DendroTime: Progressive Hierarchical Clustering for Variable-Length Time Series

Sebastian Schmidl
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
sebastian.schmidl@hpi.de

Ferdinand Rewicki
German Aerospace Center (DLR), Friedrich Schiller University Jena
Jena, Germany
ferdinand.rewicki@dlr.de

Felix Naumann
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
felix.naumann@hpi.de

Thorsten Papenbrock
Philipps University of Marburg
Marburg, Germany
papenbrock@informatik.uni-marburg.de

Abstract

Many effective dissimilarity measures for variable-length time series, such as DTW, MSM, or TWED, are expensive to compute because their runtimes increase quadratically with the time series' lengths. When used in hierarchical agglomerative clustering algorithms that need to compute all pairwise time series dissimilarities, they cause slow runtimes and do not scale to large time series collections. However, there are use cases, where fast, interactive hierarchical clustering is necessary. For these use cases, progressive hierarchical clustering algorithms can improve runtimes and interactivity. Progressive algorithms are incremental algorithms that produce and continuously improve an approximate solution, which eventually converges to the exact solution.

In this paper, we present DendroTime, the first (parallel) progressive clustering system for variable-length time series collections. The system incrementally computes the pairwise dissimilarities between the input time series and supports different ordering strategies to achieve progressivity. Our evaluation demonstrates that DendroTime's progressive strategies are very effective for clustering scenarios with expensive time series dissimilarity computations.

Keywords

Hierarchical Clustering, Time Series, Progressive Algorithm, Dendrogram, EDEN ISS, Pattern Mining, Classification

1 Clustering time series

Time series clustering is an active and popular research area with hundreds of publications per year [1, 21, 22, 24]. The objective is to group similar time series, such that the time series within a group are more similar to each other than time series of different groups. Time series clustering is often the starting point for exploratory analysis and, thus, applied in many application domains, such as greenhouse monitoring [51], speech recognition [18], environmental monitoring [9], healthcare [56], and energy management [19]. A clustering can be calculated in different ways, including feature-based (e. g., Time2Feat [8]), partitional sequence-based (e. g., k-Means [22] or k-Shapes [41]), or hierarchical sequence-based (e. g., hierarchical agglomerative clustering (HAC) [1, 24, 41]).

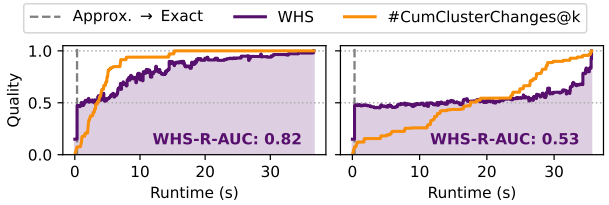


Figure 1: Dendrogram quality (WHS) and convergence indicator (*#CumClusterChanges@k*) of DendroTime's *ada* (progressive) and *fcfs* (non-progressive) strategies for the ACSF1 dataset with *Move-Split-Merge* [57] (MSM) and *average*-linkage. The vertical dashed line marks the switch from the approximate to the exact dissimilarity computation phase.

In this paper, we propose a novel HAC technique because HAC is particularly useful for the (interactive) clustering of time series [1, 14, 24, 41]: It can leverage elastic dissimilarity measures to cluster variable length input time series, it does not require users to specify the number of clusters upfront [1], and it achieves high comparative qualities in benchmarks with elastic dissimilarity measures [24, 61]. However, because HAC has to call a usually costly (elastic) dissimilarity function for all pairs of time series, it scales poorly and works for only small datasets [1].

To deal with the high computational complexity of HAC, researchers have proposed approximate HAC algorithms [12, 25, 26, 61] that compute the dissimilarities for a small selection of time series pairs and infer them for the remaining ones. Because this strategy works only for certain dissimilarity measures and linkage methods, existing approximate HAC algorithms are limited in their effectiveness and applicability. So to control the computational complexity of HAC and integrate it into interactive clustering processes, we propose to calculate a hierarchical clustering progressively.

Progressive algorithms [5, 33, 44, 45, 55, 62, 68] are incremental algorithms that try to achieve better early qualities than traditional incremental algorithms; they produce the same results as exact algorithms when they terminate. Progressive algorithms are different from online or streaming algorithms, in that they process static datasets but intend to gradually improve the result quality. A progressive clustering algorithm should, thus, quickly compute an approximate dendrogram that rapidly converges to the exact dendrogram. It allows scientists to monitor the dendrogram over time and stop the clustering process early, shortening

EDBT '26, Tampere (Finland)
© 2025 Copyright held by the owner/author(s). Published on OpenProceedings.org under ISBN 978-3-89318-103-2, series ISSN 2367-2005. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

the feedback cycle. This is visualized in Figure 1, which shows the result quality (measured using WHS) of a progressive algorithm (left) compared to a non-progressive algorithm (right) over its runtime. Because measuring the result quality requires knowledge of the ground truth dendrogram, which is unavailable in practice, a novel, unsupervised convergence indicator (*#CumClusterChanges@k* in Figure 1) is used to communicate the progress of the algorithm to the user. The progressive clustering approximates the exact result relatively quickly, which allows for early termination, but the non-progressive curve converges only at the very end.

