




DTW-C++: Fast dynamic time warping and clustering of time series data

Volkan Kumtepli ¹, Rebecca Perriment ¹, and David A. Howey ¹

¹ Department of Engineering Science, University of Oxford, OX1 3PJ, Oxford, UK

DOI: [10.21105/joss.06881](https://doi.org/10.21105/joss.06881)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Tristan Miller](#)  

Reviewers:

- [@ZhenchenHong](#)
- [@i64](#)

Submitted: 29 April 2024

Published: 06 September 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

Time-series data analysis is of interest in a huge number of different applications, from finding patterns of energy consumption to detecting brain activity or discovering stock price trends. Unsupervised learning methods can help analysts unlock patterns in data, and a key example of this is clustering. However, clustering of time series data can be computationally expensive for large datasets. We present an approach for computationally efficient dynamic time warping (DTW) and clustering of time-series data. The method frames the dynamic warping of time series datasets as an optimisation problem solved using dynamic programming, and then clusters time series data by solving a second optimisation problem using integer programming. There is also an option to use k-medoids clustering when a certificate for global optimality is not essential. The increased speed of our approach is due to task-level parallelisation and memory efficiency improvements. The method was tested using the UCR Time Series Archive, and was found to be on average 33% faster than the next fastest option when using the same clustering approach. This increases to 64% faster when considering only larger datasets (with more than 1000 time series). The integer programming clustering is most effective on small numbers of longer time series, because the DTW computation is faster than other approaches, but the clustering problem becomes increasingly computationally expensive as the number of time series increases.

Statement of need

The target audience for this software is very broad, since clustering of time series data is relevant in many applications from energy to finance and medicine. However, as data availability increases, so does the complexity of the clustering problem. Most time series clustering algorithms depend on dimension reduction or feature extraction techniques to enable scaling to large datasets, but this can induce bias in the clustering ([Aghabozorgi et al., 2015](#)). Dynamic time warping ([Sakoe & Chiba, 1978](#)) is a well-known technique for manipulating time series to enable comparisons between datasets, using local warping (stretching or compressing along the time axis) of the elements within each time series to find an optimal alignment between series. This emphasises the similarity of the shapes of the respective time series rather than the exact alignment of specific features. Unfortunately, DTW does not scale well in computational speed as the length and number of time series to be compared increases—the computational complexity grows quadratically with the total number of data points. This is a barrier to DTW being widely implemented in large-scale time series clustering ([Rajabi et al., 2020](#)). In response, DTW-C++ was written to handle large time series efficiently, directly processing the raw data rather than first extracting features.

In contrast to existing tools available for time series clustering using DTW, such as DTAIDistance ([Meert et al., 2022](#)) and TSlearn ([Tavenard et al., 2020](#)), DTW-C++ offers significant improvements in speed and memory use, enabling larger datasets to be clustered.

This is achieved by

1. task-level parallelisation, where multiple pairwise comparisons between time series can be evaluated simultaneously, and,
2. improved memory management—since the clustering algorithm only needs the final distance computed between pairwise time series, the DTW distance computation stores only the most recent previous vector, rather than the entire warping matrix.

In addition, DTW-C++ offers the option of clustering using a new algorithm (described below) based on integer programming. The advantage of this over k-based methods is that it guarantees finding a global optimal solution in most cases, and in the rare event that the global optimum cannot be found, the gap between the best solution and the global optimum is given.

