Journal of Statistical Software

September 2021, Volume 99, Issue 9. *doi: 10.18637/jss.v099.i09*

IncDTW: An R Package for Incremental Calculation of Dynamic Time Warping

Maximilian Leodolter
Austrian Institute
of Technology

Claudia Plant
University of Vienna

Norbert Brändle
Austrian Institute
of Technology

Abstract

Dynamic time warping (DTW) is a popular distance measure for time series analysis and has been applied in many research domains. This paper proposes the R package **IncDTW** for the incremental calculation of DTW, and based on this principle **IncDTW** also helps to classify or cluster time series, or perform subsequence matching and k-nearest neighbor search. DTW can measure dissimilarity between two temporal sequences which may vary in speed, with a major downside of high computational costs. Especially for analyzing live data streams, subsequence matching or calculating pairwise distance matrices, runtime intensive computations are unfavorable or can even make the analysis intractable. **IncDTW** tackles this problem by a vector-based implementation of the DTW algorithm to reduce the space complexity from a quadratic to a linear level in number of observations, and an incremental calculation of DTW for updating interim results to reduce the runtime complexity for online applications.

We discuss the fundamental functionalities of **IncDTW** and apply the package to classify multivariate live stream accelerometer time series for activity recognition. Finally, comparative runtime experiments with various R and Python packages for various data analysis tasks emphasize the broad applicability of **IncDTW**.

Keywords: dynamic time warping, time series, k-NN, subsequence matching, distance measure, clustering, classification.

---

1. Introduction

Time series are sets of observations that follow a consecutive temporal relation. Many time series data analysis tasks such as clustering, classification, outlier detection or pattern matching require the definition of a distance measure. Many distance measures such as the Euclidean distance are rather ill-suited whenever two time series are shifted in time, locally recorded with different sampling rates, warped, or have different lengths. Dynamic time warping (DTW) was originally proposed by Sakoe and Chiba (1978), and has since been the distance measure of choice in many works for time series analysis (Berndt and Clifford 1994; Keogh 2002; Ding, Trajcevski, Scheuermann, Wang, and Keogh 2008; Kwankhoom and Muneesawang 2017; Oregi, Pérez, Del Ser, and Lozano 2017; Giorgino *et al.* 2009). DTW is capable of dealing with deformed time series by identifying the best alignment of two time series in a dynamic way.

The major downside of DTW are its expensive computational costs, which are particularly unfavorable for online algorithms processing continuous data streams, where time series analysis must be faster than the elapsed time between consecutive observations. One solution to reduce the runtime for online processing is to incrementally calculate DTW by recycling interim results of previous calculations for every new observation. Without any loss of accuracy, such an incremental processing allows reducing computation time complexity towards linear level in number of observations. Section 2.1 and 3.2 give a detailed discussion about the runtime and space complexity of the DTW algorithm.

The groundwork for the incremental calculation of DTW was done by Rabiner, Rosenberg, and Levinson (1978), who proposed adjustments to the DTW algorithm - open alignments. Since then the principle of the incremental DTW computation has been applied in multiple works, as e.g.: Dixon (2005) applied it for an online algorithm to track musical performances, Mori, Uchida, Kurazume, Taniguchi, Hasegawa, and Sakoe (2006) for an algorithm to early recognize gestures, Tormene, Giorgino, Quaglini, and Stefanelli (2008) to analyze multivariate sensor readings to support neurological patients with real-time information while undergoing motor rehabilitation, Kwankhoom and Muneesawang (2017) for online algorithms which re-identify movement trajectories of persons captured with a 3D depth sensing camera, where time series matching is updated as soon as new video frames are recorded, and Oregi *et al.* (2017) for proposing the Online-DTW (ODTW) algorithm.

Dynamic time warping has already been applied in many research domains and also published in different software packages and programming languages. Table 1 gives an overview of R (R Core Team 2021) packages for DTW computation available at the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`, and Python (Van Rossum *et al.* 2011) packages available at the Python Package Index (PyPI) at `https://pypi.org/`. The package **dtw** (Giorgino *et al.* 2009) offers functions for DTW calculation with different step patterns (see (2) and (3)), warping path restrictions and plotting functions, also for a profound visual analysis of warping alignments of two time series. **dtwclust** (Sarda-Espinosa 2019) puts emphasis on clustering time series based on DTW distances. The functions for DTW calculations are wrappers for those of the **dtw** package. The package **dtwSat** (Maus *et al.* 2019) provides with the time-weighted dynamic time warping a distance method customized to analyzing satellite image time series. **rucrdtw** (Boersch-Supan 2016) is the R version of UCR Suite (Rakthanmanon, Campana, Mueen, Batista, Westover, Zhu, Zakaria, and Keogh 2012) which is a nearest neighbor search algorithm accelerated by lower bounding and pruning methods. It detects the closest fit to a query time series in either one long time series or many of the same length. To the best of our knowledge **rucrdtw** is – besides **IncDTW** – the only R package with a vector-based implementation of the DTW algorithm, thus avoiding memory allocation of matrices. However, the package does neither support multivariate time series nor full alignments for time series of different lengths (i.e., from begin to end for both time series). The package **parallelDist** (Eckert 2018) is the parallel implementation of the function `dist()` – of the package **stats** (R Core Team 2021) – by incorporating

Package & Repo.	First release	Incremental	Vector-based	Diff. lengths	Multivariate	k-NN	Description / Focus
IncDTW CRAN	2017	Yes	Yes	Yes	Yes	Yes	Incremental and fast vector-based DTW calculations, described in this paper.
dtw CRAN	2007	No	No	Yes	Yes	No	Highly functional implementation of DTW (Giorgino *et al.* 2009).
dtwclust CRAN	2015	No	No	Yes	Yes	No	Time series clustering with DTW (Sarda-Espinosa 2019).
dtwSat CRAN	2015	No	No	Yes	Yes	No	Time-Weighted DTW for satellite images (Maus, Câmara, Appel, and Pebesma 2019).
rucrdtw CRAN	2016	No	Yes	No	No	No	1NN-search via DTW (Boersch-Supan 2016).
parallelDist CRAN	2017	No	No	Yes	Yes	No	Parallel distance calculation (Eckert 2018).
dtw PyPI	2014	No	No	Yes	Yes	No	Highly forked and starred (Rouanet 2014).
dtaidistance PyPI	2017	No	Yes	Yes	No	No	Functional and fast (Meert 2017).
cydtw PyPI	2017	No	No	Yes	Yes	No	Simple and fast (Tavenard 2017).

Table 1: Overview of various R and Python packages with different emphasis on calculating and applying the DTW distance.

RcppParallel (Allaire, François, Ushey, Vandenbrouck, Geelnard, and Intel 2021) to speed up computations. Apart from R packages for DTW computation, in the following we discuss Python software, since, – similar to R – Python is probably one of the most taught and applied programming languages for time series analysis, and data mining tasks as clustering, classification and pattern recognition applied on time series data. The **dtw** package (Rouanet 2014) is one of the highest forked and starred packages for DTW computation on the Python Package Index (`https://pypi.org/`). To compute the DTW distance for multivariate time series the package **cydtw** (Tavenard 2017) offers a solution, and the main computation part of the algorithm is implemented in C via **Cython** (Behnel, Bradshaw, Citro, Dalcin, Seljebotn, and Smith 2011). The package **dtaidistance** (Meert 2017) offers a more comprehensive set of functions and also a vector based implementation in C via **Cython**, but does not support multivariate time series. Next to the R packages, Table 1 also lists the Python packages and Section 4.2 details a runtime experiment which compares the Python packages with **IncDTW**. The main contributions of this paper and the R package **IncDTW** are:

- the principle of the incremental DTW calculation ready to use in R functions,
- vector-based implementation of the DTW algorithm – also for multivariate time series – to decrease the computation time,
- the demonstration of applying **IncDTW** for an online time series classification task,
- and comparative runtime experiments with other R and Python packages.

This paper is organized as follows: Section 2 gives an introduction to DTW in general and explains the incremental calculation. Section 3 describes the R package **IncDTW**, discusses the vector-based functions and how to apply the incremental calculation. Section 4 discusses the advantages of **IncDTW** by hand of a typical time series classification experiment, and shows runtime comparisons. Section 5 concludes this paper and gives an outlook of future developments.