We present DendroTime, a parallel, progressive clustering system that computes a hierarchical clustering for large collections of time series. The algorithm creates and continuously improves an approximate dendrogram, and, thus, an approximate solution of the time series clustering problem. The approximate dendrogram converges to the exact solution when DendroTime finishes. To achieve progressivity, the system sequentially computes the pairwise dissimilarities between the input time series and supports different ordering strategies. DendroTime is independent of the time series dissimilarity measure and the agglomerative linkage method. While some progressive algorithms can provide convergence guarantees [47, 68], heuristics-driven progressive algorithms, such as algorithms for progressive data cleaning [33, 44, 55, 62], progressive data mining [5, 45], and the approach to progressive hierarchical clustering presented in this paper, cannot provide these guarantees. Hence, we experimentally evaluate DendroTime with the most effective dissimilarity measures in their respective categories and four linkage methods. We start the paper with an introduction to time series and hierarchical agglomerative clustering (Section 2). Then, we make the following contributions:

(1) We propose PrAC, the first *progressive HAC algorithm*, which is based on intelligent ordering strategies for dissimilarity computations; and *Area under the WHS-runtime-curve*, a novel convergence measure for HAC algorithms (Section 3).
(2) We develop a *task- and data-parallel execution strategy*, which separates dissimilarity computations from periodic dendrogram constructions, and optimizes the *progressive dissimilarity ordering strategies* for improved scalability (Section 4).
(3) We introduce DendroTime, a *practical clustering system* that combines the parallelized hierarchical clustering algorithm PrAC with the *novel unsupervised convergence indicator #CumClusterChanges@k* to enable interactive, progressive HAC processes (Section 5).
(4) We conduct an *evaluation of PrAC and* DendroTime on 135 time series clustering datasets of various sizes (Section 6) and demonstrate their applicability for time series anomaly detection in the EDEN ISS case study (Section 7).

2 Hierarchical agglomerative clustering (HAC) for time series

DendroTime is a progressive HAC system for time series. It takes a collection of variable-length time series as input and incrementally computes their pairwise dissimilarities. Using the pairwise dissimilarities, DendroTime periodically constructs a dendrogram by iteratively merging the two mutually closest clusters.

Time series. A *time series* is an ordered sequence of real-valued data points, recorded in regular intervals over time. For illustrations and experiments, we focus on univariate time series, in which data points consist of a single variable.

Definition 2.1 (time series cf. [53]). A univariate *time series* $T \in \mathbb{R}^m$ is a sequence of real-valued points $t_i \in \mathbb{R}$, where $1 \leq i \leq m$. The length of the time series T is denoted as $m = |T|$, and the i^{th}-point of the series as $T[i] = t_i$. A *subsequence* $T[i, l]$ of a time series T is a continuous subset of the values in T starting from index i with length l: $\{t_i, t_{i+1}, \ldots, t_{i+l-1}\}$, where $1 \leq i \leq m - l$. Usually $l \ll m$.

The input to DendroTime is a potentially large set of time series with varying lengths:

Definition 2.2 (Dataset). A *dataset* $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a set of n time series T_i with $1 \leq i \leq n$, where $n = |\mathcal{T}|$ is the cardinality of $\mathcal{T}$.

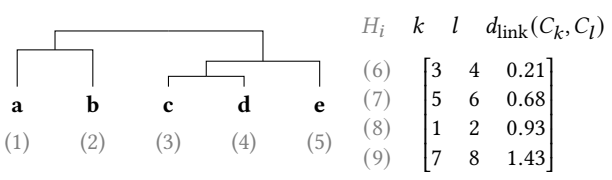
Time series dissimilarity measures. A dissimilarity measure $d(\cdot, \cdot)$ determines a numerical value that describes the distance between two objects (in our case, two time series T_i, T_j). A HAC algorithm uses a dissimilarity matrix of all pairwise time series to find similar and dissimilar time series pairs. Dissimilarity measures for HAC need not be metrics: they do not always satisfy the triangle inequality, and the dissimilarity between two different objects might be zero [20, 37].

Time series dissimilarity measures can be classified into five categories [43]: lock-step, sliding, elastic, kernel, and embedding measures. The most widely used dissimilarity measure, also for HAC algorithms, is the *Euclidean distance* (ED). It is a lock-step measure [43] that compares all points of two time series element-wise. While lock-step measures can technically be applied to time series, they are not well suited for this purpose because (i) they ignore the ordering of the points, (ii) they are sensitive to shifts in time or scaling differences, and (iii) they require truncation, padding, or resampling to deal with unequal-length time series.

Sliding and elastic dissimilarity measures, on the other hand, try to align two time series optimally by comparing one time series with all shifted versions of the other or computing a warping/editing path between the two time series, respectively. This allows these measures to accurately capture dissimilarities in variable-length time series with shifted patterns. However, both sliding and elastic measures are harder to compute than lock-step measures.

Kernel measures use a mapping function to implicitly transform the time series into a higher-dimensional space, in which they compute the time series' dissimilarity. The mapping function can have lock-step, sliding, or elastic properties [43].

