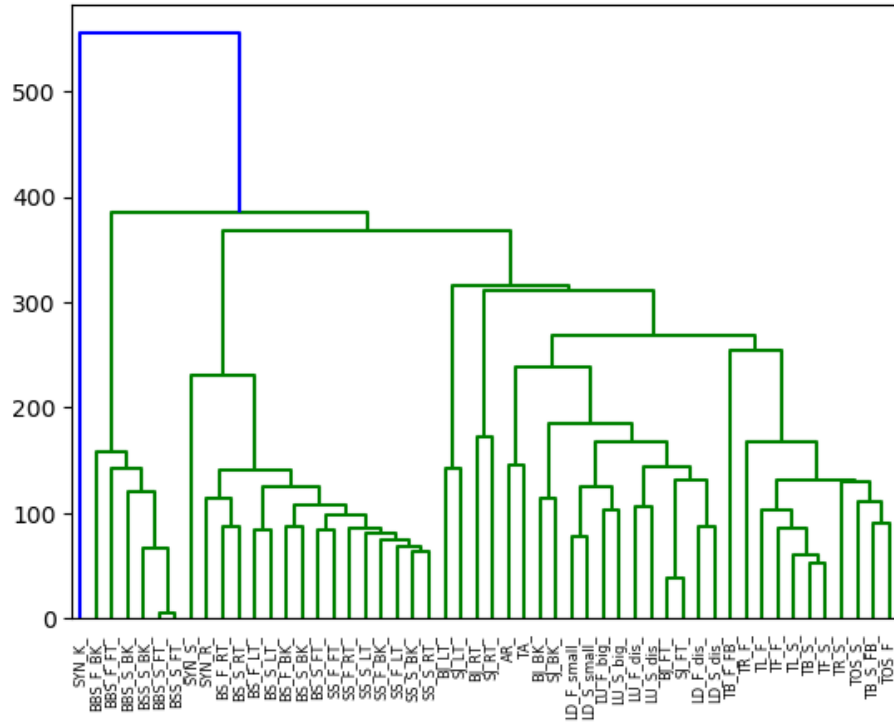


Figure 12: DTW ED horizontal

Using this method, the total number of correctly clustered videos is 42.

6.2 Cosine Similarity horizontal for a multi-dimensional space

The set up for this approach is similar to the one in the previous section. Here, a DTW matrix is also created for each body part. Then, individual entries of each vector of angles of two videos are iterated, similarity measures are evaluated with Cosine Similarity and used for the DTW matrix. However, the result is not divided by the number of steps as that part is already accounted for by the squared sum of the angles for each vector. All body parts outputs are summed together.

Listing 6: DTW_cosSim_h_for_MD

```
def dtw_cosSim_horizontal(s, t):
    n, m = len(s), len(t)
    comp_vals = 0
    for i in range(n):

        # initialize DTW matrix
        # initialize vector to account for missing values
        # both wrt the number of frames for each body part

        for j in range(1, len(s_frames[i]) + 1):
            for k in range(1, len(t_frames[i]) + 1):
                count = 0

                # multi-dimensional case
                if (s_frames[i][j - 1] * t_frames[i][k - 1]) != NA:
                    mult_entries = s_frames[i][j - 1] * t_frames[i][k - 1]
                    count += 1

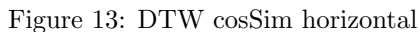
                if count > 0:
                    denominator = sqrt(s_squared_sum) * sqrt(t_squared_sum)
                    cost = mult_entries / denominator
                    cost = 1 - cost

                # fill in matrix
                # in the same way as for 1D case

                # fill in vector of values to account for missing data
            else:
                dtw_matrix[j, k] = vec_vals[k]

        # divide by length of DTW path, aka the number of steps
        dtw_final = dtw_matrix[len(s_frames[i]), len(t_frames[i])] / n_steps
        comp_vals += dtw_final
    return comp_vals
```

Groups	Videos Labels
1	LU_S_dis, SJ_LT, SJ_RT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_K, SJ_FT, SYN_R, TA, TB_F_FB, TB_S, TB_S_FB, TF_F_, TF_S, TL_F, TL_S, TOS_F, TOS_S, SYN_S, SJ_BK, TR_S, LU_S_big, BBS_F_FT, BJ_FT, BJ_LT, BJ_RT, BSS_S_BK, BSS_S_FT, TR_F, BBS_S_FT, BBS_F_BK, LD_F_dis, LD_F_small, LD_S_dis, LD_S_small, LU_F_big, LU_F_dis, BS_S_FT, BBS_S_BK
2	BS_S_LT
3	BS_S_RT
4	BS_F_BK
5	BS_F_RT
6	BS_F_LT
7	BS_S_BK
8	AR
9	BJ_BK



Using this method, the total number of correctly clustered videos is 13.