2. Dynamic time warping

In the following we recapitulate the classic dynamic time warping algorithm from Sakoe and Chiba (1978) which calculates the distance measure between a query time series $\mathbf{q}$ and a candidate time series $\mathbf{c}$, and their alignment – the so-called warping path – providing information which observations of $\mathbf{q}$ are best matched to the respective observations of $\mathbf{c}$.

The distance measure DTW is defined as the minimal cumulative costs of the shortest non-linear alignment of two time series $\mathbf{q}$ and $\mathbf{c}$. This alignment has the following properties:

1. Boundary conditions: The first element of $\mathbf{q}$ is aligned to the first element of $\mathbf{c}$, and the last element of $\mathbf{q}$ is aligned to the last element of $\mathbf{c}$. Relaxing these conditions allows to find an open alignment, i.e., a partial alignment of two time series with lowest DTW distance (normalized for the lengths). For a more detailed discussion on open alignments (open-end, open-begin and open increment) we refer to the package vignette of **IncDTW** (Leodolter 2021).
2. Monotonicity: Consecutive elements of $\mathbf{q}$ and $\mathbf{c}$ must not be aligned out of time order. The DTW algorithm also returns vectors of indices of $\mathbf{q}$ and $\mathbf{c}$ defining the ordering of the best aligned observations. These vectors must be monotonically increasing, such that $i_k \leq i_{k+1}$, where $1 \leq i_k \leq n = |\mathbf{q}|$, and i_k defines which elements of $\mathbf{q}$ are aligned to $\mathbf{c}$ at the k-th point of time. The same applies to the indices $j_k \leq j_{k+1}$ defining which elements of $\mathbf{c}$ are aligned to $\mathbf{q}$ at the k-th point of time.
3. Non-linear alignment: In contrast to the Euclidean distance, one observation of $\mathbf{q}$ can be aligned to more than one observation of $\mathbf{c}$, and vice versa. Hence it is possible that $i_k = i_{k+1}$ or $j_k = j_{k+1}$.
4. Restrictions: Global or local warping path restrictions can be applied to reduce the space of possible alignments. The most known is the Sakoe Chiba warping window (Sakoe and Chiba 1978), where the time difference of two aligned observations must not exceed the window size parameter, ω: $|i_k - j_k| \leq \omega \ \forall k$.

5. Local distance measure: The distance of two (possibly multivariate) observations of the time series $\mathbf{q}$ and $\mathbf{c}$ can be defined by any distance metric. The standard metrics are the 1-norm and 2-norm. The package vignette of **IncDTW** elaborates how to customize the local distance functions.

6. Step Pattern: The step pattern defines how the local distances are accumulated to calculate the global cost matrix and the walking path. The two popular step patterns (2) and (3) are implemented in **IncDTW**. Sakoe and Chiba (1978), Rabiner and Juang (1993) or Giorgino *et al.* (2009) give a more detailed discussion on step patterns.

It is worth noting that the DTW distance measure is not a metric, since it does not fulfill the triangle inequality. Consequently, lower bounding with the help of the reverse triangle inequality is not possible, which is a method applied for fast nearest neighbor search (Wang 2011).

For the two time series $\mathbf{q}$ of length n and $\mathbf{c}$ of length m, we define $\mathbf{C} \in \mathbf{R}^{n \times m}$ as the local cost matrix, where

$$\mathbf{C}_{i,j} := d(\mathbf{q}_i, \mathbf{c}_j), \tag{1}$$

with $d(\cdot, \cdot)$ as a local distance function for univariate or multivariate time series as described above. The global cost matrix $\mathbf{G} \in \mathbf{R}^{n \times m}$ is determined in an iterative fashion, where each element depends on its predecessors. The step pattern defines these dependencies by weighting and selecting the predecessors. Giorgino *et al.* (2009) and Rabiner and Juang (1993) present a more detailed discussion on step patterns, here we concentrate on two of the most popular and start with the naive step pattern that regards the direct neighboring elements in $\mathbf{G}$ equally weighted:

$$\mathbf{G}_{i,j} = \begin{cases} \sum_{k \le i} \mathbf{C}_{k,1} & j = 1 \\ \sum_{l \le j} \mathbf{C}_{1,l} & i = 1 \\ \mathbf{C}_{i,j} + \min(\mathbf{G}_{i-1,j}, \;\; \mathbf{G}_{i,j-1}, \;\; \mathbf{G}_{i-1,j-1}) & i, j > 1. \end{cases} \tag{2}$$

The step pattern described by Algorithm 2 – "symmetric1" – was not part of the original work about DTW of Sakoe and Chiba (1978) since it does not admit a normalization factor. Nevertheless, it has been applied in several works (Fu 2011; Berndt and Clifford 1994; Sakurai, Faloutsos, and Yamamuro 2007; Keogh 2002; Rath and Manmatha 2003b; Keogh and Pazzani 2000; Rakthanmanon *et al.* 2012) about time series clustering, classification, indexing and pattern mining, and so gained popularity, possibly due to its simplicity to understand and implement.

Another typical step pattern, that is also the default step pattern in the R package **dtw**, is called "symmetric2". Here the diagonal step is weighted with a weight of 2:

$$\mathbf{G}_{i,j} = \begin{cases} \sum_{k \le i} \mathbf{C}_{k,1} & j = 1 \\ \sum_{l \le j} \mathbf{C}_{1,l} & i = 1 \\ \min(\mathbf{C}_{i,j} + \mathbf{G}_{i-1,j}, \;\; \mathbf{C}_{i,j} + \mathbf{G}_{i,j-1}, \;\; 2 \cdot \mathbf{C}_{i,j} + \mathbf{G}_{i-1,j-1}) & i, j > 1. \end{cases} \tag{3}$$

The step pattern (3) is also discussed as special case of the general formulation in Sakoe and Chiba (1978). The direction matrix $\mathbf{D} \in \mathbf{N}^{n \times m}$ gives information about the alignment of

Algorithm 1 Backtracking the direction matrix **D** delivers the warping path **w**.

```
procedure BACKTRACKING(D)
    i ← n                                      ▷ $n$ = length of the time series q
    j ← m                                      ▷ $m$ = length of the time series c
    w, ii, jj ← empty vectors
    repeat
        step ← $\mathbf{D}(i, j)$;
        if step == 1 then
            i ← i - 1
            j ← j - 1
        else if step == 2 then
            j ← j - 1
        else
            i ← i - 1
        end if
        ii ← append(i, ii)
        jj ← append(j, jj)
        w ← append(step, w)
    until $i < 0 \mid j < 0$ return w, ii and jj
end procedure
```

the two time series and is calculated simultaneously with the calculation of **G**. The following equation defines **D** for the step pattern of (2):

$$\mathbf{D}_{i,j} = \begin{cases} 1 & \text{if} \quad \mathbf{G}_{i,j} = \mathbf{C}_{i,j} + \mathbf{G}_{i-1,j-1} \\ 2 & \text{if} \quad \mathbf{G}_{i,j} = \mathbf{C}_{i,j} + \mathbf{G}_{i,j-1} \\ 3 & \text{if} \quad \mathbf{G}_{i,j} = \mathbf{C}_{i,j} + \mathbf{G}_{i-1,j}. \end{cases} \tag{4}$$

The DTW distance measure is stored in the last column of the last row of **G**, $\mathbf{G}_{nm}$, and indicates the cheapest cumulative costs to align **q** and **c**. The warping path vector **w** is an excerpt of the direction matrix **D** and achieved by backtracking **D**. Starting at the last row and last column of **D**, backtracking (Algorithm 1) checks the cheapest next step (1 is diagonal, 2 is horizontal, 3 is vertical) and stores this integer in a vector. The backtracking algorithm also returns the vectors **ii** and **jj**, the vectors of indices of **q** and **c** for the best alignment in the respective order.

2.1. Incremental DTW calculation

Calculating the DTW distance measure is computationally expensive, especially for long time series without a warping window, due to the quadratic runtime complexity $O(n \cdot m)$, where n and m are the lengths of the time series **q** and **c**, respectively. If the DTW distance is calculated with a Sakoe Chiba warping window of size ω, where $|m - n| \leq \omega \leq \max(m, n)$, the runtime complexity reduces to $O(\omega \cdot \min(m, n))$. Consequently, if ω increases and approaches its maximum value, then the runtime complexity approximates the quadratic level, and conversely if ω decreases, then it approximates a linear level. The space complexity is discussed in Section 3.2.