To cover the runtime-behaviors of all dissimilarity measure categories, we use the best performing measure per category as representative in our experiments, according to the evaluation of Paparrizos et al. [43]: *Lorentzian distance* (LD) for lock-step, *Shape-based Distance* [41] (SBD) for sliding, *Move-Split-Merge* [57] (MSM) for elastic, and *Dynamic Time Warping Kernel* [34] (KDTW) for kernel measures. We exclude embedding measures because they perform significantly worse than the other measures and require a learning step, which is not applicable in our clustering setting. We also include the popular *Dynamic Time Warping* [6] (DTW) measure. Adding more dissimilarity measures to DendroTime is easy but does not provide further insights because DendroTime's convergence behavior mainly depends on the measures' computational complexity. The



(a) Graphical dendrogram. **(b) Stepwise dendrogram.**

Figure 2: Output of a HAC algorithm for $n = 5$ instances. The implicit cluster identifiers ($k, l, \ldots$) are displayed as (<ID>).

computation of MSM, DTW, and KDTW is expensive, having a time and space complexity in $O(m^2)$. SBD can be computed in $O(m \cdot log(m))$ time, and LD in $O(m)$. We refer the interested reader to Paparrizos et al. [43] and Holder et al. [22] for the detailed definitions and analysis of these and further time series dissimilarity measures. As suggested by Ratanamahatana and Keogh [49] and confirmed by Holder et al. [22], DendroTime uses Sakoe-Chiba bounding with a 5 % window-size for MSM and DTW by default; as recommended by Paparrizos et al. [43], unit-length normalization for LD and z-normalization for KDTW; and no standardization for SBD.

Hierarchical agglomerative clustering. HAC is an unsupervised, well-established machine learning method for constructing hierarchical partitionings of data instances. Agglomerative clustering methods start with singleton clusters (clusters that contain a single instance) and iteratively merge the two mutually closest clusters until all instances of a dataset are part of a single cluster. This sequential procedure creates a hierarchy of cluster merges, which can be depicted as a *dendrogram*. Each level in the hierarchy corresponds to a dataset partitioning. Figure 2a shows a dendrogram for a dataset with $n = 5$ instances.

We denote the set of all (hierarchical) clusters of a HAC algorithm as $C = \{C_1, \ldots, C_{2n-1}\}$. The clusters in C can have different cardinalities, ranging from 1 for singletons to n for the final cluster at the root containing all nodes.

HAC algorithms can use different *linkage methods* to calculate the inter-cluster dissimilarities $d_{\text{link}}(C_k, C_l)$, which determine the order of cluster merges. The most common linkage methods are *single*, *complete*, *average*, *weighted*, *ward*, *centroid*, and *median* linkage [17, 37]. To compute the results for *single* linkage, we use the MST algorithm [52], and to compute the results for *complete*, *average*, and *weighted* linkage, we use the NN-chain algorithm [38], as recommended by Müllner [36]. *Ward*, *centroid*, and *median* linkage are excluded because they assume Euclidean distances [36].

The input to an HAC algorithm are the $\binom{n}{2} = \frac{n(n-1)}{2}$ dissimilarities $d(T_i, T_j)$ for all pairs of time series T_i, T_j in the input dataset $\mathcal{T}$, where $T_i \neq T_j$. Because all hierarchical clustering methods are sensitive to each input value [36], they need to ultimately process all input values and their runtime is, thus, bounded by $\Omega(n^2)$.

The output of a HAC algorithm is a series of cluster merges, which we can parse into a tree-based graphical *dendrogram*. We use the effective *stepwise dendrogram* data model [36] for storing the cluster merges in memory and on disk. A stepwise dendrogram is a compact representation of the merging steps performed by a HAC algorithm. Instead of storing all possible $2n - 1$ clusters of the dataset, it records the cluster merging steps. Figure 2 displays a stepwise dendrogram for $n = 5$ instances with its corresponding graphical representation. In a stepwise dendrogram, the order of merge steps matters and also needs to be chosen for ties, i. e., two merges with the same dissimilarity. For the remainder of the paper, we use the terms stepwise dendrogram and dendrogram interchangeably; for the tree-based representation, we use the term *graphical dendrogram*.

3 Progressive dissimilarity computation

Our goal is the design of a progressive HAC algorithm that can be used for interactive variable-length time series clustering with expensive dissimilarity measures on large collections of time series. For this, we need an algorithm that (i) has a very fast initialization time (time to first approximate result), (ii) continuously updates the approximate result while new dissimilarities become available, and (iii) converges to the exact dendrogram upon completion.

Unlike other papers on improving HAC, e. g., [54, 64], we consider the computation of the pairwise dissimilarities to be part of the clustering algorithm: these computations make up most of the runtime. The actual share of runtime depends on the computational complexity of the used dissimilarity measure, but ranges from 55 % (ED, LD) to 98 % (DTW, MSM, SBD, KDTW).

