– ORIGINAL ARTICLE –

EudypTS: A Web-based tool for the visual comparison of time-series

Facundo A. Alvarado [1], Dana K. Urribarri [1, 2, 3], Martín L. Larrea [1, 2, 3]

[1]*Department of Computer Science and Engineering, Universidad Nacional del Sur (UNS), Argentina*
facundoalvarado9@gmail.com, {dku, mll}@cs.uns.edu.ar
[2]*Computer Graphics and Visualization R&D Laboratory, Universidad Nacional del Sur (UNS) - CIC Prov. Buenos Aires, Argentina*
[3] *Institute for Computer Science and Engineering, Universidad Nacional del Sur (UNS) - CONICET, Argentina*

Abstract

EudypTS is a web-application that allows for the visual comparison of time-series datasets using a Dynamic-Time-Warping-based visualization technique developed by researchers at the Universidad Nacional del Sur in Argentina. EudypTS offers user-interaction capabilities such as dataset editing, distance-measure selection and interactive chart controls. It also leverages web-workers and memoization for an efficient processing and rendering of the comparison results. The following paper provides an overview of the base visualization technique and an insight on the design and development of EudypTS.

Keywords: Visualization, Comparative visualization, Time-series, Web-based visualization

Resumen

EudypTS es una aplicación web que permite la comparación de dos conjuntos de datos de series temporales utilizando una técnica basada en Dynamic-Time-Warping desarrollada por investigadores de la Universidad Nacional del Sur en Argentina. EudypTS ofrece al usuario posibilidades de interacción como la edición de los conjuntos de datos, selección de una medida de distancia y controles de visualización interactivos. También aprovecha web-workers y memorización para un eficiente procesamiento y renderizado de los resultados de la comparación. El *paper* a continuación presenta un resumen de la técnica de visualización base y un detalle acerca del diseño y desarrollo de EudypTS

Palabras claves: Visualización, Visualización comparativa, Series temporales, Visualización basada en web

1. Introduction

Time-series is the name given to an ordered collection of observations made sequentially at often uniform time intervals [1]. The comparison of these time-series is a fundamental task in many fields, such as finance, meteorology, medicine, and astrophysics [2]. Visualization tools, traditionally focused on individual data objects, may offer support for these comparison tasks [3]. By generating visual representations of data, visualizations exploit the human perceptual system to aid in seeing and understanding otherwise abstract data [4, 5].

For the specific case of meteorological data, researchers at the Universidad Nacional del Sur in Argentina proposed a visualization technique [6] to assist in the comparison of time-series. The technique proposed by researchers Larrea and Urribarri aims to visualize how a time-series should be transformed in order to match another one by using Dynamic-Time-Warping (DTW) [7, 8]. First, an optimal mapping between the series is calculated using DTW. Then, a warping function is induced from this optimal mapping. The warping function allows the transformation of one of these time-series to obtain two time-aligned sequences. An overview+detailed visual analysis tool is the result of these calculations. The overview visualization shows a quantitative summary of the comparison between each pair of sequences. And the detailed view shows the specifics of the DTW-based

comparison between two specific time-series [6].

In this paper, we present a client-side web application that allows experts in the meteorological and other fields to upload two time-series datasets and explore them freely using this technique.

This work continues with an explanation of the proposed comparison technique. Section 3 will present related web-based visualization tools with similar goals. A functional description of the web-application is given in section 4, along with an overview of key design and architectural decisions. Section 5 will contain specifics about the implementation of EudypTS. And finally, section 6 will provide conclusions and intended future work.

2. Proposed comparison technique

As mentioned, the proposed comparison technique is based on Dynamic-Time-Warping (DTW). DTW is used to find an optimal mapping between a *reference* time-series and a *target* time-series. A warping function is then induced by this optimal mapping. The warping function allows the transformation of one of the sequences to be time-aligned with the other. Out of this warping function, a misalignment function and a degree-of-misalignment function for these time-series are calculated in order to quantify the misalignment between them. Then, an *overview+detail* visualization is generated.
The following is a detailed explanation of this comparison.

2.1 Optimal mapping using DTW

2.1.1. Accumulated-distance matrix

Given a *reference* time-series R and a *target* time-series S of length L and N respectively, an accumulated distance matrix is first computed as follows (see Algorithm 1).

Algorithm 1 Accumulated distance matrix M between time-series R and S

Input: Two time-series R and S of length L and N respectively
Output: The accumulated distance matrix M between the two time-series

// Distance between the initial records
1. $M_{1,1} \leftarrow dist(R_1, S_1)$

// Accumulated distance between S_1 and all R's records
2. **for all** $i \in [2, L]$ **do**
3. $\quad M_{i,1} \leftarrow dist(R_i, S_1) + M_{i-1,1}$

// Accumulated distance between R1 and all S's records
4. **for all** $j \in [2, N]$ **do**
5. $\quad M_{1,j} \leftarrow dist(R_1, S_j) + M_{1,j-1}$

// Accumulated distance between every other pair of records
6. **for all** pair $i, j \in [2, L] \times [2, N]$ **do**
7. $\quad M_{i,j} \leftarrow dist(R_i, S_j) + \min\{M_{i-1,j-1}, M_{i-1,j}, M_{i,j-1}\}$

Any available distance function can be used for calculating the distance between points in the time-series [9]. For instance, the Euclidean distance, the Manhattan distance, among other options [10].

2.1.2. Minimal distance path

A minimal distance path from (1,1) to (L,N) is then calculated in the following way (see Algorithm 2).