To update an alignment of two time series after recording new observations, it is possible to reuse interim results instead of calculating DTW from scratch. So, if a time series $\mathbf{c}$ is observed for $i = 1 \ldots m$, calculating the DTW distance measure to $\mathbf{q}$ of length n has a runtime complexity[1] of $O(n \cdot m)$. As soon as new observations of $\mathbf{c}$ are recorded for $i = m+1 \ldots m+k$, calculating the DTW distance measure from scratch has a runtime complexity of $O(n \cdot (m+k))$. Contrary the incremental approach is based on storing the necessary components of the results of the initial DTW computation after observing $\mathbf{c}$ for $i = 1 \ldots m$, and recycling these interim results when new observations are recorded. This way the incremental update of the DTW distance at time $i = m + k$ has a runtime complexity of $O(n \cdot k)$. The examples in Section 3.3 and the experiment in Section 4.1 demonstrate this principle in more detail.

The input to incrementally calculate DTW of $\mathbf{q}[1 : n]$ and $\mathbf{c}[1 : m + k]$ is the output of the former calculation $\text{DTW}(\mathbf{q}[1 : n],\ \mathbf{c}[1 : m])$. This output is composed of three matrices: the global cost matrix $\mathbf{G}^0$, the local cost matrix $\mathbf{C}^0$ and the direction matrix $\mathbf{D}^0$. Additional required input is the time series of new observations of $\mathbf{c}$. To calculate the global cost matrix $\mathbf{G}^1$ of $\text{DTW}(\mathbf{q}[1 : n],\ \mathbf{c}[1 : m+k])$, we append new costs and direction entries to the previously calculated matrices and proceed analogously to (2):

1. First we build the local cost Matrix $\mathbf{C}^1$:

$$\mathbf{C}^1_{ij} := \begin{cases} \mathbf{C}^0 ij & i \leq m \\ dist(q_i, c_j) & m < i \leq m + k. \end{cases} \tag{5}$$

2. Next the global cost matrix is appended to the former results and new entries are defined analogously to (2):

$$\mathbf{G}^1_{ij} := \begin{cases} \mathbf{G}^0 ij & i \leq m \\ \sum_{k \leq i} \mathbf{C}_{k,1} & j = 1 \\ \mathbf{C}_{i,j} + \min(\mathbf{G}_{i-1,j},\ \ \mathbf{G}_{i,j-1},\ \ \mathbf{G}_{i-1,j-1}) & else \end{cases} \tag{6}$$

3. The direction matrix $\mathbf{D}^1$ is calculated simultaneously to $\mathbf{G}^1$.

4. Finally, the warping path needs to be calculated completely new from scratch, since in general it can not be excluded that new observations may open up completely different options to warp the two time series.

Equation 6 is the incremental version of (2). For (3) the definition of the new entries of $\mathbf{G}$ is analogous, as for any other step pattern presented in Sakoe and Chiba (1978).

In fact, not the complete matrix $\mathbf{G}^0$ is required to update the DTW distance for new observations. Section 3.2 and 3.3 discuss an vector-based implementation for the incremental calculation that only requires the very last column (and row) of $\mathbf{G}^0$.

Especially for live streaming data computation time is key. **IncDTW** (in particular the functions `increment()`, `idtw2vec()` and `idtw()` as demonstrated in Section 2 and 4) facilitates a fast update of time series distance measures when new observations arise. This can be of interest for any system analyzing live data streams.

---

[1] For simplicity we reduce the following runtime complexity discussion for the general case of DTW calculation without a warping window. The derivation for DTW calculation with a warping window follows analogous arguments.

3. The R package IncDTW

This section describes the functions of the R package **IncDTW** and how to apply them to calculate the DTW distance: (1) Matrix-based, (2) vector-based, and (3) from scratch or incrementally. For details about further functionalities of **IncDTW** (e.g., an algorithm for searching the k-nearest subsequences of a time series with DTW, or time series clustering with DTW) we refer to the package documentation and vignette (Leodolter 2021). All results presented in this paper are achieved with version 1.1.4.3 of **IncDTW**. The computationally expensive parts of **IncDTW** are outsourced to C++ via the packages **Rcpp** (Eddelbuettel and François 2011) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014), and parallelized via the packages **parallel** (R Core Team 2021) and **RcppParallel** (Allaire *et al.* 2021).

3.1. Matrix-based implementation

The classical DTW implementation relies on the local cost matrix $\mathbf{C}$, the direction matrix $\mathbf{D}$ and the global cost matrix $\mathbf{G}$ (see Section 2). $\mathbf{C}$ can be stored as matrix or calculated entry-wise when $\mathbf{G}$ is calculated. Returning the matrices $\mathbf{G}$ and $\mathbf{D}$ facilitates a detailed analysis of the alignment of two time series. Plotting Figure 2 for visual analysis is possible due to the information provided by the warping path, which in turn is an excerpt of the direction matrix $\mathbf{D}$ and is achieved by backtracking. The entry $\mathbf{C}_{ij}$ is the distance between q_i and c_j and can be described by any distance metric for univariate or multivariate time series dependent on the dimension of $\mathbf{q}$ and $\mathbf{c}$. In case of multivariate time series, they need to have the same dimension, but still can vary in number of observations. In the univariate case the 1-norm is equivalent to the 2-norm, which is the absolute value of the difference $|q_i - c_j|$.

The basic DTW algorithm for computing the global cost matrix $\mathbf{G}$, according to (2), steps through the local cost matrix $\mathbf{C}$. The following parameters characterize in detail how the algorithm defines $\mathbf{G}$ and finds the warping path:

- `dist_method`: The local distances are stored in $\mathbf{C}$, where $\mathbf{C}_{ij} = \texttt{dist\_method}(q_i, c_j)$. So the parameter `dist_method` defines how the local distance of observations are measured. For O-dimensional time series the distances “norm1”, “norm2” and “norm2_square” are defined as:

$$\begin{aligned} ||\mathbf{q}_i, \mathbf{c}_j||_1 &:= \sum_{o=1}^{O} |\mathbf{q}_{io} - \mathbf{c}_{jo}| \\ ||\mathbf{q}_i, \mathbf{c}_j||_2 &:= \sqrt{\sum_{o=1}^{O} (\mathbf{q}_{io} - \mathbf{c}_{jo})^2} \\ ||\mathbf{q}_i, \mathbf{c}_j||_2^2 &:= \sum_{o=1}^{O} (\mathbf{q}_{io} - \mathbf{c}_{jo})^2. \end{aligned} \tag{7}$$

  Apart from these three predefined local distance functions **IncDTW** also allows to define customized local distance functions.

- `ws`: The space of all possible alignments of two time series can be constrained by warping windows. As Section 2 mentions, the most popular constraint is the Sakoe-Chiba window (Sakoe and Chiba 1978), which adjusts the DTW algorithm by setting $\mathbf{G}_{ij} = \infty$

if $|i-j| > \texttt{ws}$. So `ws` defines the window size of allowed warping paths. If we set $\texttt{ws} = 0$ then only the diagonal of $\mathbf{G}$ is used for aligning $\mathbf{q}$ and $\mathbf{c}$, which is identical to the Euclidean distance. In this case the time series must have the same length. If the lengths of the time series differ more than `ws`, then obviously no valid alignment can be found.

- `step_pattern`: The step pattern defines how the DTW algorithm finds the cheapest path through the local cost matrix. In (2) the most basic and broadly applied step pattern "symmetric1" is used, where the direct neighbors are considered and all are weighted equally. In (3) the step pattern "symmetric2" is uses a weight of 2 for the diagonal step and 1 for the vertical and horizontal to compensate the favor of diagonal steps. The current version of **IncDTW** concentrates on these two patterns and we consider other step patterns for future developments. For a more detailed discussion of step patterns we refer to Giorgino *et al.* (2009) and Rabiner and Juang (1993).

The following commands install and load the package **IncDTW**:

```
R> install.packages("IncDTW")
R> library("IncDTW")
```

First we define the help function `rw()` (which we also use in the next sections) to simulate a Gaussian random walk. Then a basic calculation of the DTW distance is done as follows:

```
R> rw <- function(n) cumsum(rnorm(n))
R> Q <- rw(100)
R> C <- rw(80)
R> result <- dtw(Q, C, ws = 30, step_pattern = "symmetric2")
R> result$distance
```

```
[1] 197.1266
```

3.2. Vector-based implementation

The matrix-based implementation is necessary for a detailed analysis of the alignment of two time series since it allows to calculate and return the warping path. Tasks such as nearest neighbor search, or the calculation of a matrix of pairwise distances to cluster or classify a database of time series require many DTW computations, and so the computation time of DTW is a major bottleneck.