First, this section introduces our two-phased algorithm PrAC, which continuously updates an approximate dendrogram until it converges to the exact dendrogram (Section 3.1). Subsequently, we discuss different strategies to order the computation of the exact pairwise time series dissimilarities (Section 3.2). Finally, we propose a convergence measure for PrAC that allows us to assess the rate of convergence for different strategies (Section 3.3)

3.1 PrAC: Progressive agglomerative clustering

The main intuition behind progressive clustering is that not all pairwise dissimilarities are equally important for the clustering process. For dense regions, the dissimilarities within the region are crucial and must be accurate: small changes in the dissimilarity values can drastically change the order of cluster merges. However, small changes in dissimilarities between time series that are far apart do not affect the order of cluster merges much. Thus, dissimilarities of close time series need to be exact, while dissimilarities of distant time series can be approximated, and a progressive clustering strategy should compute the exact dissimilarities for close time series before those for distant time series.

PrAC first approximates all pairwise dissimilarities and then sequentially computes the exact dissimilarities, while continuously updating the dendrogram. Using the MST or NN-chain algorithm, PrAC periodically constructs the dendrogram from scratch based on the latest dissimilarity matrix. The periodicity of the dendrogram updates could be defined based on the number of dissimilarity computations, but since every modern machine offers multicore support, PrAC simply runs the dendrogram constructions continuously in parallel (Section 4.1). The dissimilarities between time series are computed in two phases: The first phase computes all approximated time series' dissimilarities; these are needed as hints for any progressive refinement strategy. The second phase computes all exact dissimilarities according to a heuristic ordering strategy that tries to maximize refinement gains. This two-phased approach offers early approximate results

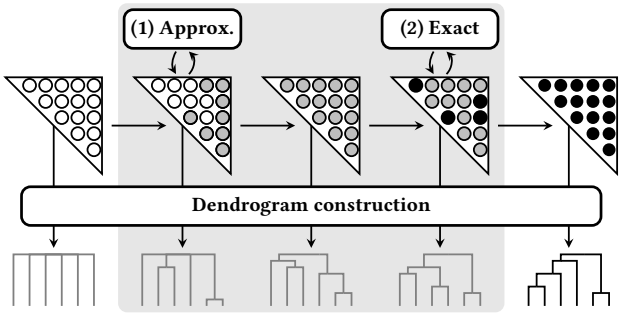


Figure 3: PrAC consisting of two phases: (1) approximated dissimilarity computation and (2) exact dissimilarity computation; ○ unset dissimilarity (= ∞), ◎ approximated dissimilarity $\widetilde{d}(T_i, T_j)$, ● exact dissimilarity $d(T_i, T_j)$.

(approximate dendrograms), gradual and possibly fast result improvements, and guarantees to eventually converge to the exact result. Note that the exact dissimilarity calculations take significantly longer to compute than the approximations, so that PrAC spends most time in the second phase. While PrAC cannot guarantee a monotone or optimal convergence, we show empirically in Section 6.2 that our heuristics provide fast convergence for computationally expensive dissimilarity measures.

Figure 3 provides a visualization of the progressive clustering approach: PrAC starts by loading all time series into memory and initializing all elements of the dissimilarity vector to $+\infty$. The initial dendrogram has all time series in the root cluster.

In the *approximated dissimilarity computation phase* (1), we consecutively estimate the pairwise dissimilarities between time series and periodically construct the dendrogram. To estimate the dissimilarity between two time series, we extract a random small subsequence of length $m_{\text{sub}} = 20$ out of the time series and pass it through our dissimilarity measure d. If a time series T_i is shorter than m_{sub}, we set $m_{\text{sub}} = |T_i|$. The estimated dissimilarity $\widetilde{d}$ between two time series T_i and T_j is calculated as

$$\widetilde{d}(T_i, T_j) = \frac{max\left(|T_i|, |T_j|\right)}{m_{\text{sub}}} \cdot d\left(T_i[c_i, m_{\text{sub}}], T_j[c_j, m_{\text{sub}}]\right)$$
$$\text{where } 0 \leq c_i < |T_i| - m_{\text{sub}} \text{ and } 0 \leq c_j < |T_j| - m_{\text{sub}}$$
$$\text{and } m_{\text{sub}} \leq |T_i| \text{ and } m_{\text{sub}} \leq |T_j|$$

Because the approximation does not consider length differences, we need to extrapolate the approximated dissimilarity $d(T_i[c_i, m_{\text{sub}}], T_j[c_j, m_{\text{sub}}])$, which is based on short random subsequences of length m_{sub} of T_i and T_j. Without the extrapolation, the approximated dissimilarities would have a different value domain than the exact dissimilarities, and, thus, not estimate our desired values well. The chosen extrapolation factor $\frac{max(|T_i|,|T_j|)}{m_{\text{sub}}}$ is efficient to compute and generalizes to all our time series dissimilarity measures. Setting the subsequence length m_{sub} to a small, constant value, i. e. to 20, drastically speeds up the dissimilarity computations ($O(n^2) \ll O(n^2 \cdot m^2)$), but still results in a good and distance-measure-specific estimation quality, as we can observe in Figure 1 (the initial jump is due to the approximated dissimilarities). In our experiments, we use a subsequence from the middle of the time series. In practice, however, we did not observe significant differences in the approximation quality when extracting the subsequence from other positions. While more and more dissimilarities are estimated, the dendrogram takes shape and time series are removed from the root cluster and put into sub-clusters. When all dissimilarities have been estimated, the approximated dendrogram is finished and the algorithm switches into the second phase.