Current DTW-C++ functionality

The current functionality of the software is:

- Calculate DTW pairwise distances between all pairs of time series in a set, using a vector based approach to reduce memory use. There is also the option to use a Sakoe-Chiba band to restrict warping in the DTW distance calculation (Sakoe & Chiba, 1978). This speeds up the computation time, as well as being a useful constraint for some clustering scenarios (e.g., if an event must occur within a certain time window to be considered similar).
- Produce a distance matrix containing all pairwise comparisons between each time series in the dataset.
- Split all time series into a predefined number of clusters, with a representative centroid time series for each cluster. This can be done using integer programming or k-medoids clustering, depending on user choice.
- Output the clustering cost, which is the sum of distances between every time series within each cluster and its cluster centroid.
- Find the silhouette score and elbow score for the clusters to aid the user decision on how many clusters, k , to include. The silhouette score is defined by the difference between the mean intra-cluster distance and the mean nearest-cluster distance, divided by the maximum of these two distances (Rousseeuw, 1987). This considers both the similarity of a time series to its own cluster as well as its dissimilarity from other clusters. The elbow score is based on the cost of the clustering exercise, which sums together the distance between each time series and its centroid. Therefore the similarity of a time series to its own cluster is considered, but not its dissimilarity from other clusters.

Mathematical background

Dynamic time warping

Consider a time series to be a vector of arbitrary length. Consider that we have p such vectors in total, each possibly differing in length. To find a subset of k clusters within the set of p vectors, we must first make $\frac{1}{2}\binom{p}{2}$ pairwise comparisons between all vectors within the total set and find the 'similarity' between each pair. In this case, the similarity is defined as the DTW distance. Consider two time series x and y of differing lengths n and m respectively,

$$x = (x_1, x_2, \dots, x_n)$$
$$y = (y_1, y_2, \dots, y_m).$$

The DTW distance is the sum of the Euclidean distance between each point and its matched point(s) in the other vector, as shown in Figure 1. To find the DTW distance, the following constraints must be met:

1. The first and last elements of each series must be matched.
2. Only unidirectional forward movement through relative time is allowed, i.e., if x_1 is mapped to y_2 then x_2 may not be mapped to y_1 (monotonicity).
3. Each point is mapped to at least one other point, i.e., there are no jumps in time (continuity).

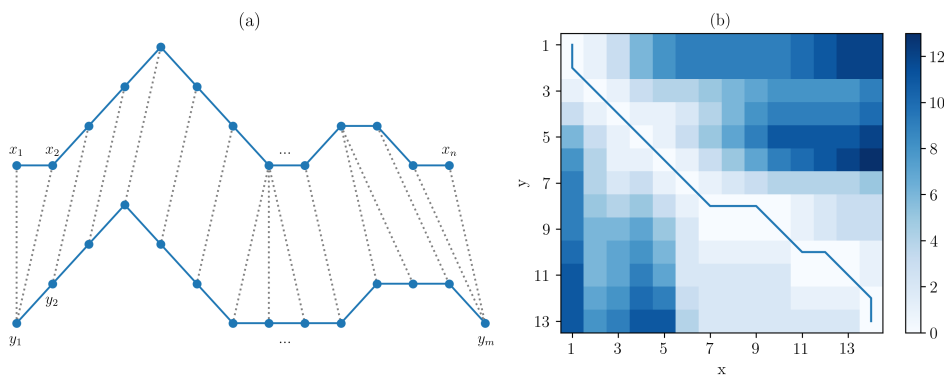


Figure 1: (a) Two time series with DTW pairwise alignment between each point, showing the one-to-many mapping properties of DTW. (b) Cost matrix C for the two time series, showing the warping path and final DTW cost at $c_{14,13}$.

Finding the optimal warping arrangement is an optimisation problem that can be solved using dynamic programming, which splits the problem into easier sub-problems and solves them recursively, storing intermediate solutions until the final solution is reached. To understand the memory-efficient method used in DTW-C++, it is useful to first examine the full cost matrix solution, as follows. For each pairwise comparison, an n by m matrix $C^{n \times m}$ is calculated, where each element represents the cumulative cost between series up to the points x_i and y_j :

$$c_{i,j} = (x_i - y_j)^2 + \min \begin{cases} c_{i-1,j-1} \\ c_{i-1,j} \\ c_{i,j-1} \end{cases} \quad (1)$$

The final element in the matrix $c_{n,m}$ is then the total cost, and this provides the metric for comparing the two series x and y . Figure 1 shows an example of this cost matrix C and the warping path through it.