The vector-based implementation offers a solution which is faster than the matrix based implementation, since memory allocation for matrices is not required. The space complexity for the matrix-based implementation is $O(m \cdot n)$ for calculating the local and global cost matrix, and the direction matrix. The vector-based computation principle is the same as for the matrix-based method, but instead of allocating matrices only two vectors are needed, and so the space complexity is reduced to $O(n)$. To obtain the DTW distance between the time series $\mathbf{q}$ and $\mathbf{c}$ the calculation of the j-th column of the global cost matrix $\mathbf{G}_{.,j}$ solely depends on the values of the previous column $\mathbf{G}_{.,j-1}$ and the respective distances of the time series $\mathbf{c}$ and the j-th entry of $\mathbf{q}$. Since there is no dependence on the column $\mathbf{G}_{.,j-2}$, the algorithm overwrites $\mathbf{G}_{.,j-2}$ with the newly calculated vector $\mathbf{G}_{.,j}$. Figure 1 demonstrates this principle

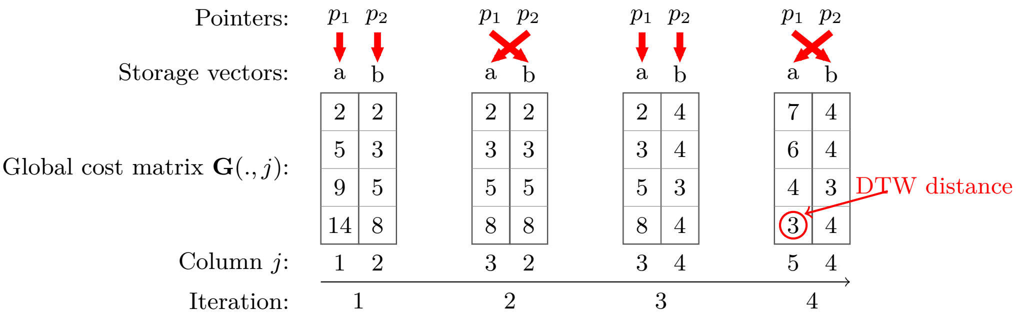


Figure 1: Iteratively overwriting vectors makes it obsolete to allocate matrices for DTW distance calculation.

with a simple example, and the following lines of code perform the DTW calculation for the same time series first via matrix based implementation (`dtw`) and second via vector based implementation (`dtw2vec`). The global cost matrix $\mathbf{G}$ is also printed to compare it to the vectors illustrated in Figure 1.

```
R> Q <- c(3,4,5,6)
R> C <- c(1,3,3,5,6)
R> result <- IncDTW::dtw(Q,C)
R> result$gcm
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    2    2    4    7
[2,]    5    3    3    4    6
[3,]    9    5    5    3    4
[4,]   14    8    8    4    3
```

```
R> dtw2vec(Q,C)$distance
```

```
[1] 3
```

In the first iteration step in Figure 1 the initial two vectors a and b are defined according to the DTW step pattern and are identical to the first two columns of $\mathbf{G}$. In the second iteration the pointers p_1 and p_2 switch the address, so that the new entries of $\mathbf{G}_{.,3}$ overwrite a (where p_2 points to) and b (where p_1 points to) stores the entries of $\mathbf{G}_{.,2}$ of the previous iteration. Finally after four iterations the DTW distance measure (red encircled) is given in the last row of the last vector, which is identical to the fifth column of $\mathbf{G}$. Algorithm 2 formalizes this principle for the general case.

Even though the information about the warping alignment is lost by applying the vector-based method, the warping path still can be constrained by the parameter `ws`, defining the Sakoe Chiba warping window size. To continue with the same time series we constrain the warping path to allow a maximum deviation of the time index of $\mathbf{q}$ and $\mathbf{c}$ of 1, so $|i - j| \leq 1$. Since the warping path needs to adapt slightly the calculated distance changes from 3 to 4.

Algorithm 2 Vector based implementation of DTW without allocating any matrices.

```
procedure VECTOR BASED DTW($\mathbf{q} \in \mathbb{R}^{n \times O}$, $\mathbf{c} \in \mathbb{R}^{m \times O}$)
    p1 ← cumsum(dist($q_1$, $\mathbf{c}$))            ▷ initial column of $\mathbf{G}$, $\mathbf{G}_{.,1}$
    for j in 2:m do
        $p2[1] \leftarrow$ dist($q_1$, $c_j$) + $p1[1]$
        for i in 2:n do
            $p2[i] \leftarrow$ step( dist($q_i$, $c_j$), min($p2[i-1]$, $p1[i]$, $p1[i-1]$))
        end for
        ptmp ← p1
        p1 ← p2
        p2 ← ptmp
    end for
    return p1[n]
end procedure
```

```
R> IncDTW::dtw2vec(Q, C, ws = 1)$distance

[1] 4
```

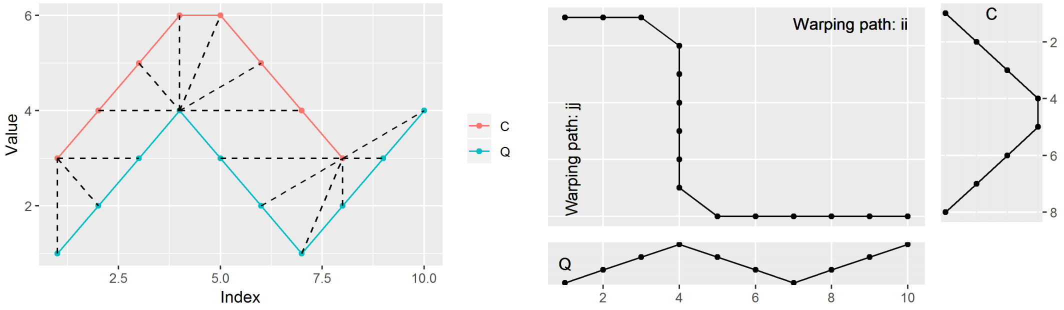
“Early abandoning” is a pruning method to break calculations if the cheapest possible alignment of two time series hits an upper bound (set by the user). This method helps to lower the calculation runtime when comparing many time series. If the DTW algorithm hits this threshold the for-loop breaks and returns `NaN`. We continue the example and set the threshold to 2. Since no value in the fourth column of the global cost matrix is smaller or equal to 2, so $\mathbf{G}_{i,4} > 2 \ \forall i$, the calculation stops here and `NaN` is returned.

```
R> IncDTW::dtw2vec(Q, C, threshold = 2)$distance

[1] NaN
```

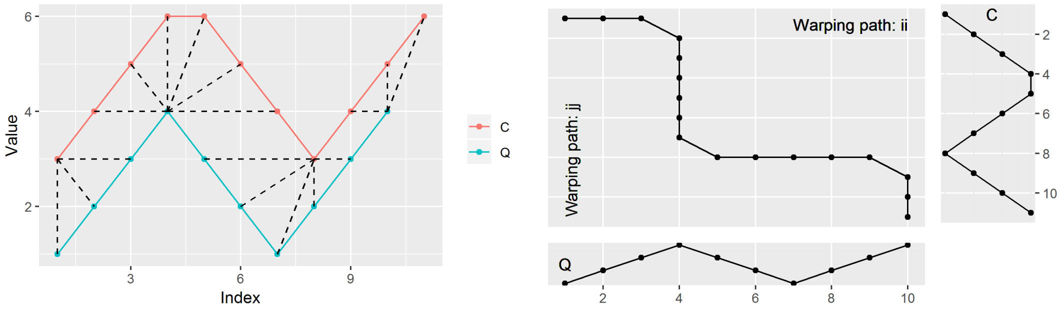
3.3. IncDTW for incremental DTW calculation

For the incremental calculation of DTW we can choose between (1) the matrix based implementation to get more information about the alignment of the two time series and to facilitate analyses of the warping paths and (2) the vector based implementation for a faster distance calculation. For the latter the initial column in Algorithm 2 is defined as the last column of the former calculated global cost matrix, the last pointer vector respectively. That is, instead of passing matrices as input to the incremental DTW function, only the last column vector of $\mathbf{G}$ is passed for the vector based implementation. Further the class ‘`planedtw`’ and its methods deal as convenient wrapper functions around the vector based implementation. For a better understanding the following examples first walk through the more basic matrix based and vector based incremental update, and finally present the incremental update by hand of the ‘`planedtw`’ class.