In the *exact dissimilarity computation phase* (2), we incrementally compute the exact pairwise dissimilarities, update the dissimilarities in the dissimilarity vector, and also periodically construct the dendrogram. We use the approximated dissimilarities and the approximate dendrogram to determine a "good" (cf. Section 3.3) order for the computation of the expensive exact dissimilarities, such that the stepwise dendrogram quickly converges to the exact solution. Depending on the dissimilarity computation ordering strategy (cf. Section 3.2), we compute the exact dissimilarities one after the other and update the dissimilarity matrix and the dendrogram. The exact dissimilarities gradually, but not necessarily monotonically, improve the order of cluster merges, whereby the dendrogram progressively approaches the exact solution. When all approximated dissimilarities have been replaced by their exact value, we trigger the dendrogram construction one last time. This ensures that, independent of the periodic dendrogram update schedule, the algorithm terminates with an exact dendrogram.

3.2 Dissimilarity computation ordering

Our progressive HAC algorithm PrAC incrementally computes all approximated dissimilarities and then all exact dissimilarities. The order of the approximated dissimilarity computations is insignificant for the convergence rate of the algorithm because Phase 1 constitutes only a small fraction of the overall runtime. Hence, PrAC simply computes the approximated dissimilarities in the order in which the time series are loaded from disk (cf. *fcfs* strategy below). Because the exact dissimilarity computations are expensive to compute, their order needs to be optimized, such that dissimilarities between close time series are computed before dissimilarities between distant time series.

We now describe six dissimilarity computation ordering strategies with the first two (*fcfs* and *rand*) being baseline strategies.

first-come-first-served. The *fcfs* baseline strategy simply orders the dissimilarity computations in the order we load the time series from disk. When loading time series T_i (for $2 \leq i \leq n$), it generates the pairs (T_j, T_i) for all $1 \leq j < i$. The time series in our datasets are not stored in any specific order.

random. This baseline strategy takes the order of the *fcfs* strategy and shuffles the time series pairs. In our experiments in Section 6, we sample 1,000 random orders to demonstrate that random orders lead to slow convergence.

time-series-length-ascending. The *tsla* strategy sorts the time series pairs in ascending order by their average length. For sliding and elastic dissimilarity measures, computing the dissimilarity between two small time series is over-proportionally faster than between two large time series because the dissimilarity computations scale superlinearly with the time series' lengths. Thus, PrAC maximizes the update rate at the beginning of Phase 2 with decreasing update rates toward the end. This strategy degrades to the *fcfs* strategy if all time series have the same length.

approximate-dissimilarity-ascending. The *ada* strategy defines an ordering heuristic based on the dissimilarities from the approximation phase, from small to large. As motivated earlier, approximation errors for close time series have a larger impact on the clustering than approximation errors for distant time series. Hence, *ada* tries to fix the high-impact errors early, so that

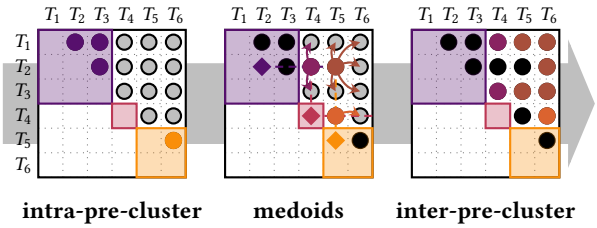


Figure 4: Dissimilarity matrix for the *precl* strategy for a dataset with 6 time series and 3 pre-clusters (◆ medoid, ○ approximate dissimilarity, ● exact dissimilarity, ▢ pre-cluster), updates at each step are highlighted with color.

the estimated dendrogram converges to the exact dendrogram progressively.

pre-clustering. The *precl* strategy, which is visualized in Figure 4, follows a three-step approach inspired by the JET algorithm [61]: (i) The *intra-pre-cluster* step partitions the approximated dendrogram into $\lceil 3\sqrt{n} \rceil$ pre-clusters and computes the exact dissimilarities for all time series pairs within these pre–clusters. Figure 4 shows the dissimilarity matrix for a dataset with six time series and three pre-clusters; the computed exact dissimilarities of all pairs of time series within a cluster are shown as colored, filled dots in each pre-cluster. (ii) The *medoids* step determines the medoid (colored diamond) for each pre-cluster and computes the dissimilarities of all medoid pairs (colored, filled dots). The medoid dissimilarities are more precise than the approximated dissimilarities and are, therefore, used to overwrite all inter-pre-cluster pairs of corresponding pre-clusters to improve their dissimilarity estimations. (iii) The *inter-pre-cluster* step computes the exact dissimilarities of all inter-pre-cluster pairs that were set to the medoid dissimilarities in the previous step (light gray dots).

We compute the dissimilarities between time series from close pre-clusters before those between far-apart pre-clusters. The final step covers the most dissimilarity computations and, thus, takes most of the runtime. However, inter-cluster pairs also tend to be farther apart than the previous pairs, so that updating the medoid-estimated dissimilarities once more with the exact dissimilarities causes fewer changes to the dendrogram than the updates before, again causing updates to the dendrogram to arrive progressively.