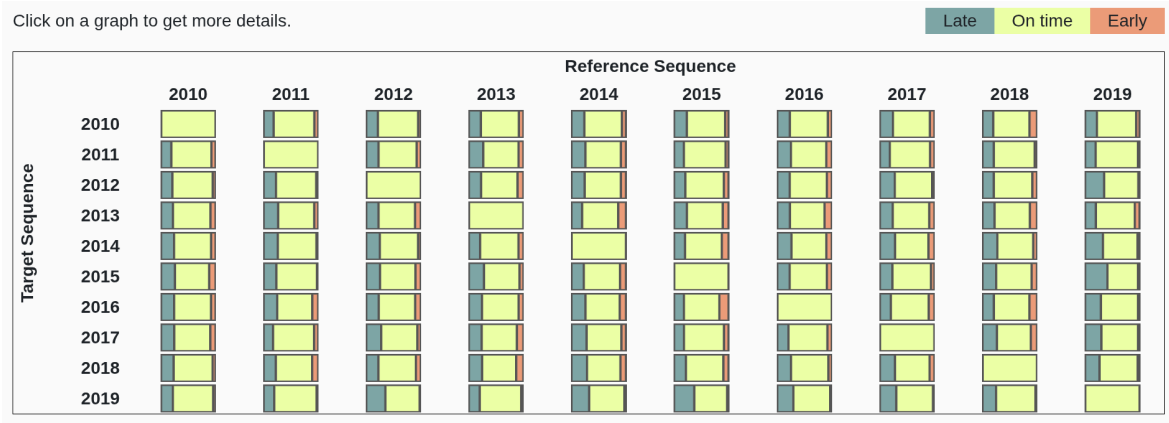

Fig. 1 *Overview* summary-boxes visualization

Algorithm 2 Warping path P between sequences R and S

Input: Accumulated-distance matrix M of $L \times N$ elements

Output: Warping path P between the two sequences

1. P starts empty
2. $i \leftarrow L, j \leftarrow N$
3. **while** $i \geq 1$ and $j \geq 1$ **do**
4. Add (i, j) to the path P
5. Let m be $\min\{M_{i-1,j-1}, M_{i-1,j}, M_{i,j-1}\}$
6.

$$(i,j) \leftarrow \begin{cases} (i, j-1) & \text{if } i = 1, \\ (i-1, j) & \text{if } j = 1, \\ (i, j-1) & \text{if } M_{i,j-1} = m, \\ (i-1, j) & \text{if } M_{i-1,j} = m, \\ (i-1, j-1) & \text{otherwise.} \end{cases}$$

2.2. Warping function

Given the computed minimal distance path P, a warping function F : [1, L] → [1, N] is induced. This function is defined as follows

$$F(n) = \lfloor \text{mean}_{(n,j) \in P}\{j\} \rfloor \quad (1)$$

For an observation of index n in the reference time-series R, the function F returns the index of the best matching observation in the target time-series S. For all pairs (n,j) in P (that is, having the index n as their first element), the index of the best-match is the integer truncation of the mean of all second elements j.

2.3. Misalignment function

For each observation with index n in R, the misalignment function G computes the distance between an index n in R to the index of the best-matching observation in S. G is defined as follows

$$G(n) = F(n) - n \quad (2)$$

If the slope of this misalignment function is negative, we can say that S is delayed with respect to R; if the slope is positive, we can say that S is early with respect to R; and if the slope is zero, we can say that S is on-time.

For the purpose of this work we use a non-smoothed version of the misalignment function; for the smoothed refer to [6].

2.4. Degree-of-misalignment function

To quantify *how* misaligned a record is, a degree of misalignment function H(n) can be defined as follows

$$H(n) = \begin{cases} \frac{d\tilde{G}(n)}{\max(d\tilde{G})} & \text{if } d\tilde{G}(n) > 0, \\ 0 & \text{if } d\tilde{G}(n) = 0, \\ \frac{d\tilde{G}(n)}{|\min(d\tilde{G})|} & \text{if } d\tilde{G}(n) < 0. \end{cases} \quad (3)$$

Where dG is the discrete derivative of the misalignment function G.

H(n) ranges from -1 to 1 with negative values indicating that a record is delayed, and positive values indicating that a record is early. The