Clustering

For the clustering algorithm, only the final cost for each pairwise comparison is required; the actual warping path (i.e., mapping between time series) is superfluous. The memory complexity of the cost matrix C is $\mathcal{O}(nm)$, so as the length of the time series grows, the memory required greatly increases. Therefore, significant reductions in memory use can be achieved by not storing the entire cost matrix. Since the warping path is not required, we only need to store a vector containing the previous row relating to the current step of the dynamic programming sub-problem (i.e., the previous three values $c_{i-1,j-1}$, $c_{i-1,j}$, $c_{i,j-1}$), as indicated in Equation 1.

We now introduce the notation $d_{x,y} = c_{n,m}$ to denote the final (scalar) cost relating to the pairwise comparison between time series x and y , given by the final element in the cost matrix relating to the x and y time series. To cluster several time series, this cost is first computed for every pairwise comparison between every time series. As shown in Figure 2, all of the pairwise distances are then stored in a separate symmetric matrix, $D^{p \times p}$, where p is the total number

of time series in the clustering exercise. In other words, the element $d_{i,j}$ in this matrix gives the cost between time series i and j .

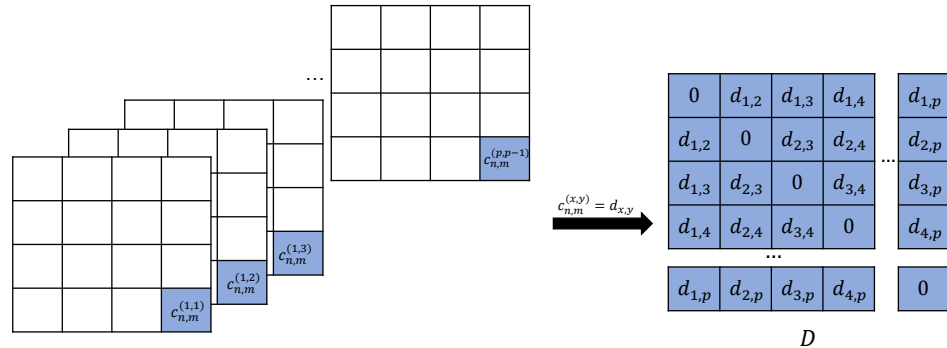


Figure 2: The individual DTW costs from each pairwise comparison between time series in the dataset are all combined to form a distance matrix D .

Using this distance matrix, D , the full set of time series can be split into k separate clusters with integer programming. The problem formulation begins by considering a binary square matrix $A^{p \times p}$, where $A_{ij} = 1$ if time series j is a member of the i th cluster centroid, and 0 otherwise, as shown in Figure 3.

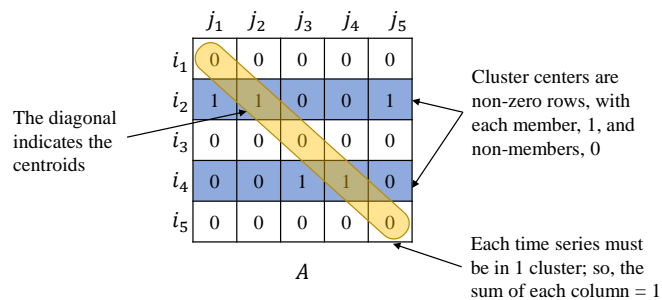


Figure 3: Example clustering matrix, where an entry of 1 indicates that time series j belongs to the cluster with centroid i .