In our evaluation (Section 6.3), we demonstrate that our novel *ada* and *precl* strategies significantly outperform the other strategies and that *ada* scales better than *precl* to larger time series collections.

3.3 Measuring progressiveness

To evaluate the effectiveness of our PrAC algorithm and its ordering strategies, some measure of *progressiveness* should capture the speed of convergence for a gradually improving dendrogram toward an exact solution. Because no such measure currently exists for this setting, we propose a novel convergence measure called *Area under the WHS-runtime-curve (WHS-R-AUC)*. It periodically assesses the similarity between the current approximate dendrogram and the final exact dendrogram using our novel dendrogram similarity measure *weighted hierarchy similarity (WHS)*. The faster the dendrogram similarity approaches 1.0, the better the progressive algorithm and strategy are. Because the final exact dendrogram is not available in practice, we additionally propose an unsupervised convergence indicator, called *#CumClusterChanges@k*, in Section 5 to communicate the convergence to users. In the following, we first discuss a suitable similarity measure for dendrograms, and then utilize it to formulate the convergence measure WHS-R-AUC.

Similarity measure. Internal clustering evaluation measures [16], such as the Silhouette coefficient or the Davies-Bouldin index, are not applicable for measuring the similarity of two dendrograms by definition: These unsupervised measures assess the cluster cohesion of a single flat clustering but cannot judge the similarity between two clusterings. Traditional external clustering evaluation measures, such as (A)RI, (A)MI, or the Dice index, compare two flat clusterings, which are just single cuts of the dendrogram [46]. Because deciding where to cut the dendrogram has a considerable impact on the measured similarity and the desired number of clusters is problem-specific, our similarity measure should consider all potential dendrogram cuts.

We propose the dendrogram similarity measure *weighted hierarchy similarity (WHS)* to quantify the similarity of an approximated dendrogram to the exact dendrogram: For every cluster in the approximated dendrogram, we find the most similar cluster in the exact dendrogram; then, we take the average of all cluster similarities as the dendrogram similarity. Searching for the most similar cluster is necessary because dendrograms may vary in their structure so that level-wise cluster matches are rather inaccurate. Because we can represent clusters of the dendrogram as sets of time series identifiers, we can define the most similar cluster as the partner cluster with the highest Jaccard similarity. For the comparison, we propose a greedy matching approach because calculating all cluster similarities and finding a perfect bi-partite matching is expensive. It exploits that larger clusters are more indicative of the final clustering result than smaller clusters; thus, we match clusters starting from the top of the dendrogram.

To calculate the WHS between an approximated stepwise dendrogram $\widetilde{H}$ and the target stepwise dendrogram H, we first compute all clusters for both dendrograms: $\widetilde{C}$ for $\widetilde{H}$ and C for H. Because the singleton clusters and the root cluster are always identical, we average the best match similarity just for the clusters $2n-2$ to $n+1$ (from large clusters to small clusters):

$$WHS(\widetilde{C}, C) = \frac{1}{n-2} \sum_{i=n+1}^{2n-2} \left(max \left\{ J(\widetilde{C}_i, C_j) \mid C_j \in C' \right\} \right)$$

where $J(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$ is the Jaccard similarity of two clusters, and C' is C without the singleton clusters and the root cluster. Once a cluster C_j from the target dendrogram is matched, it is removed from C', so that every target cluster is matched once.

In addition to the greedy matching strategy, we further accelerated the Jaccard similarity calculation with the use of Bloom filter representations for all clusters. Both performance optimizations of the similarity measure lead to approximate results. We did not observe any (significant) differences in precision, but can efficiently calculate the similarities after every dendrogram construction.

Convergence measure. Convergence assesses the qualitative progress of a progressive algorithm during its execution. Because a progressive algorithm arrives at the exact solution when it terminates, we are not interested in the final result quality but rather in the time needed to get *sufficiently close* to the exact result. Because sufficiently close is use-case dependent, we continuously

measure the similarity of the approximated dendrogram to the target dendrogram during the execution of PRAC with the WHS similarity measure. The resulting list of similarities can be plotted against the runtime of the algorithm, as shown in Figure 1. To measure the convergence of the similarities in a single number (*WHS-R-AUC*), we calculate the area under the curve (AUC) [7]:

$$WHS\text{-}R\text{-}AUC = \int_0^{t_{\max}} WHS(\widetilde{C_t}, C)\, dt$$

where $t_{\max}$ is the maximum runtime of the compared algorithms and $\widetilde{C_t}$ represents the clusters of the approximated dendrogram at timestamp t. To calculate *WHS-R-AUC*, we use the *trapezoidal rule*.

4 Scaling PRAC to large datasets

Time series HAC is easy to parallelize and scales almost linearly with the number of cores because the runtime is dominated by independent pairwise dissimilarity computations. So to make PRAC competitive, the latter needs to be parallelized as well. We present how to parallelize both the dissimilarity computations and the dendrogram construction of PRAC (Section 4.1), and how the different dissimilarity computation ordering strategies can use the available resources efficiently. The baseline strategies (*fcfs* and *rand*) and the *tsla* strategy do not perform any initial computations and are, therefore, already very efficient. The *ada* and *precl* strategies, however, require some preparation before candidates for exact dissimilarity computations can be suggested. We describe their implementations and optimizations in Sections 4.2 and 4.3, respectively.