We demonstrate the principle of incrementally updating the DTW global distance matrix and the distance measure by hand of the following example. We define the time series $\mathbf{q}$ and $\mathbf{c}$, and calculate the initial alignment with `dtw()`.



(a) Initial time series (b) Initial warping path



(c) Updated time series (d) Updated warping path

Figure 2: Initially **c** and **q** are aligned. The warping path has 4 vertical steps before the update of **c**. The alignment is updated with an update of **c**.

```
R> Q <- c(1:3, 4:1, 2:4)
R> C_initial <- c(1:3, 4, 4,  3:1) + 2
R> align_initial <- IncDTW::dtw(Q = Q, C = C_initial, return_wp = TRUE,
+    return_QC = TRUE, step_pattern = "symmetric1")
```

Figure 2a shows the time series and the aligned observations connected with dashed lines, and Figure 2b contains the same information but focuses on the warping path (the main plot). One can see that the last observation of **c** is matched to the final six observations of **q**. We plotted the results with `plot(align_initial, type = "warp")` and `type = "QC"` respectively.

With new observations of **c** we can easily update the global cost matrix and the warping path by applying `idtw()` and compare the initial and updated versions of **G**.

```
R> C_newObs <- Q[8:10] + 2
R> C_update <- c(C_initial, C_newObs)
R> align_inc <- IncDTW::idtw(Q = Q, C = C_initial, newObs = C_newObs,
+    gcm = align_initial$gcm, dm = align_initial$dm, return_wp = TRUE,
+    return_QC = TRUE, step_pattern = "symmetric1")
R> identical(align_inc$gcm[, 1:8], align_initial$gcm)

[1] TRUE
```

As expected the first eight columns of the updated **G** and the initial **G** are identical. Figure 2c and 2d show the updated alignment and warping path. Finally, we compare the DTW distance of the updated calculation with the one from scratch (again using the basic function `dtw()`) and see that they are equal:

```
R> align_scr <- IncDTW::dtw(Q = Q, C = C_update, return_wp = TRUE,
+    return_QC = TRUE, step_pattern = "symmetric1")
R> align_scr$distance - align_inc$distance

[1] 0
```

We continue with the former example and perform the incremental calculation with the vector based implementation with `idtw2vec()`. This function distinguishes between an initial calculation and the incremental by checking whether results of previous calculations are passed or not, particularly the argument `gcm_lc`.

```
R> alignV_init <- IncDTW::idtw2vec(Q = Q, newObs = C_initial, gcm_lc = NULL)
R> alignV_inc  <- IncDTW::idtw2vec(Q = Q, newObs = C_newObs,
+    gcm_lc = alignV_init$gcm_lc_new)
```

Finally we compare the DTW distances of the incremental calculation (`idtw2vec()`) with the one from scratch (`dtw2vec()`) and their matrix based counterparts. As expected they are identical:

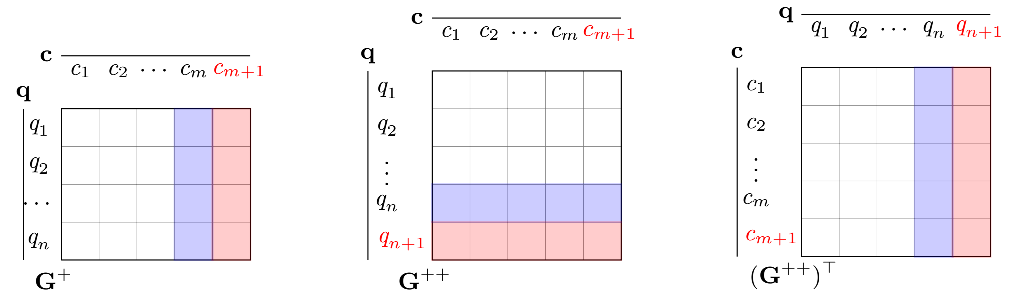
```
R> C_update <- c(C_initial, C_newObs)
R> alignV_scr <- IncDTW::dtw2vec(Q = Q, C = C_update)
R> c(align_scr$distance,  align_inc$distance,
+    alignV_scr$distance, alignV_inc$distance)

[1] 16 16 16 16
```

Section 4.2 gives runtime comparisons for these update functions.

New observations for both time series

With the knowledge of the basics and main modules for incremental calculation of DTW, `idtw()` and `idtw2vec()`, we apply the functions `initialize_plane()` and `increment()` which are convenient wrappers around `idtw2vec()`. The former function performs the initial calculation of `idtw2vec()` and returns an object of class '`planedtw`', whereas the latter function applies the incremental calculation of `idtw2vec()`. The package vignette (Leodolter 2021) discusses further methods for the S3 class '`planedtw`' that support the navigation in the plane of possible fits, which means to adjust incrementally the partial alignment of two time series.

If new observations for both time series are available, the update of the DTW calculation works in a consecutive fashion, similar to the case where only one time series is updated. The initial step is to apply `initialize_plane()` on the initial observations of **c** and **q**. Next we update the calculations for the first time series with `increment()`:



(a) New observations for **c**. (b) New observations for **q**. (c) Transposed, new observations for **q**.

Figure 3: Incremental update of **G** for new observations of **c** and **q**. The updated areas of **G** are coloured in red and the areas storing the required input for the vector based update calculation are coloured blue.

```
R> x <- initialize_plane(Q = Q, C = C_initial)
R> print(x)

control:
 dist_method step_pattern nQ nC   ws reverse
       norm1   symmetric2 10  8 NULL   FALSE

DTW distance:
14

Normalized DTW distance:
0.7777778

R> x <- increment(x, newObs = C_newObs)
```

Figure 3a visualizes relevant sections of the updated global cost matrix **G**. For a new observation of **c** the new area of **G** is colored red and the required column for the update in blue. Next we update **G** for the new observations of **q**. Again the red and blue rows in Figure 3b indicate the updated and required areas. So we switch places of **q** and **c** as input for `idtw2vec()` and proceed analogously. Also we need to switch the last column with the last row of the global cost matrix. Figure 3c illustrates that switching **c** and **q** and the `gcm_lr` with `gcm_lc` is the same as transposing **G**. We could either switch the positions of these elements by hand and apply `idtw2vec()` directly, or apply the more convenient function `increment()` and set `direction = "Q"` to tell the function in which direction to update the last row and column of the global cost matrix:

```
R> Q_newObs <- rw(10)
R> x <- increment(x, newObs = Q_newObs, direction = "Q")
```

Finally we compare the results with the results from scratch and see that the calculated distance measures are equal:

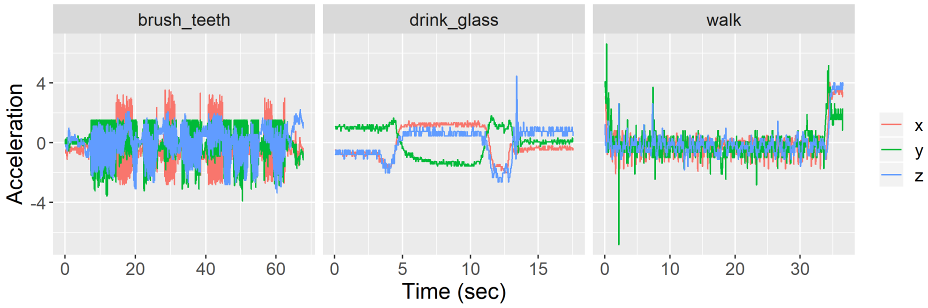


Figure 4: Typical accelerometer time series recorded while brushing teeth, drinking a glass, or walking.

```
R> x$distance - dtw2vec(c(Q, Q_newObs), c(C_initial, C_newObs))$distance
```

```
[1] 0
```

4. Applying IncDTW

This section demonstrates the applicability of **IncDTW** by (1) discussing a time series classification task for live data streams solved by either the traditional DTW implementation `dtw2vec()` or the incremental updating of DTW distances to speed up calculations with `idtw2vec()`, and (2) discussing runtime experiments that compare **IncDTW** with other R and Python packages.