It was important to analyse these brute force methods to understand which one would bring the highest accuracy level. However they take a long time, thus the lower bound Keough indexing method was implemented to speed the analysis up while maintaining the same level of accuracy.

7 LB_Keough for 1D

The Lower Bound Keough (LB_Keough) performs early data pruning by pre-filtering data for a candidate series (C) that does not fall below a certain set distance. If this "best-so-far" condition is passed, it computes the exact DTW distance between C and the query (Q). The early pruning is done by enveloping Q based on a set Sakoe-Chiba distance (r):

$$U_i = \max(q_{i-r} : q_{i+r}), L_i = \min(q_{i-r} : q_{i+r}) \quad (9)$$

Where U_i represents the upper envelope value for data point i and L_i the corresponding lower.

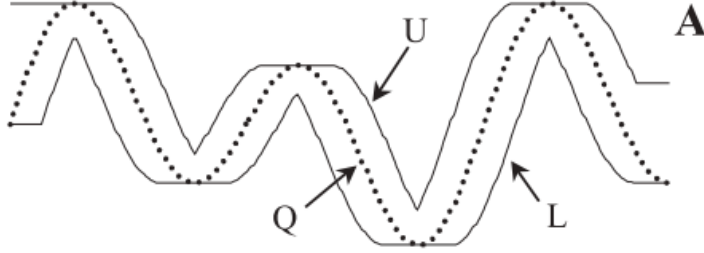


Figure 14: LB_Keough Query envelope

This envelope is then segmented into different Minimum Binding Rectangles (MBRs). MBRs are built based on the maximum value in the upper envelope and the minimum in the lower for a chunk of the data set.

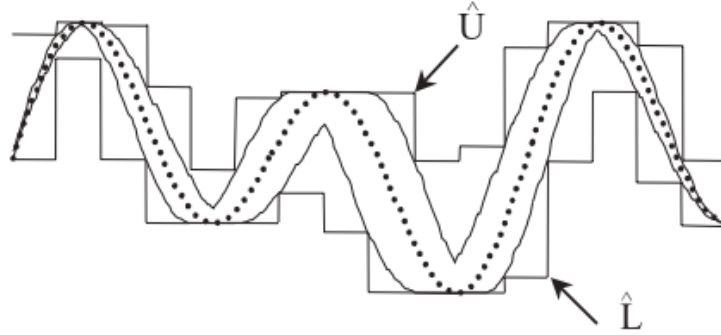


Figure 15: LB_Keough Query MBRs

The candidate is then also divided into MBRs

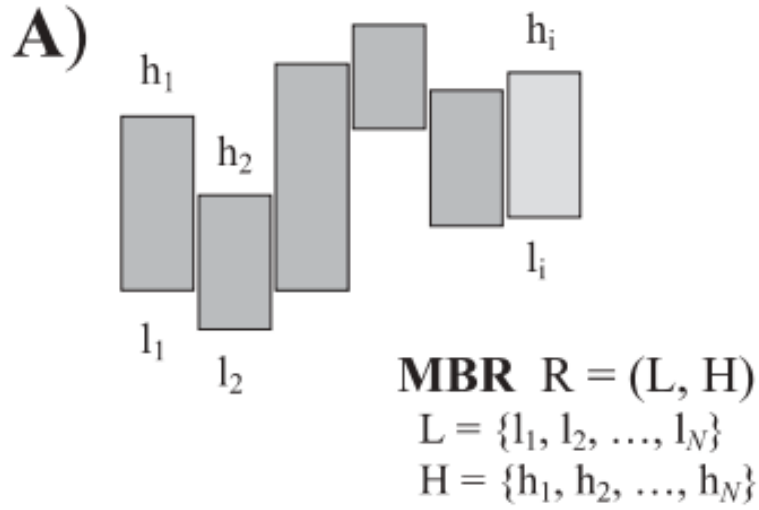


Figure 16: LB_Keough Candidate MBRs

and the distance between the query and the candidate MBRs is calculated.