4.1 Parallelizing dissimilarity computations

To process the dissimilarity computations as fast as possible, we use *data-parallelism for the dissimilarity computations*: Following the coordinator-worker pattern, one thread (coordinator) is responsible for generating and tracking the dissimilarity computation jobs and the remaining threads (workers) pull the jobs from the coordinator and perform the dissimilarity computations. The coordinator, first, generates jobs for the approximation of the dissimilarities following the *fcfs* strategy; then, it generates a second wave of jobs for the computation of the exact dissimilarities following one of the dissimilarity computation ordering strategies. For balanced resource utilization and reactivity, PRAC uses *adaptive batch sizes*. Batching groups x dissimilarity computations into a single job (= batch) to be processed by the same worker. The workers track how much time they take to process their jobs and send the processing times back to the coordinator. The coordinator keeps track of all processing times and adjusts the batch size x, such that a certain target processing time (default: 500 ms) is attained. This batching strategy minimizes communications costs while keeping workloads balanced.

Even if the dissimilarity computations account for the majority (up to 95 %) of the runtime, for larger collections, the construction of the dendrograms becomes relatively slow compared to an individual dissimilarity computation. Thus, PRAC also needs to balance the time spent on computing dissimilarities, which is *making progress*, and the time spent on updating the dendrogram, which is *communicating progress*. In the proposed solution, we use *task-parallelism for the dissimilarity computations and the dendrogram construction*: We designate a specific thread to continuously update the dendrogram and use all remaining threads for all other activities. The maximum of dendrogram construction time and dissimilarity computation time determines the dendrogram update periodicity; for larger input collections, this frequency lowers naturally.

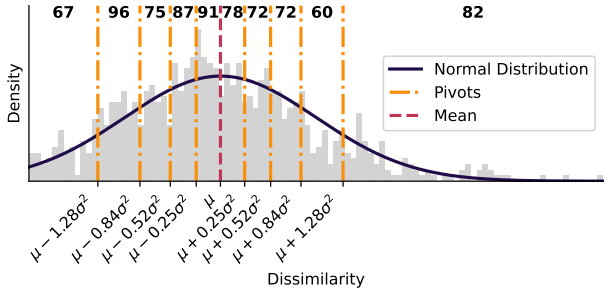


Figure 5: Distribution of dissimilarities for the BeetleFly dataset and the DTW measure. We show a segmentation with 10 segments. Each segment contains roughly the same number of dissimilarities (bold numbers on top).

4.2 Optimizing the *ada* strategy

The *ada* strategy takes as input the approximate dissimilarity vector and generates an order for all time series pairs, starting from the pair with the smallest approximate dissimilarity to that with the largest approximate dissimilarity.

A naive implementation of this strategy would simply create and sort a vector of time series ID pairs and then iterate over this vector. However, this approach requires a significant amount of memory for the vector and a significant amount of time for initialization, during which no dissimilarities are computed. For example, a dataset with $n = 25{,}000$ time series would contain $\binom{n}{2} = 312.487.500$ pairs and take up 11.6 GiB of memory. Sorting this array takes a lot of time compared to the dissimilarity computations, especially if $n > m$. So to scale the *ada* strategy to large datasets, we only approximate the ordering of the time series pairs: the order is based on only approximated dissimilarities anyway. More specifically, instead of sorting time series pairs, we simply divide the dissimilarities into segments with roughly the same cardinality. To minimize memory requirements, we avoid materializing time series' ID pairs and store only segment boundaries and a copy of the current dissimilarity vector (requiring 2.3 GiB in our example). We can then process one segment after the other by generating the ID pairs on demand.

As shown in Figure 5, the (approximated) dissimilarities are not evenly distributed in the value range, but roughly follow a (skewed) normal distribution. Thus, equally spaced segment boundaries would yield segments with diverging cardinalities. With the observation that the dissimilarities usually follow a normal distribution, we can efficiently estimate the parameters of the distribution (μ and σ^2) in a single pass over the data using Welford's algorithm [60], and then use the inverse error function (erf^{-1}) to compute our segment boundaries.

We split the value range into $k = 3 * log(\binom{n}{2})$ segments to balance the quality of the approximated ordering (possibly large k for smaller segments) with the time required to generate time series pairs (possibly small k because we need to iterate over the $\binom{n}{2}$ pairs k times). Because the normal distribution is symmetric, we always generate an even number of segments. The mean μ is our first segment boundary (pivot $x_{k/2}$) separating segment $k/2 - 1$ and $k/2$. The next two pivot values $x_{k/2-1}$ and $x_{k/2+1}$ should enclose (together with μ) $p = 100/k$ % of the dissimilarities,

on the left (for segment $k/2-1$) and right (for segment $k/2$) side of the mean, respectively. We can estimate the x in $\mu + x\sigma^2$ for covering p % of the values using the inverse error function: $x = \sqrt{2} \cdot erf^{-1}(2p)$. For the example in Figure 5, we have $k = 10$ segments. The first pair of pivots each enclose $p = 10$ % of the values between themselves and the mean: $x = \sqrt{2} \cdot erf^{-1}(2 \cdot 0.1) = 0.25$. Thus, the pivots must be at $\mu + 0.25\sigma^2$ and $\mu - 0.25\sigma^2$. The next pair of pivots each enclose $p = 20$ % between themselves and the mean ($x = \sqrt{2} \cdot erf^{-1}(2 \cdot 0.2) = 0.52$), and so forth. The most outer boundaries are $-\infty$ and $+\infty$.