In the following experiment we work with data sets (Bruno, Mastrogiovanni, Sgorbissa, Vernazza, and Zaccaria 2013) downloaded from UCI machine learning repository (Dheeru and Karra Taniskidou 2017). The data was collected by participants wearing a smart watch recording a 3-dimensional accelerometer signal with a sampling rate of 32 Hz. Among other actions the participants were asked to collect data during walking (`walk`), drinking a glass (`drink_glass`) and brushing teeth (`brush_teeth`), Figure 4 depicts examples of the three activities. The time series data of these experiments are included in the package **IncDTW**. The package documentation and vignette (Leodolter 2021) also include further experiments about time series clustering and scanning longer time series to detect similar representations of a shorter query pattern.

4.1. Incremental DTW update for live data

When applying data mining methods on live streams of data, it is mandatory that the computation time of the analysis is smaller than the time in between two consecutive observations. In this experiment we simulate the situation of dealing with data streams by iteratively including more observations of the time series into analysis. As soon as new observations are "recorded" we classify the time series streams by comparing their DTW distances to prototype patterns, so we need to update the DTW calculation for each set of new observations.

We start this experiment with determining representative centroid patterns for each of the recorded activities, stored in the accelerometer data sets `walk`, `drink_glass` and `brush_teeth`.

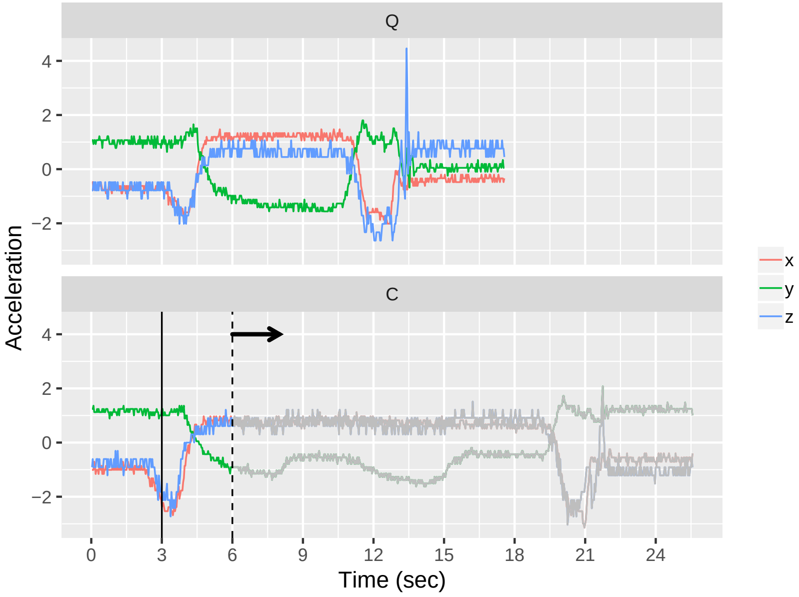


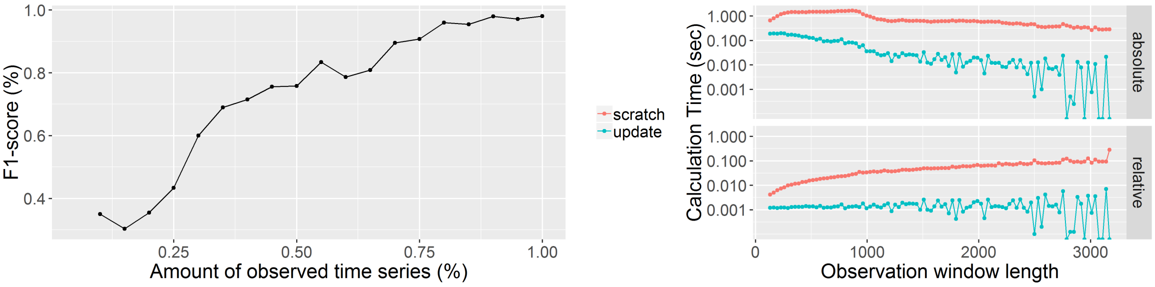
Figure 5: Iteratively increasing the observation window. As the dashed line moves to the right, more data is included in analysis and the DTW alignment is updated for the new observations.

We calculate these representatives with `IncDTW::dba()`, which is the DTW Barycenter Averaging method by Petitjean, Ketterlin, and Gançarski (2011) for averaging multiple time series that are non-linearly aligned by DTW.

Next we calculate the initial DTW distances for the first 100 observations (about 3 seconds) of each time series of the three data sets to the three centroids. Then we simulate the continuous recording of new observations and apply `idtw2vec()` to update the DTW distance measures, which requires to store the last columns of $\mathbf{G}$ (see (2)) of the previous calculations. For comparing the computation times we fulfill the same classification task with `dtw2vec()`, and of course the classification results are identical. Figure 5 depicts this simulation of a data stream $\mathbf{c}$ and the query time series $\mathbf{q}$, both selected from the `drink_glass` data set. This plot shows the situation after the initial step – the first three seconds are already observed (vertical solid line) – when $\mathbf{c}$ has already been recorded for six seconds in total (the vertical dashed line). As the data stream continuously updates the dashed line moves to the right and more observations are included to the DTW alignment with $\mathbf{q}$.

Figure 6a plots the classification accuracy against the "observed" (used) percentage of the time series, and shows that the accuracy increases the more observations are recorded. Already about 75% are enough to reach an F1-score of 90%. We used 4-fold cross validation, where we calculated the representatives via `dba()` on one fold and classified the remaining 3 folds. Figure 6a shows aggregated results.

Figure 6b compares the computation times of `idtw2vec()` (incremental) and `dtw2vec()` (from scratch) to process one set of new observations, which we represent as the set of observations recorded within one second, so 3-dimensional time series with 32 rows (since originally recorded with 32Hz). The collection of these three data sets consists of 212 time series of different lengths. The calculation times depend on the length of the observation window and



(a) Prediction Accuracy.

(b) Runtime: Absolute (top) for all time series, and relative (bottom) per time series.

Figure 6: Prediction accuracy and computation time comparison for classifying multivariate time series of the data sets `walk`, `drink_glass` and `brush_teeth` by simulating to observe these time series live and update the prediction once per second.

the number of time series that are at least as long as the observation window. Since the time series are of different lengths, with increasing observation window, more and more time series can not be processed further until the observation window is equal to the length of the longest time series. For this reason the graph for "scratch" in Figure 6b (top) first rises and then drops continuously. All time series are at least 187 observations long and beyond this observation window length the shorter time series drop out of further analysis and so are not relevant for the total computation time. For clarification we also plot the relative times per time series in Figure 6b (bottom). It is worth mentioning that the y-axis are log-scaled.

We conclude that the incremental update can process about 7 to 108 times more time series than the calculation from scratch, dependent on the length of the time series, the observation window respectively. This exemplary data analysis task would not be solvable in time by applying `dtw2vec()` (which is vector-based implemented in C++ via **Rcpp**) since the calculation of DTW distances and classification takes longer than one second, which is the time in-between two sets of new observations. However, the incremental method with `idtw2vec()` is capable. As expected this experiment demonstrates the calculation time for the incremental step to be independent of the total length of the time series, see Figure 6b. We performed this experiment applying a single core of a 2.8 GHz and 16GB RAM laptop. If we split the work for this example across multiple cores `dtw2vec()` would manage the classification in time as well, however the relation of 7 to 108 remains the same, so the incremental solution is capable to deal with much more time series updates in less time.

4.2. Runtime comparisons

In the following we compare computation times for the 3 data analysis tasks: (1) the incremental update for new observations, (2) single DTW computation for two time series, and (3) computing the matrix of pairwise DTW distances for a set of time series. Further, we also compare **IncDTW** with Python packages for the second task, since this is probably the most generic and most applied use case. To compare the calculation times of R packages we use the package **microbenchmark** (Mersmann 2019). For comparisons to Python packages we measure the wall clock time. To the best of our knowledge we set the parameters of all functions so that a fair computation time comparisons is guaranteed. So we omit to return additional output objects (like the warping path) which obviously would cause higher computation times. All runtime experiments were performed on a standard laptop computer with 2.8 GHz and 16GB RAM. We applied the following versions of the respective packages (please see Section 1 and Table 1 for more details about the packages):

- R: **IncDTW** (1.0.4), **dtw** (1.22-3), **dtwclust** (5.5.6), **rucrdtw** (0.1.4), **parallelDist** (0.2.4)
- Python: **cydtw** (0.1.4), **dtaidistance** (1.2.3), **dtw** (1.4.0).