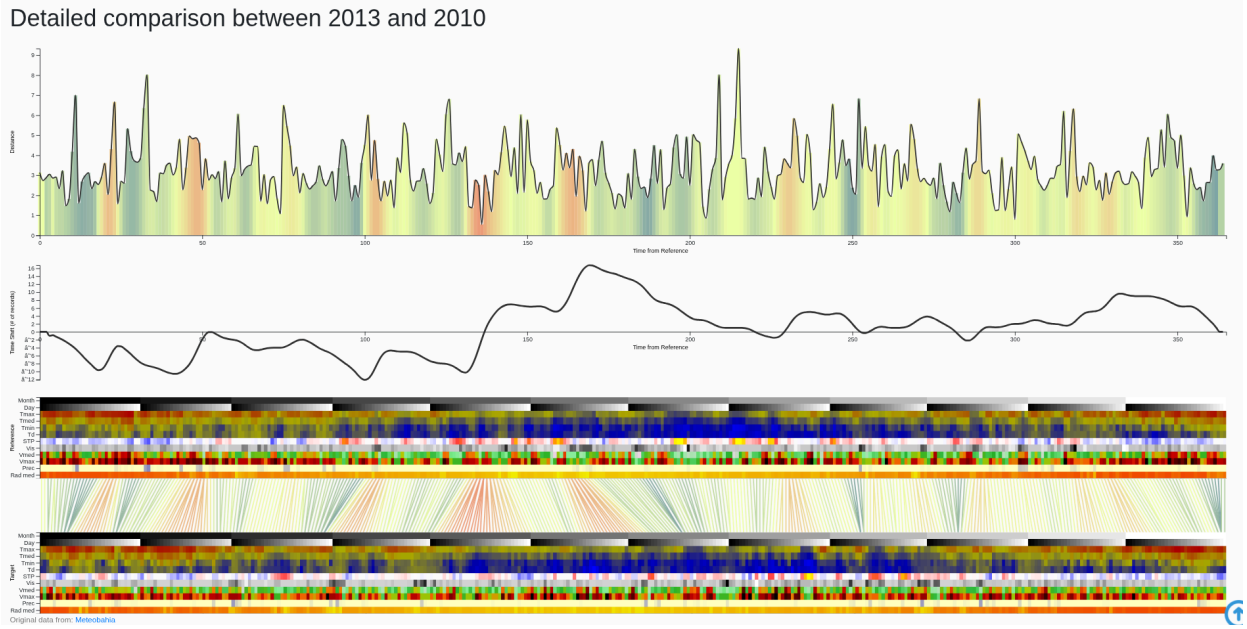

Fig. 2 *Detailed* view visualization

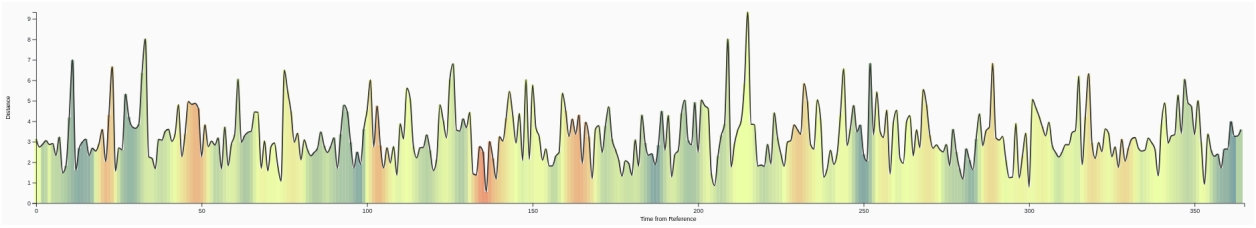

Fig. 3 Distance and degree-of-misalignment visualization

magnitude |H(*n*)| indicates *how* misaligned a record is. A zero value for H(*n*) means that the record is on-time.

2.4. Resulting visualization

An *overview+detail* visualization is generated as a result of this comparison technique. The *overview* visualization consists of all the pairwise summary-boxes for each one of the two-time-series comparisons (see Fig. 1). These summary-boxes show *how* misaligned the records are with the help of a discrete color scale. The *detailed view*

incorporates three charts (see Fig. 2). First, a graph displays the distance from each point in the reference dataset R to its best-match in the target dataset S. The height of the curve represents the distance and the color under it displays the degree-of-misalignment according to a color scale (see Fig. 3). Secondly, a line chart is used to visualize the misalignment function G (see Fig. 4). Finally, a parallel heatmap visualization shows both time series and a parallel time-relationship visualization to visualize the time- warping (see Fig. 5). The accompanying figures belong to the implementation of the comparison technique on meteorological datasets, which is part of the original paper [6].

3. Overview of similar tools

A survey of related academic and commercial visualization solutions has been conducted in order to review current practices on their design, architecture and implementation. Additionally, this overview focuses on those decisions that address problems we anticipate in our own development, for instance, how they handle heterogeneous input data and the processing of operations.

Related or *similar* solutions refer to applications that let an end-user generate visualizations based on a certain specific technique, without requiring expert programming knowledge. In this section two solutions of this kind will be reviewed: *Adaguc-server*, for visualizing meteorological data; and *SPOT*, developed for scientific visualization of high-dimensional data.

3.1. Adaguc-server

Adaguc-server is a geographical information system dedicated to visualize meteorological, climatological and atmospheric data according to the Web Map Service and Web Coverage Service specifications from the Open Geospatial Consortium [11].

3.1.1. Data input

Adaguc supports multiple data formats, such as CSV, JSON, GeoJSON and other formats specific to the meteorological field. The system supports both static and streamed data, which must be loaded as files into specific directories in the file-system called `ADAGUC-autowms`, `ADAGUC-datasets` and `ADAGUC-data`, each of which serve specific purposes. Then, the new files need to be indexed into a database by running a script called `updatedatasets`.

3.1.2. Handling of heterogeneous datasets

In the meteorological domain, datasets can be heterogeneous [11]. To address this problem, all datasets are converted into an internal data model called the ADAGUC common data model (ACDM)

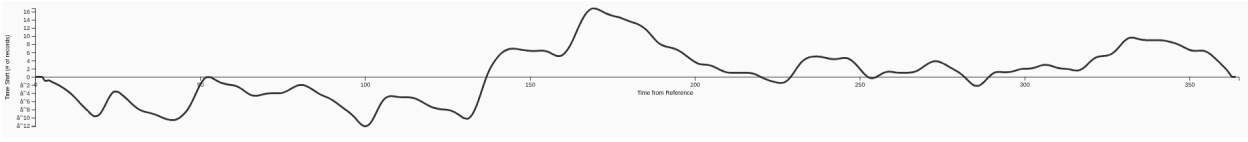

Fig. 4 Line-chart displaying the misalignment function

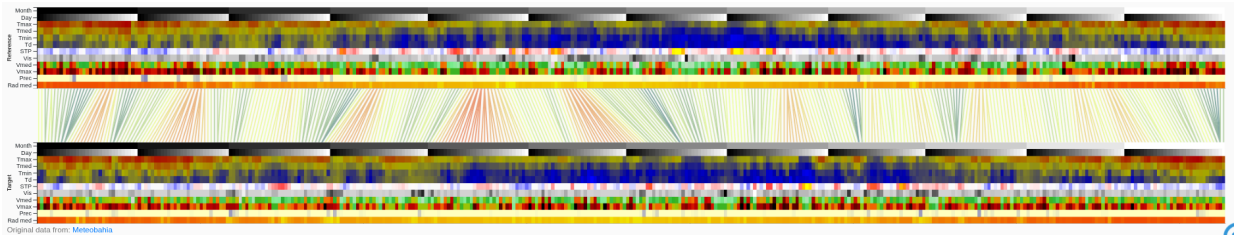

Fig. 5 Parallel heatmaps and parallel time-relationship visualization displaying both time-series

right after indexing. Multiple adapters are used for converting the many specific data models into ACDM. With this initial conversion, downstream software components can consume the data without having to support multiple specific data formats.