$$MINDIST(Q, R) = \sqrt{\sum_{i=1}^N \frac{n}{N} \begin{cases} (l_i - U_i)^2 & \text{if } l_i > U_i \\ (h_i - L_i)^2 & \text{if } l_i > L_i \\ 0 & \text{otherwise} \end{cases}} \quad (10)$$

Where n represents the length of the series and N a pre-defined number of MBRs.

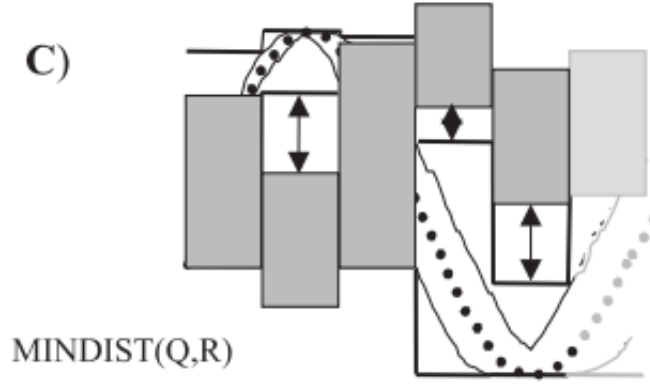


Figure 17: Distance between query and candidate MBRs

In other words, the LB_Keough calculates the minimum distance between two MBRs if the upper value of the candidate falls below the lower of the query or if the lower value of the candidate falls above the upper of the query. The algorithm described in the paper was set to only find the candidate series with the shortest distance from the query. For every video, if this distance is less than the best-so-far condition, the accurate DTW distance is calculated:

Algorithm 1 LB_Keough

```

best_so_far = infinity
for  $i$  in all vectors in single video do
   $LB\_dist = LB\_Keough\_dist(C[i], Q)$ 
  if  $LB\_dist < best\_so\_far$  then
     $true\_dist = DTW(C[i], Q)$ 
    if  $true\_dist < best\_so\_far$  then
       $best\_so\_far = true\_dist$ 
       $index\_of\_best\_match = i$ 
    end if
  end if
end for

```

This best-so-far information was not used in this work and this algorithm was modified to obtain different query results which will be explicated later in the paper. To verify the validity of this method, the LB_Keough distance was used in the same way as the DTW horizontal approach.

8 LB_Keough for a multi-dimensional space

To verify the validity of the LB_Keough method, the same concept as for the horizontal DTW is applied to estimate the ideal combination of values for the Sakoe-Chiba length and the number of rectangles.

The range of values for the MBRs and Sakoe-Chiba lengths was set between 1 and 20 as the smallest video in the dataset comprised of 21 frames.