Incremental update of DTW

This paper emphasizes methods for accelerating DTW calculations and demonstrates how to apply the incremental DTW calculation for updating existing results for new observations (Section 2.1 and 4.1). The following experiment underpins that this principle of recycling former calculated results is a considerable faster approach to compute the DTW distance measure. For this experiment we simulate the situation of continuously recording new observations and compare the runtime for the incremental calculation with a traditional calculation from scratch. Figure 7a shows the results. Each red point is the median of 100 computations of the DTW distance with `dtw2vec()` of two univariate time series, both of the respective length given at the x-axis. The blue points visualize the median computation time for one incremental step (via `idtw2vec()`), so one new observation of $\mathbf{c}$, and $\mathbf{q}$ of length as given by the x-axis. Both axes are in log scale.

Single computations

Figure 7b depicts the runtime comparison in a log-scale. The only two methods using a vector-based implementations (as discussed in Section 3.2) are `rucrdtw::ucrdtw_vv()` and `IncDTW::dtw2vec()`, and these are considerably faster than the remaining functions. To guarantee a fair comparison we set the step pattern to “symmetric1” (since `rucrdtw::ucrdtw_vv()` only supports “symmetric1”) and the warping window size equal 10 for all functions.

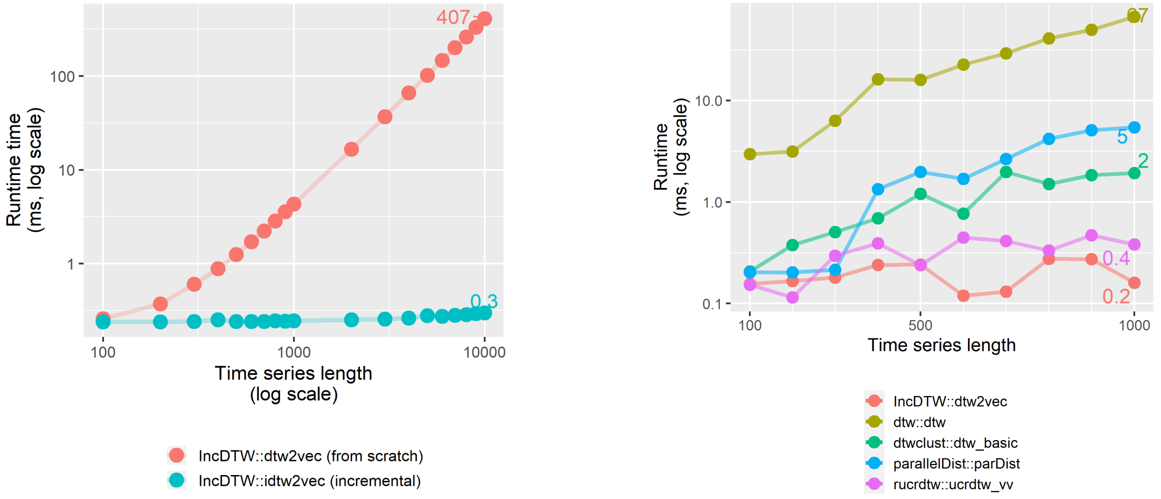
Compute a distance matrix

Time series clustering is a typical task in time series analysis and data mining. Time series clustering based on the DTW distance measure requires a distance matrix of pairwise DTW distances. The function `IncDTW::dtw_dismat()` helps to get this matrix for a list of univariate or multivariate time series of possibly different lengths. The calculations can be performed single threaded (`ncores = 1`) or multithreaded.

We compare the runtimes for calculating distance matrices for a set of 500 time series of varying lengths and also set the window size parameter to 10. Figure 7c depicts the runtimes, where `dtw_dismat_1()` is the standard function `dis_mat()` without parallelization. `dtw_dismat_3Rcpp()` splits the work via **RcppParallel** and `dtw_dismat_3R()` uses the package **parallel**, both with three cores (`ncores = 3`). For short time series `parDist_3()` is up to 10 times faster than `dtw_dismat_3Rcpp()` and for long time series it’s the other way round (about 17 times faster).

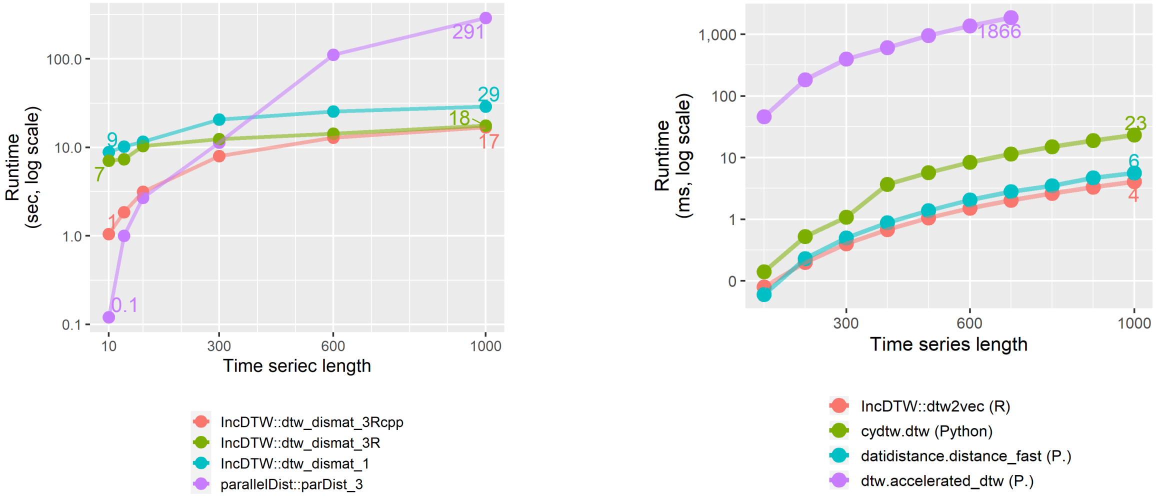
Comparison with Python

For many data analysis tasks R and Python are interchangeable and it is just a matter of



(a) Incremental vs. from scratch.

(b) Single DTW computations for two time series.



(c) Matrices of pairwise DTW distances for a list of time series.

(d) Single DTW without warping window compared with Python packages.

Figure 7: Runtime comparisons for different data analysis tasks.

taste which to prefer. So, we compare the runtimes for calculating the DTW distance across these two platforms. For each of the time series lengths we measured the wall clock time for 100 DTW computations in the respective programming language environment[2], and averaged it. To guarantee a fair comparison we omit the warping window parameter since the function `cydtw.dtw()`[3] does not support warping windows. Figure 7d shows the results in log scale. The functions `dtaidistance.distance_fast()` and `cydtw.dtw()` both are functions

---

[2] We also performed the experiment by calling the Python functions inside of R via **reticulate** (Ushey, Allaire, and Tang 2021), which caused a computation overhead.

[3] We notate R and Python functions according to their syntax: `package::function()` in R and `package.function()` in Python.

written in C, via **Cython**, but only the former is vector based and so it is comparable fast as `IncDTW::dtw2vec()`.

5. Conclusion

This paper discusses the incremental calculation of the widely applied DTW distance measure (Fu 2011). We present the R package **IncDTW** (Leodolter 2021) – current version 1.1.4.3 available from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=IncDTW` – that mainly focuses on fast R functions for vector based and incremental DTW computation. **IncDTW** also offers functions for familiar time series analysis tasks, as time series clustering and pattern recognition. Section 4.1 showcases how to apply **IncDTW** to classify three dimensional time series in a simulated live stream setting, and why the incremental calculation of DTW is capable to process 7 to 108 times more data.

Due to the intensive computational costs of DTW, we put a special emphasis on accelerating our algorithms. Consequently, **IncDTW** transfers the most intensive computations to C++ via **Rcpp** and stresses on the one hand the vector based implementation, and on the other hand the principle of the incremental calculation of DTW, by recycling previous calculation results. Section 4.2 demonstrates the benefits of these acceleration methods using runtime comparisons for various settings. Further accelerating methods as lower bounding (Keogh, Wei, Xi, Lee, and Vlachos 2006; Rath and Manmatha 2003a) and early abandoning methods are also applied and discussed in more detail in the package vignette (Leodolter 2021).