3.1.3. Architecture

Adaguc-server is a visualization engine written in C++ and provides services to client visualization applications that support Web Map Service (WMS) and Web Coverage Service (WCS) standards. Client applications send their request to an Adaguc-server instance through the Common Gateway Interface (CGI) protocol. This protocol allows an HTTP server to handle requests that involve the execution

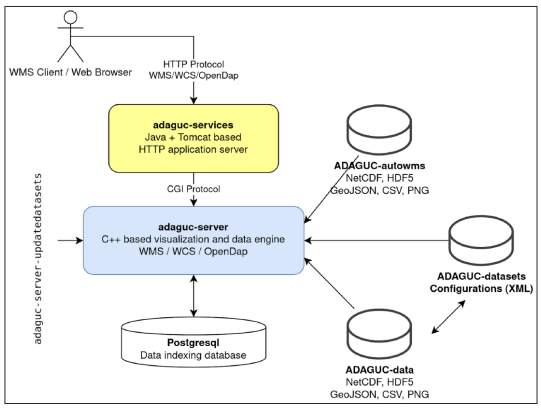

Fig. 6 Architecture of *Adaguc-server* [Plieger2021]

of an external program or script [12]. It is worth mentioning that Adaguc-server does not implement an application server directly out-of-the-box. Extra tooling is therefore needed when setting Adaguc-server up. See Fig. 6 for a diagram of Adaguc's architecture

3.1.4. Parallel request processing

For each request made to the server application, a new process of the Adaguc-server engine is created in order to process it. The application server allows for the parallel execution of as many processes as desired. After completion of each request, the associated process terminates and resources are automatically freed. Optimizations have been put in place, such as the termination of processes that take more than a certain time limit or the deallocation of resources that were not properly released.

3.2. SPOT

SPOT is an open-source web-based data visualization tool. Its main goal is to provide an interface for the exploration of high-dimensional data-sets to users without extensive programming expertise [13, 14].

3.2.1. Architecture

Three components make up *SPOT*: SPOT-Framework, which provides the classes and operations necessary for working with data inside the application; a frontend web-application that serves as the main interface for user-interaction; and SPOT-Server, that processes requests for operations on the data.

3.2.2. Internal data representation

As mentioned, SPOT-Framework provides classes and operations for working with data. Data is

Fig. 7 Client-side-only workflow for *SPOT* [Diblen2021]

represented as a `data-set`. `Data-sets` have items, or rows, each with one or more `facet` instances.

Facets represent a property that an item can have. A selection of facets make up a *filter*. One or more *filters* make up a *data-view*. Users interact

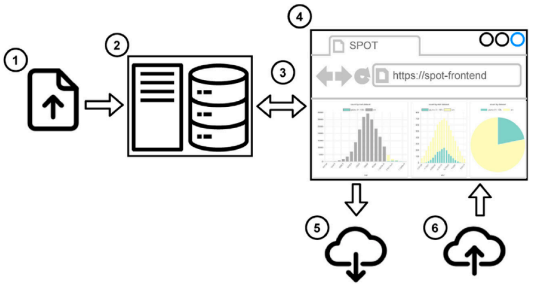

Fig. 8 Second workflow for *SPOT* with a remote database [Diblen2021]

with the data by setting ranges or selections on the different facets to create filters, and by adding or removing filters from *data-views*.

3.2.3. On-demand client- and server-side processing

The *SPOT-Server* component offers two different implementations, each enabling a distinct workflow. In the first workflow, the user can upload local datasets directly to the frontend web-application component. From then on, data operations are run directly in the web-browser where the web-application is running. This approach needs no internet connection. The second approach requires the user to upload the datasets to the *SPOT-Server* component. The datasets are then stored in a PostgreSQL database. Requests made by the user through the frontend component are sent as SQL queries to the database. This approach makes use of indices and parallelism for performance optimization. Downloading the generated visualizations is possible on both workflows. See Figures 7 and 8 for a diagram of both workflows.

4. Solution Design

4.1. Functional description of our solution

Our aim is to develop a web-application that allows experts to upload a *reference* and a *target* time-series dataset and compare them using the visualization technique described in Section 2. The user should be able to select the distance measure most appropriate for their use-case. For multivariate datasets, the user should also be able to select which attributes to compare. Once the visualization is generated, user interaction capabilities are also desired. More functional requirements will be explored throughout this section.

4.2. The *pipeline* model as a starting point

Pipeline models are common in the information visualization field, since they easily describe transformations on the data from its raw state to the rendered visualizations [15]. More specifically, the Unified Visual Analytics Model (UVAM) was adopted [16].

4.2.1. The UVAM Pipeline

This model consists of five data-oriented states. Data transformation operations move data from one step to the next. Nevertheless, different operations can be run on each stage without leaving it (see Fig. 9). We will now describe each component of the UVAM model while simultaneously adapting it to the design of our solution (see Fig. 10 for the final pipeline).

Raw-Data. This is the initial state of the visualization process. In our case, these are the datasets uploaded by the user. Supported formats will be discussed in the implementation section.

Operations on raw-data. These are often data pre-processing tasks. In our case, no operations will be carried out on the raw datasets other than file validity checks.