Listing 7: LB_Keough_for_MD

```

highest_n_matches = 0
sc_N_comb = []
for sakoe_chiba in range(1, 21):
    for N in range(1, 21):
        # start time

        for index_query in range(0, 52):

            # envelope upper and lower query envelope
            u = upper_envelope(DF_query, sakoe_chiba)
            l = lower_envelope(DF_query, sakoe_chiba)

            # construct upper and lower query MBRs
            Q_u_r = construct_upper_MBRs(u, N)
            Q_l_r = construct_lower_MBRs(l, N)

            # loop over all other videos to be compared with query
            for g in range(index_query + 1, 52):

                # construct upper and lower candidate MBRs
                T_u_r = construct_upper_MBRs(newDF, N)
                T_l_r = construct_lower_MBRs(newDF, N)

                # calculate LB Keough distance
                lb_keough = calc_min_dist_MD(T_u_r, T_l_r, Q_u_r, Q_l_r, N)

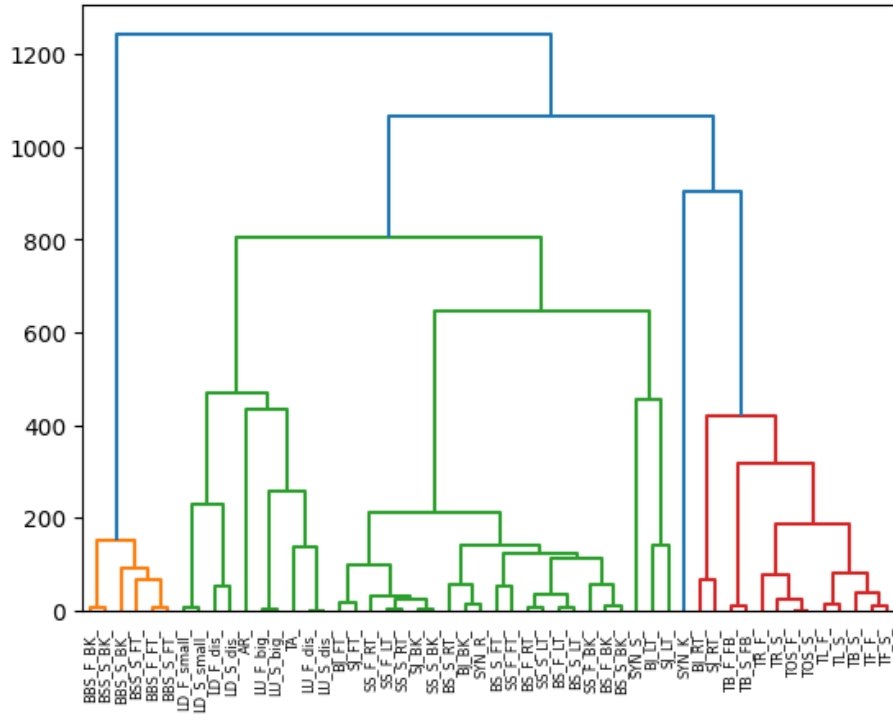
            # end time
            # compare clustering groups with ground truth
            # if the number of matches is greater than highest_n_matches
            # than previously saved value —> save new sc_N_comb and new
            # highest_n_matches
print(sc_N_comb)

```

The highest number of matches with the lowest time was 43 out of 52 videos correctly clustered. For this result, Sakoe-Chiba was set to 19 and the number of rectangles to 15. The run time was 146 seconds.

The resulting clustering groups and dendrogram are as follows:

Groups	Videos Labels
1	AR, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, TA
2	TB_F_FB, TB_S, TB_S_FB, TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S, TR_F, TR_S
3	LD_F_dis, LD_F_small, LD_S_dis, LD_S_small
4	BJ_RT, SJ_RT
5	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
6	BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, SJ_BK, SJ_FT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R, BJ_BK, BJ_FT
7	BJ_LT, SJ_LT
8	SYN_K
9	SYN_S



8.1 Filtering dimensions for LB_Keough

It was noticed that the same clustering groups could be obtained without computing some of the angles.

To keep the same clustering groups order, it is possible to remove the following angles:

- angle from left shoulder to right shoulder
- angle from nose to neck to right eye to nose
- angle from left hip to left upper leg
- angle from left upper leg to left lower leg
- angle from left lower leg left ankle to heel
- angle from torso to right hip
- angle from torso to left hip

To keep the same clustering groups but with a different order, it is possible to additionally remove the following angles:

- angle from right hip to right upper leg
- angle from right lower leg to right ankle to heel
- angle from right foot to right toes
- angle from right foot to right lower leg
- angle from left foot to left lower leg
- angle from left foot to left toes

The resulting clustering groups and dendrogram are as follows:

Groups	Videos Labels
1	AR, LU_F_big, LU_F_dis, LU_S_big, LU_S_dis, TA
2	TB_F_FB, TB_S, TB_S_FB, TF_F, TF_S, TL_F, TL_S, TOS_F, TOS_S, TR_F, TR_S
3	BJ_LT, SJ_LT
4	LD_F_dis, LD_F_small, LD_S_dis, LD_S_small
5	BJ_BK, BJ_FT, BS_F_BK, BS_F_LT, BS_F_RT, BS_S_BK, BS_S_FT, BS_S_LT, BS_S_RT, SJ_BK, SJ_FT, SS_F_BK, SS_F_FT, SS_F_LT, SS_F_RT, SS_S_BK, SS_S_LT, SS_S_RT, SYN_R
6	SYN_K
7	SYN_S
8	BBS_F_BK, BBS_F_FT, BBS_S_BK, BBS_S_FT, BSS_S_BK, BSS_S_FT
9	SJ_RT, BJ_RT

ing the candidate set. The idea is to calculate the similarity between two videos, if this value is below a certain threshold, the video will be appended to a list.