As each centroid has to be in its own cluster, non-zero diagonal entries in A represent centroids. Our objective is to find A , and this may be formulated as an optimisation problem

$$A^* = \operatorname{argmin}_A \sum_i \sum_j D_{ij} \times A_{ij}, \quad (2)$$

subject to the following constraints:

1. Only k series can be centroids,

$$\sum_{i=1}^p A_{ii} = k.$$

2. Each time series must be a member of one and only one cluster,

$$\sum_{i=1}^p A_{ij} = 1 \quad \forall j \in [1, p].$$

3. In any row, there can only be non-zero entries if the corresponding diagonal entry is non-zero, so a time series can only be in a cluster where the row corresponds to a centroid time series,

$$A_{ij} \leq A_{ii} \quad \forall i, j \in [1, p].$$

This integer program is solved in DTW-C++ using Gurobi (Gurobi Optimization, LLC, 2024) or HiGHS (Huangfu & Hall, 2018). After solution, the non-zero diagonal entries of A represent the centroids, and the non-zero elements in the corresponding columns in A represent the members of that cluster. In the example in Figure 3, the clusters are time series 1, 2, 5 and 3, 4 with the bold type face entries indicating the centroids.

Finding a globally optimal solution with this method can result in increased computation times depending on the number of time series within the dataset and the DTW distances. Therefore, there is also a built-in option to cluster using k-medoids, as used in other packages such as DTAIDistance (Meert et al., 2022). The k-medoids method is often quicker as it is an iterative approach, however it is subject to getting stuck in local optima. The results in the next section show the timing and memory performance of both integer programming clustering and k-medoids clustering using DTW-C++ compared to other packages.

Comparison

We compared our approach with two other DTW clustering packages, DTAIDistance (Meert et al., 2022) and TSlearn (Tavenard et al., 2020) using data from the UCR Time Series Classification Archive Dau et al. (2019), which consists of 128 time series datasets with up to 16,800 data series of lengths up to 2,844. Benchmarking against TSlearn was stopped after the first 22 datasets because the results were consistently over 20 times slower than DTW-C++. Table 1 shows the results for datasets downselected to have the number of time series, N , greater than 100, and the length of each time series greater than 500 points. This is because DTW-C++ is aimed at larger datasets where the speed improvements are more relevant.

DTW-C++ is the fastest package for 90% of the datasets, and all 13 datasets where DTAIDistance was faster were cases where the entire clustering process was completed in 1.06 seconds or less. Across the whole collection of datasets, DTW-C++ was on average 32% faster. When looking at larger datasets, with $N > 1000$, DTW-C++ is on average 65% faster. In all, apart from 2 of the 115 cases where DTW-C++ is the fastest, we used the k-medoids algorithm for clustering. Figure 4 shows the increasing performance of DTW-C++ as the number of time series increases. In this comparison, both algorithms used k-medoids, so the speed improvement is due to faster dynamic time warping method in DTW-C++.

With respect to clustering, DTW-C++ with integer programming was on average 16 times slower than DTAIDistance over all samples, and as the number of time series increases, integer programming clustering becomes increasingly slower (Figure 5). This is to be expected because the computational complexity of the integer programming optimisation increases significantly as the number of time series in the clustering problem increases. However, as the lengths of each time series increase, the performance of integer programming converges to the speed of DTAIDistance, and the former finds globally optimal results. Therefore, the integer programming approach is recommended for occasions when the individual time series to be clustered are very long, but the number of individual time series is small (e.g., fewer than 1000).

The performance comparison on all datasets in the UCR Time Series Classification Archive and any updated benchmarking tests can be found in the repository.