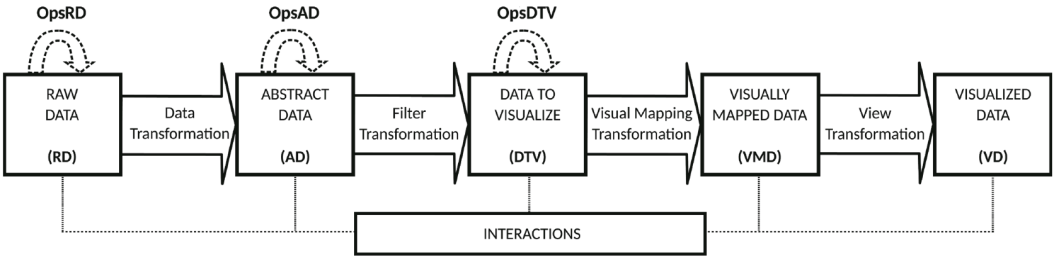

Fig. 9 Pipeline for the Unified Visual Analytics Model

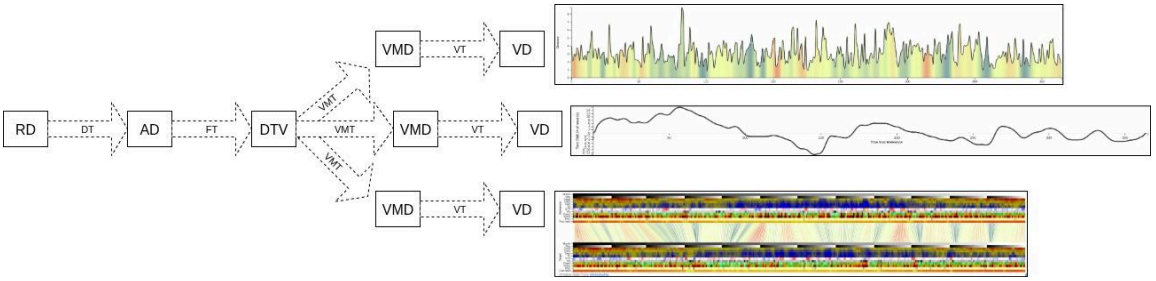

Fig. 10 Pipeline for our solution

Data-Transformation. This step often involves transforming the raw-data into a format that is manageable throughout the rest of the *pipeline*. In our solution, such as in the ones discussed in the previous section, the raw datasets will be parsed into a suitable internal data format.

Abstract Data (AD). At this stage, both datasets will be in a table format with rows and columns. They will not be validated as time series, yet.

Operations on Abstract data. These operations often involve data enhancement and analysis tasks. In our case, the user will be able to edit the data in the tables in order to meet the data-quality requirements of the comparison technique. That is, the user will be responsible for transforming each dataset into a valid time-series for comparison. The user will trigger the execution of the comparison. After successful completion, the result of the comparison is added to the abstract data.

Filter Transformation. In this step, the specific data to be visualized is selected through projection and filtering.

Data-to-Visualize (DTV). It consists of both the *reference* and *target* time-series and the corresponding comparison results.

Operations on Data to Visualize. These are operations on the data that will be present in the visualization. In our case, the user will be able to select the appropriate color scale for each time-series variable, according to their use-case.

Visual Mapping Transformation. In this step, data attributes are mapped to visual variables. Since the comparison technique presented in Section 2 makes use of three charts, three different visual mappings will be applied on the DTV.

Visually Mapped Data (VMD). It consists of the DTV enriched with a visual structure. In our case, one set of VMD will be available for each of the three visualizations that make up the comparison technique from Section 2.

View Transformation. This transformation consists of rendering the final visual representations on screen. This transformation will be delegated to an external library. However, details on this specific step will be discussed in the implementation section.

Visualized Data (VD). It constitutes the exploration space for the user. In our case, it is composed of each of the visualizations shown in Fig. 3, 4, and 5.

4.3. Client-only architecture

Dynamic-Time-Warping (DTW) is time- and memory-consuming [6]. This fact raised the question of whether the DTW-based comparison should be run directly in the web-browser where EudypTS will run, or in a separate server with dedicated resources.

Similar solutions like the ones discussed in the previous section have a client-server architecture, with most data-processing tasks running in the server. However, a client-only architecture was picked for our solution. Such an architecture eases installation and use by experts of fields other than computer-science. This decision also focuses development efforts on the correct implementation of the time-series comparison technique and on optimizing user interaction and experience.

5. Implementation

5.1. Main technology stack

This web application was developed with React, a component-based JavaScript library for developing interactive user-interfaces for the web. Reasons for this choice were its large developer community and previous use from our part, which eased the learning curve.

Typescript, a strongly-typed programming language based on Javascript was picked for this project for better code reliability and maintainability. Typescript is, of course, supported by React [17].

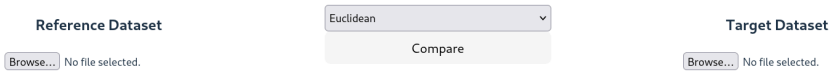

Fig. 11 Initial screen of our solution

When developing React- and Typescript-based applications, a build tool is strictly necessary. Firstly, all Typescript code must be transpiled into JavaScript code. And React-specific files and code structure must be transformed into browser-readable Javascript, HTML and CSS files. Vite was the build-tool chosen for this project due to its minimal configuration needs. Other external libraries are part of EudypTS and will be explored throughout this chapter.

We will now follow the expected user workflow in order to describe the architecture of our solution.

5.2. Dataset upload and editing

The main user-interface component is the `App` component. Figure 11 shows a screenshot of the initial screen. It shows a button in the middle that allows the user to run the comparison, and two panels to the left and right. These two panels are instances of the `DatasetEditor` component and allow the user to upload their datasets as CSV files by clicking the file-upload button.