Listing 8: DTW_videos_set

```
for g in range(0, 52):
    if g != index_query:
        actual_dtw = dtw_horizontal(DF, DF_query)
        if actual_dtw <= th:
            actual_dtw_ls.append([g, actual_dtw])
```

The threshold was set to 200 and the query video was the 6th, i.e. BJ_BK. Following are the results of the simulation:

Videos indexes	Similarity measure
LD_F_dis	130.601
LD_S_dis	148.298
LD_S_small	166.775
SJ_BK	113.162

The simulation took approximately 4606 seconds. Using this results, it is possible now to observe the accuracy of the candidate set produced by the LB_Keough and compare run time.

9.2 Candidate set with DTW and LB_Keough

Using LB_Keough before performing DTW calculations helps to reduce the run time. The idea here is to first create a candidate set using the LB_Keough method and then perform final filtering with DTW:

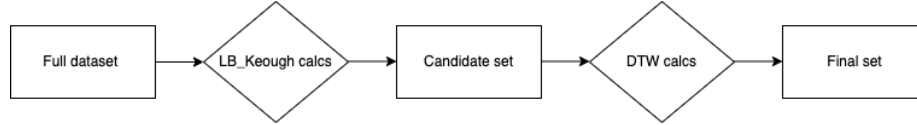


Figure 18: LB_Keough and DTW candidate set

In this scenario, the LB_Keough has to output a candidate set that would include the correct videos and perhaps additional ones to be filtered later on. For this reason, the values of the MBRs and Sakoe-Chiba length need to be optimized such that the combination of LB_Keough and DTW is optimal. In order to find the optimal combination of number of rectangles and Sakoe-Chiba band, all combinations need to be performed:

Listing 9: DTW_LB-query

```

index_query = 5
tot_time_current = np.inf
sc_N_comb = []

for sakoe_chiba in range(1, 21):
    for N in range(1, 21):
        # start time

        # construct query envelope
        # construct query MBRs  $\rightarrow$   $Q_{u-r}$  and  $Q_{l-r}$ 

        # loop over all other videos to be compared with query
        for g in range(0, 52):

            # construct candidate MBRs  $\rightarrow$   $T_{u-r}$  and  $T_{l-r}$ 

            # calculate LB Keough distance
            lb_keough = calc_min_dist_MD( $T_{u-r}$ ,  $T_{l-r}$ ,  $Q_{u-r}$ ,  $Q_{l-r}$ , N)

            # check for threshold condition
            if lb1 <= th:
                ls_lb_files.append([g, DF, lb_keough])

        # calculate accurate DTW
        for i in range(len(ls_lb_files)):
            # calculate DTW horizontal for videos in list
            # check that new distance passes threshold condition
            # append to new list

        # end time
        # if time is less than currently saved tot_time_current,
        # replace sc_N_comb list and tot_time_current

    print(sc_N_comb, tot_time_current)

```

The ideal combination of Sakoe-Chiba length and MBRs is INSERT RESULTS HERE with a total run time of approximately 6.509 seconds. The final set of videos resulted to be as such: INSERT LIST OF VIDEOS HERE

9.3 Candidate set with LB_Keough only

Seeing that the LB_Keough clustering outputted the same number of matches as the DTW horizontal, it is of interest to see the effects of using the LB_Keough not only to create the candidate set, but also to obtain the final set of videos.

The candidate set used INSERT VALUES FROM DTW LB COMBO HERE respectively for the MBRs and the Sakoe-Chiba length, as outputted in the previous section, whereas the final set is retrieved using the ideal values, i.e. 15 for the MBRs and 19 for the Sakoe-Chiba length.

INSERT RESULTS HERE

10 Vitriwr

11 Conclusions and Future Work