	Number of time series	Length of time series	DTW-C++ IP (s)	DTW-C++ k-Medoids (s)	DTAI Distance* (s)	Dis- tance* (s)	Time de- crease (%)
CinCECGTorso	1380	1639	3008.4	1104.2	1955.9		44
Computers	250	720	16.1	10.5	12.8		18
Earthquakes	139	512	3.2	2.4	2.5		3
EOGHorizontalSignal	362	1250	81.8	27.6	82.9		67
EOGVerticalSignal	362	1250	85.9	30.2	85.2		65
EthanolLevel	500	1751	325.7	198.9	302.3		34
HandOutlines	370	2709	383.7	280.9	415.9		32
Haptics	308	1092	65.5	24.0	45.5		47
HouseTwenty	119	2000	23.8	19.1	22.0		13
InlineSkate	550	1882	412.4	198.9	423.4		53
InsectEPGRegularTrain	249	601	12.3	5.6	8.9		37
InsectEPGSmallTrain	249	601	11.6	5.3	8.9		41
LargeKitchenAppliances	375	720	44.6	25.6	31.8		20
Mallat	2345	1024	2948.7	517.0	2251.3		77
MixedShapesRegularTrain	2425	1024	2811.8	1221.9	2367.1		48
MixedShapesSmallTrain	2425	1024	2793.7	934.0	2369.3		61
NonInvasiveFetalECGThorax1	1965	750	52599.0	128.7	941.9		86
NonInvasiveFetalECGThorax2	1965	750	4905.4	115.6	951.0		88
Phoneme	1896	1024	46549.0	198.4	1560.6		87
PigAirwayPressure	208	2000	84.6	56.7	73.2		23
PigArtPressure	208	2000	78.9	41.8	71.1		41
PigCVP	208	2000	73.5	51.7	69.5		26
RefrigerationDevices	375	720	36.8	20.3	28.4		28
ScreenType	375	720	38.6	16.1	28.5		43
SemgHandGenderCh2	600	1500	335.9	315.2	325.4		3
SemgHandMovementCh2	450	1500	177.7	107.2	181.1		41
SemgHandSubjectCh2	450	1500	186.4	96.7	177.6		46
ShapesAll	600	512	67.5	15.1	44.4		66
SmallKitchenAppliances	375	720	41.7	23.8	30.1		21
StarLightCurves	8236	1024	N/A	18551.7	27558.1		33
UWaveGestureLibraryAll	3582	945	N/A	1194.6	4436.9		73

*Benchmark results for Python libraries *may* include an overhead of 10% due to the usage of the *tracemalloc* library.

Table 1: Computational time comparison between DTW-C++ using integer programming and k-medoids, vs. DTAIDistance, and TSlearn, on datasets in the UCR Time Series Classification Archive where $N > 100$ and $L > 500$. The fastest result for each dataset is in bold type.

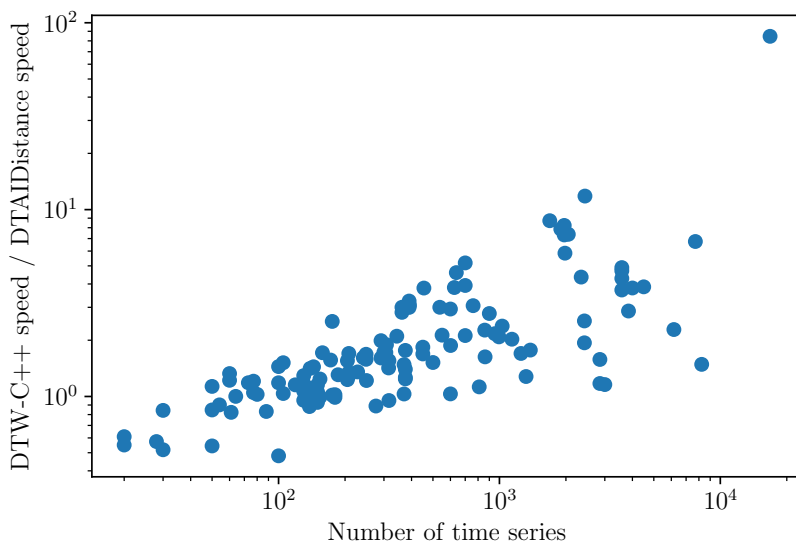


Figure 4: DTW-C++ with k-medoids clustering becomes increasingly faster compared to DTAIDistance as the number of time series increases.