Once the user has selected the dataset files from their file-system, these are parsed using the external library PapaParse and then formatted into the internal `TableData` type, which consists of a *headers* array containing column title information; and a *data* array, containing the dataset's rows.

Datasets are stored in the *reference* and *target* state variables in the `useTSCompare` custom-hook (we will discuss this custom-hook later in this section).

As stated in the previous chapter, there are no initial restrictions on the raw datasets to be uploaded by the user other than to be valid CSV files. It is a responsibility of the user to bring them into a valid time-series format. For this purpose, the user is able to freely edit the datasets through a data table (see Fig. 12) provided by the external library Handsontable. This data table is encapsulated in the `DataTable` React-component.

The user may run the comparison at any time by clicking the Compare button in the center of the screen (see Fig. 11).

5.3 Comparison Logic

During development, the problem of comparing two generic time-series was tackled first. This logic was then adapted to the comparison of the two `TableData` instances being handled by the user through the UI.

5.3.1. Comparison of time-series

Let us first clarify what we mean by *time-series* in our implementation. A *time-series* is an ordered array of numerical arrays. Each internal numerical array is considered an observation or measurement made at a certain point in time. All the observation points that make up a time-series must have the same number of variables, that is, all numerical arrays of the time-series must have the same length. It is important to notice that at the time of comparison, both time-series must have the same number of variables. The numerical array at index 0 is considered the first observation in the time-series.

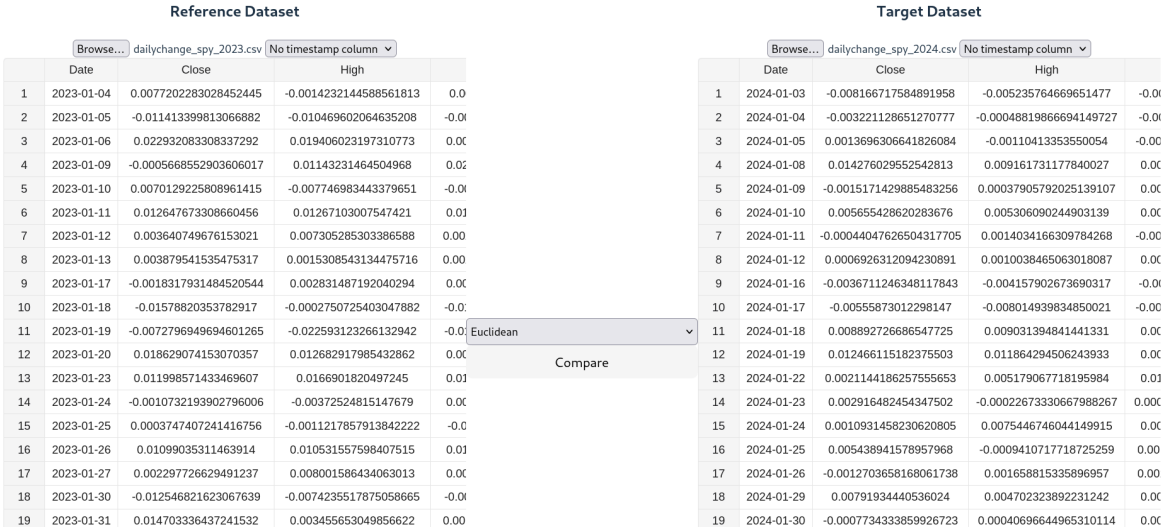

Fig. 12 Data-tables for dataset editing

The one at the last index is considered the last observation. A numerical sub-array at an index *n* (that is not the first or last) models an observation that succeeds the observation at index *n-1* and precedes the observation at index *n+1*.

A `TSComparator` interface was defined for the comparison of time-series. This interface exposes the `compare` method that takes a *reference* and *target* time-series as arguments and returns an object of the type `ComparisonResult`. This result describes, for each element of the *reference* time-series, its *best-match* or *warping*, the *distance* to its best-match, the value of the *misalignment* function for this position, and the *degree of misalignment*.

As described in the first section, any available distance function could be used for this comparison technique. For that purpose, the `TSComparator` interface is first implemented by an abstract class called `AbstractTSCompare`. This abstract class contains a general implementation of the comparison between two time-series but leaves the distance calculation between two time-series observations as an unimplemented function. Distance-measure-specific `TSComparator` classes implement this function. During our development, specific `TSComparator` classes have been implemented for the Euclidean, Manhattan, and Karl-Pearson distance measures.

5.3.2 `TableData` comparison

The adapter design-pattern [18] was used for making the `TSComparator` interface available for comparing two `TableData` instances.

The `ITableDataComparator` adapter interface defines a compare method that accepts both a *reference* and *target* `TableData` instances as arguments and returns an `AdaptedResult` object. The `AdaptedResult` includes a status property to indicate whether the comparison was successful. If the comparison fails, the result will contain an error message. Otherwise, it will hold a `ComparisonResult` object. This approach makes error-handling in the frontend easier.

The `TableDataComparator` class that implements the aforementioned interface is tasked with converting the `TableData` instances into valid time-series and delegates the comparison to a `TSComparator` object to be known at runtime called *adaptee*, following the adapter design pattern (see Fig. 13).

As stated in this chapter, no restrictions are imposed on the raw datasets uploaded by the user. It is also not assumed that the `TableData` instances being handled by the user are, at any moment, valid time-series. This validity must be therefore checked before comparison. A static class called `TimeSeriesValidator` has been implemented for this purpose.

To ensure the correct ordering of the `TableData` rows in the resulting adapted time-series, the `TableDataComparator` implementation class exposes two methods for setting a timestamp column. That is, a column that contains a valid timestamp that will give an order to the resulting adapted time-series. Valid timestamps are those parseable by the `Date` static class provided natively by JavaScript. If no timestamp column is

determined, the order of the *TableData* rows will be kept.