Apart from stream processing, computation time is also key whenever relatively short query patterns must be detected in longer time series, which usually requires a large number of comparisons between many segments of the longer time series and the query pattern. For example, the Caterpillar algorithm presented by Leodolter, Brändle, and Plant (2018) scans long time series to detect patterns which are possibly warped or of different lengths than a query pattern, based on a combination of incremental DTW calculation and the Minimum Description Length. The incremental calculation of DTW enables the Caterpillar algorithm to search the space of possible fits runtime efficiently. So, the R package **IncDTW** and its functions can serve as components for building pattern recognition algorithms.

Future developments for **IncDTW** will incorporate a parallelized implementation of `dba()` and a user-friendly solution for applying lower bounding, which is currently only implemented as part of `rundtw()`.

References

Allaire JJ, François R, Ushey K, Vandenbrouck G, Geelnard M, Intel (2021). ***RcppParallel**: Parallel Programming Tools for **Rcpp***. R package version 5.1.4, URL `https://CRAN.R-project.org/package=RcppParallel`.

Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K (2011). "**Cython**: The Best of Both Worlds." *Computing in Science Engineering*, **13**(2), 31 –39. `doi:10.1109/mcse.2010.118`.

Berndt DJ, Clifford J (1994). "Using Dynamic Time Warping to Find Patterns in Time Series." In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, pp. 359–370. AAAI Press. URL `http://dl.acm.org/citation.cfm?id=3000850.3000887`.

Boersch-Supan P (2016). "**rucrdtw**: Fast Time Series Subsequence Search in R." *The Journal of Open Source Software*, **1**, 1–2. `doi:10.21105/joss.00100`.

Bruno B, Mastrogiovanni F, Sgorbissa A, Vernazza T, Zaccaria R (2013). "Analysis of Human Behavior Recognition Algorithms Based on Acceleration Data." In *IEEE International Conference on Robotics and Automation 2013*, pp. 1602–1607. IEEE.

Dheeru D, Karra Taniskidou E (2017). "UCI Machine Learning Repository." URL `http://archive.ics.uci.edu/ml`.

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008). "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures." *Proceedings of the VLDB Endowment*, **1**(2), 1542–1552. `doi:10.14778/1454159.1454226`.

Dixon S (2005). "An On-Line Time Warping Algorithm for Tracking Musical Performances." In *IJCAI*, pp. 1727–1728.

Eckert A (2018). ***parallelDist**: Parallel Distance Matrix Computation Using Multiple Threads.* R package version 0.2.4, URL `https://CRAN.R-project.org/package=parallelDist`.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. `doi:10.18637/jss.v040.i08`.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063. `doi:10.1016/j.csda.2013.02.005`.

Fu T (2011). "A Review on Time Series Data Mining." *Engineering Applications of Artificial Intelligence*, **24**(1), 164–181. `doi:10.1016/j.engappai.2010.09.007`.

Giorgino T, *et al.* (2009). "Computing and Visualizing Dynamic Time Warping Alignments in R: The **dtw** Package." *Journal of Statistical Software*, **31**(7), 1–24. `doi:10.18637/jss.v031.i07`.

Keogh E (2002). "Exact Indexing of Dynamic Time Warping." In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pp. 406–417. Elsevier.

Keogh E, Wei L, Xi X, Lee SH, Vlachos M (2006). "LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures." In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pp. 882–893. VLDB Endowment.

Keogh EJ, Pazzani MJ (2000). "Scaling up Dynamic Time Warping for Datamining Applications." In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pp. 285–289. ACM, New York. `doi:10.1145/347090.347153`.

Kwankhoom W, Muneesawang P (2017). "An Incremental Dynamic Time Warping for Person Re-Identification." In *14th International Joint Conference on Computer Science and Software Engineering 2017*, pp. 1–5. IEEE.

Leodolter M (2021). ***IncDTW**: Incremental Calculation of Dynamic Time Warping.* R package version 1.1.4.3, URL `https://CRAN.R-project.org/package=IncDTW`.

Leodolter M, Brändle N, Plant C (2018). "Automatic Detection of Warped Patterns in Time Series: The Caterpillar Algorithm." In *IEEE International Conference on Big Knowledge 2018*, pp. 423–431. `doi:10.1109/icbk.2018.00063`.

Maus V, Câmara G, Appel M, Pebesma E (2019). "**dtwSat**: Time-Weighted Dynamic Time Warping for Satellite Image Time Series Analysis in R." *Journal of Statistical Software*, **88**(5), 1–31. `doi:10.18637/jss.v088.i05`.

Meert W (2017). ***dtaidistance***. URL `https://pypi.org/project/dtaidistance/`.

Mersmann O (2019). ***microbenchmark**: Accurate Timing Functions.* R package version 1.4-7, URL `https://CRAN.R-project.org/package=microbenchmark`.

Mori A, Uchida S, Kurazume R, Taniguchi R, Hasegawa T, Sakoe H (2006). "Early Recognition and Prediction of Gestures." In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pp. 560–563. `doi:10.1109/icpr.2006.467`.

Oregi I, Pérez A, Del Ser J, Lozano JA (2017). "On-Line Dynamic Time Warping for Streaming Time Series." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 591–605. Springer-Verlag.

Petitjean F, Ketterlin A, Gançarski P (2011). "A Global Averaging Method for Dynamic Time Warping, with Applications to Clustering." *Pattern Recognition*, **44**(3), 678–693. `doi:10.1016/j.patcog.2010.09.013`.

Rabiner L, Juang BH (1993). *Fundamentals of Speech Recognition.* Prentice-Hall, Upper Saddle River.

Rabiner L, Rosenberg A, Levinson S (1978). "Considerations in Dynamic Time Warping Algorithms for Discrete Word Recognition." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **26**(6), 575–582. `doi:10.1109/tassp.1978.1163164`.

Rakthanmanon T, Campana B, Mueen A, Batista G, Westover B, Zhu Q, Zakaria J, Keogh E (2012). "Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping." In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 262–270. ACM. `doi:10.1145/2339530.2339576`.

Rath TM, Manmatha R (2003a). "Lower-Bounding of Dynamic Time Warping Distances for Multivariate Time Series." *MM 40*, University of Massachusetts Amherst.

Rath TM, Manmatha R (2003b). "Word Image Matching Using Dynamic Time Warping." In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2003*, volume 2, pp. II–II. IEEE. `doi:10.1109/cvpr.2003.1211511`.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.R-project.org/`.

Rouanet P (2014). ***dtw***. URL `https://pypi.org/project/dtw/`.

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. `doi:10.1109/tassp.1978.1163055`.

Sakurai Y, Faloutsos C, Yamamuro M (2007). "Stream Monitoring under the Time Warping Distance." In *IEEE 23rd International Conference on Data Engineering 2007*, pp. 1046–1055. IEEE.

Sarda-Espinosa A (2019). ***dtwclust**: Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance*. R package version 5.5.6, URL `https://CRAN.R-project.org/package=dtwclust`.

Tavenard R (2017). ***cydtw***. URL `https://pypi.org/project/cydtw/`.

Tormene P, Giorgino T, Quaglini S, Stefanelli M (2008). "Matching Incomplete Time Series with Dynamic Time Warping: An Algorithm and an Application to Post-Stroke Rehabilitation." *Artificial Intelligence in Medicine*, **45**(1), 11–34. `doi:10.1016/j.artmed.2008.11.007`.

Ushey K, Allaire JJ, Tang Y (2021). ***reticulate**: Interface to Python*. R package version 1.20, URL `https://CRAN.R-project.org/package=reticulate`.

Van Rossum G, *et al.* (2011). *Python Programming Language*. URL `https://www.python.org/`.

Wang X (2011). "A Fast Exact k-Nearest Neighbors Algorithm for High Dimensional Search Using k-Means Clustering and Triangle Inequality." In *The 2011 International Joint Conference on Neural Networks*, pp. 1293–1299. IEEE.

Affiliation:

Maximilian Leodolter
Austrian Institute of Technology
Center for Mobility Systems
1210 Wien, Austria
E-mail: `maximilian.leodolter@gmail.com`

Journal of Statistical Software `http://www.jstatsoft.org/`
published by the Foundation for Open Access Statistics `http://www.foastat.org/`

September 2021, Volume 99, Issue 9 *Submitted:* 2019-04-08
`doi:10.18637/jss.v099.i09` *Accepted:* 2020-03-23