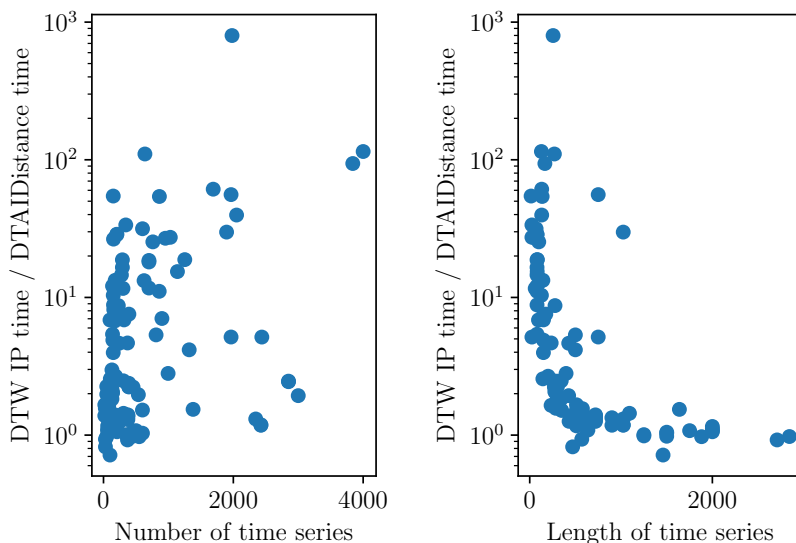


Figure 5: Change in computational time of DTW-C++ using integer programming clustering compared with DTAIDistance as the number of time series in the datasets to be clustered increases and the length of time series in the datasets increases.

Acknowledgements

We are grateful for discussions of this topic with [Battery Intelligence Lab](#) members, and thank BBOXX for project funding and access to data. This work was also funded by the UKRI PFER Energy Superhub Oxford demonstrator and the “Data-driven exploration of the carbon emissions impact of grid energy storage deployment and dispatch” project (EP/W027321/1). The authors would like to particularly thank Dau, Keogh, et al. for their extensive efforts in compiling a diverse range of datasets for the [UCR Time Series Classification Archive](#).

References

- Aghabozorgi, S., Shirkhorshidi, A. S., & Wah, T. Y. (2015). Time-series clustering - a decade review. *Information Systems*, 53, 16–38. <https://doi.org/10.1016/j.is.2015.04.007>
- Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., & Keogh, E. (2019). The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6), 1293–1305. <https://doi.org/10.1109/JAS.2019.1911747>
- Dau, H. A., Keogh, E., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, Gustavo, & Hexagon-ML. (2018). *The UCR time series classification archive*. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
- Gurobi Optimization, LLC. (2024). *Gurobi Optimizer Reference Manual*. <https://www.gurobi.com/documentation/11.0/refman/>
- Huangfu, Q., & Hall, J. A. J. (2018). Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1), 119–142. <https://doi.org/10.1007/s12532-017-0130-5>
- Meert, W., Hendrickx, K., Van Craenendonck, T., Robberechts, P., Blockeel, H., & Davis, J. (2022). Wannesm/dtaidistance v2.3.10. *Zenodo*. <https://doi.org/10.5281/zenodo.1202378>
- Rajabi, A., Eskandari, M., Ghadi, M. J., Li, L., Zhang, J., & Siano, P. (2020). A comparative study of clustering techniques for electrical load pattern segmentation. *Renewable and Sustainable Energy Reviews*, 120. <https://doi.org/10.1016/j.rser.2019.109628>
- Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53–65. [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7)
- Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1), 43–49. <https://doi.org/10.1109/TASSP.1978.1163055>
- Tavenard, R., Fauzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., Payne, M., Yurchak, R., Rußwurm, M., Kolar, K., & Woods, E. (2020). Tslern, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118), 1–6. <http://jmlr.org/papers/v21/20-091.html>