5.4 Running the comparison

5.4.1. The *useTSCompare* custom-hook

The interaction with the *TableDataComparator* and *TSComparator* classes is encapsulated in the *useTSCompare* custom-hook.

Custom-hooks are separate React components that do not have any user-interface. They allow for the encapsulation of React logic and are recommended for interacting with external components or libraries, such as our comparator and adapter classes. [19].

The *reference* and *target* *TableData* instances are kept in the *useTSCompare* custom-hook as state variables and are exposed to the main UI component (that is, the *App* component). This custom-hook also exposes methods for setting the timestamp column of the *TableData* instances; for the selection of an available distance measure; and for running the comparison.

5.4.2. Web Workers for parallel processing

As stated in section 4, this comparison technique is based on Dynamic-Time-Warping and might therefore be time- and memory- consuming. Since Javascript code runs natively on a single thread, running the comparison directly on this thread might block the UI and provide the user with a bad experience [20].

Web workers allow for the execution of Javascript code in separate threads. By running computationally intensive operations on separate web-worker threads, the main thread remains free to handle UI work, leaving the user interface fully responsive. In this spirit, web-workers are used for running the comparison. The factory design pattern is used for instantiating web-workers [21]. There is a factory for each comparator implemented, that is, for Euclidean, Manhattan and Karl-Pearson comparators. The Comlink library is used for creating and interacting with web-workers.

5.4.3 Optimizations

Some optimizations were put in place for the comparison calculations. For instance, the comparison result is memoized and will not be re-calculated unless the user changes the reference or target dataset, or selects another distance measure or timestamp column for any dataset. Web workers are also instantiated only once after being called. Once a web-worker for a certain distance measure has been instantiated, it is re-used for later comparisons made with the same distance measure.

5.5 Visualizing the result

The ECharts library was used for crafting and displaying the visualizations, along with the E-Charts-for-React library for easier use with our React-based implementation.

The three charts described in the previous sections were implemented in the *charts* sub-directory inside the *components* directory. The *EDetailedView* component contains the three

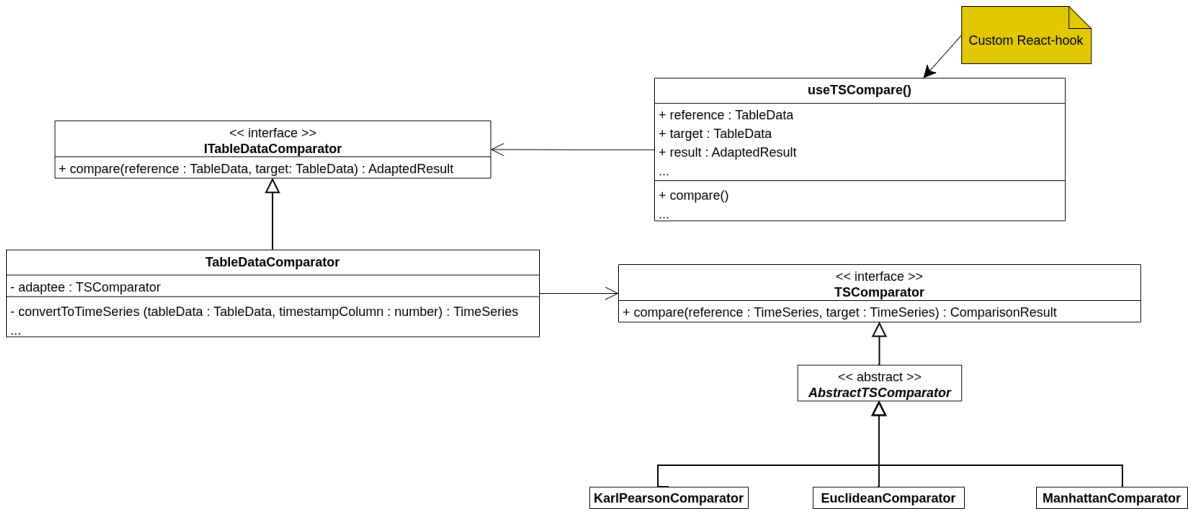

Fig. 13 Adapter-pattern for the comparison of *TableData* objects

charts, which are in turn implemented in the `EDistanceGraph`, `EMisalignmentGraph` and `HeatmapParallelCoord` React components. In the specific case of the Parallel Heatmaps visualization, the user can select a color-scale for each variable of each time-series and set minimum or maximum values for each of these color-scales. By default, the minimum and maximum values are the minimum and maximum values for that variable in the original time-series.

5.5. Implementation notes

The codebase for the solution presented can be accessed in the FacundoAlvarado9/EudypTS GitHub repository. A version of the React application is currently deployed and available at https://time-series-comparison.vercel.app/. A user-guide is at hand in the GitHub repository.

6. Conclusions and future work

6.1. Concluding remarks

This work provides a web-based general implementation of the visualization technique described in Section 2. It allows experts in meteorology and other fields to upload two datasets with minimal restrictions and compare them visually.

Generalizing the implementation and making it friendly to experts and data from fields other than meteorology and computer-science has been a point of focus. To achieve this, the application was designed to require minimal configuration and few restrictions are imposed on the input datasets. User-interaction capabilities were also put in place to this end, for instance, by allowing the user to select the color-scales most suitable to their use case in the parallel-heatmaps visualization. -

6.2. Limitations and future work

There are, of course, limitations to this project. For instance, when it comes to data formats supported, CSV is still the only one. With regard to performance, its client-only architecture limits the available computing power to that of the user's local machine. As regards the comparison technique, simplifications have been made, for example, by not applying gaussian filters to the *misalignment function* [6]. These limitations serve as opportunities for future work. For instance:

- Adding support to other data formats such as JSON, or other formats used in specific fields.
- Allowing for the potentially time- and memory-consuming computations to run in a separate environment. The *SPOT* application explored in section 3 showcases such a feature by allowing data-intensive operations to run directly in a remote database server.
- Adding support for ordering time-series with other timestamp formats, such as multi-column timestamps. Time-series datasets can be found where, for instance, the year, month, day, and hour are in separate columns. So far, our application supports the ordering of a single-column timestamp. -

Improvements like these will help further generalize the implementation of this solution, by allowing a wider variety of dataset formats and lengths to be supported while keeping performance.

Author's contribution

The authors confirm contribution to the paper as follows: FA: Software, Original-draft writing; DU: Supervision, Writing-Reviewing and Editing, Validation; ML: Supervision, Writing-Reviewing and Editing, Validation.

Competing Interests

The authors have declared that no competing interests exist.

References

[1] J. Y. Lee, R. Elmasri and J. Won, "An integrated temporal data model incorporating time series concept", *Data & Knowledge Engineering,* vol. 24, pp. 257-256, 1998, doi: 10.1016/s0169-023x(97)00034-7.

[2] B. D. Fulcher, M. A. Little and N. S. Jones, "Highly comparative time-series analysis: the empirical structure of time series and their methods", *Journal of the Royal Society Interface*, vol. 10, no. 83, 2013, Art. no. 20130048, doi: 10.1098/rsif.2013.0048.

[3] M. Gleicher et al., "Visual comparison for information visualization" *Information Visualization*, vol. 10, no. 4, pp. 289-309, 2011, doi:

10.1177/1473871611416549.

[4] W. Aigner, S. Miksch, H. Schumann and C. Tominski, “Time & Time-Oriented Data”, in *Introduction*, 2nd ed. London, United Kingdom: Springer-Verlag, 2011, ch. 1, pp. 2.

[5] C. Tominski and H. Schumann, “Interactive Visual Data Analysis”, in *Introduction*, 1st ed. Boca Raton, FL, USA: CRC Press, 2020, ch. 1, pp. 3.

[6] M. L. Larrea and D. K. Urribarri, “A visualization technique to assist in the comparison of large meteorological datasets”, *Computers & Graphics*, vol. 104, pp. 1-10, 2022, doi: 10.1016/j.cag.2022.02.011.

[7] D. J. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series” in *Proc. 3rd Intl. Conf. on Knowledge Discovery and Data Mining (AAAIWS’94)*, pp. 359-370, 1994, doi: 10.5555/3000850.3000887.

[8] L. Rabiner, B. Juang, *Fundamentals of speech recognition.* USA: Prentice-Hall, 1993.

[9] M. L. Larrea and D. K. Urribarri, "Visualization technique for comparison of time-based large data sets," in *Cloud Computing, Big Data & Emerging Topics. JCC-BD&ET 2021*, M. Naiouf, E. Rucci, F. Chichizola, and L. De Giusti, Eds., Cham, Switzerland: Springer, 2021, pp. 179–187. doi: 10.1007/978-3-030-84825-5_13.

[10] S. H. Cha, “Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions”, in *Int. J. Math. Model. Meth. Appl. Sci.* vol. 1, pp. 300-307, 2007

[11] M. Plieger and E. De Vreede, “Adaguc-Server: Interactive Access to Heterogeneous Meteorological and Climatological Datasets Using Open Standards”, *Journal of Open Research Software*, vol. 9, no. 1, p. 33, 2021, doi: 10.5334/jors.382.

[12] D. Robinson and K. Coar, “The Common Gateway Interface (CGI) Version 1.1”, RFC Editor, RFC 3875. Accessed: Aug. 29, 2025. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3875

[13] Diblen et al. “SPOT: Open Source framework for scientific data repository and interactive visualization”, *SoftwareX*, vol. 9, pp. 328-331, Apr. 2019, doi: 10.1016/j.softx.2019.04.006.

[14] Diblen et al. “Interactive Web-Based Visualization of Multidimensional Physical and Astronomical Data”, *Front. Big Data*, vol. 4, June 2021, Art. no. 626998, doi: 10.3389/fdata.2021.626998.

[15] X. M. Wang et al., “A Survey of Visual Analytic Pipelines”, *J. Comput. Sci. Technol,* vol. 31, no. 4, pp. 787-804, 2016, doi: 10.1007/s11390-016-1663-1

[16] M. L. Ganuza et al., “UVAM: The Unified Visual Analytics Model. The Unified Visualization Model Revisited” in *JCC-BD&ET.,* M. Naiouf, E. Rucci, F. Chichizola, L. De Fiusti, Eds. Aug. 2023, pp. 189-203, doi: 10.1007/978-3-031-40942-4_14

[17] R. Nabors et al., "TypeScript". React Docs. Accessed: Sep 21, 2025 [Online]. Available: https://react.dev/learn/typescript.

[18] E. Gamma et al., “Structural Patterns”, in *Design Patterns: Elements of Reusable User-Oriented Software*. U.S.A.: Addison-Wesley, 1994, ch. 4, pp. 137-150

[19] R. Nabors et al., “Reusing Logic with Custom Hooks”. React Docs. Accessed: Sep. 21, 2025. [Online]. Available: https://react.dev/learn/reusing-logic-with-custom-hooks.

[20] J. Walker et al., “Main Thread”. MDN Web Docs. Accessed: Sep 21, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Main_thread

[21] E. Gamma et al., “Creational Patterns”, in *Design Patterns: Elements of Reusable User-Oriented Software*. U.S.A: Addison-Wesley, 1994, ch. 4, pp. 87-96