Once we have computed the segment boundaries (in a single pass!), we can generate the time series pairs segment-wise: For each segment and starting with the segment with the smallest dissimilarities, we iterate over the dissimilarity vector and check for each current pair whether its approximate dissimilarity falls into the segment boundaries. If yes, we schedule the pair's exact dissimilarity computation; otherwise, we skip the current pair. When PrAC reaches the end of the dissimilarity vector, we switch to the next segment and re-start the iteration from the beginning. Hence, PrAC scans the dissimilarity vector k times. The order of pairs within one segment is arbitrary, but we ensure that the pairs with the lowest $100/k$ % of dissimilarities are generated first; then, the next $100/k$ % and so on. Even though we scan the dissimilarity vector k times, we (i) allocate only minimal additional memory, (ii) can generate comparison candidates after only a very brief initialization time ($\Theta(n)$), and (iii) can also hide the scanning latency (amortized $O(n/k)$) by pre-computing a batch of time series pairs in advance.

4.3 Optimizing the *precl* strategy

The *precl* strategy requires an assignment of time series to pre-clusters before it can start scheduling any exact dissimilarity computations. We use the existing hierarchical clustering process (MST or NN-chain algorithm) to generate a stepwise dendrogram and then cut the dendrogram, such that the *precl* strategy receives the desired number of clusters. This process is triggered once all approximate dissimilarities have been computed. Unfortunately, constructing and cutting the dendrogram can take a considerable amount of time, during which the workers would not make progress in computing exact dissimilarities. To avoid idle workers, the *precl* strategy instead uses the *fcfs* strategy to schedule some exact dissimilarity computations while it waits for the pre-cluster assignments.

Once the pre-cluster assignments have been received, the *precl* strategy switches to its original three-step processing scheme, starting with the exact intra-pre-cluster dissimilarity calculations. For the first and second processing steps, all dissimilarity computations have the same priority and can be scheduled in any order. For the final inter-pre-cluster step, which contains most of the dissimilarity computations, *precl* creates a partial order for the dissimilarity calculations: It creates pairs of pre-clusters and sorts them in ascending order by their medoid dissimilarity. Sorting the pre-clusters is only in $O\left((n-\sqrt{n})\, log(n-\sqrt{n})\right)$. The inter-pre-cluster dissimilarities are then scheduled, one pre-cluster pair after the other. The individual dissimilarity computations within a pre-cluster pair are again independent and can be computed concurrently. The transitions between processing steps are synchronized to ensure the correctness of the final dendrogram.

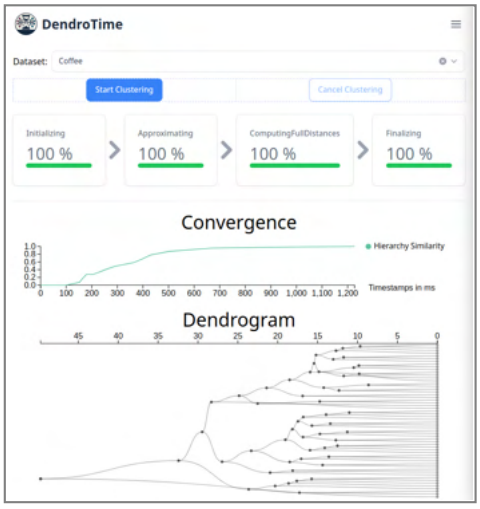


Figure 6: DendroTime's frontend visualizes the computational progress indicator (top), the *unsupervised* convergence indicator (middle), and changes to the dendrogram (bottom).

5 Communicating progress: The DendroTime system

To enable the usage of PrAC in interactive clustering processes, we combined it with the dissimilarity computation ordering strategies and a practical user interface into a system called DendroTime. It consists of a reactive client-server architecture that executes PrAC on a collection of time series in the backend and visualizes dendrogram updates in the frontend, which is shown in Figure 6.

To use DendroTime effectively, scientists need to be able to stop it early. This decision is made based on the current result quality and the expected remaining runtime. Existing progressive algorithms, such as PSNM [44], use *results per second* as an implicit progress indicator because they progressively produce fewer results over time. In contrast, DendroTime is progressive in the result quality and not in the result completeness; it produces a single approximated result, i. e. the stepwise dendrogram, and refines it over time. Thus, in addition to visualizing the dendrogram (and its changes), DendroTime shows two progress indicators to communicate (i) the *computational progress* and (ii) the *convergence* of the clustering.

Computational progress. Because DendroTime knows the number of completed and pending tasks, it can simply display the fraction of completed processing steps to visualize the computational progress. For this, we split the clustering process into four phases and display the percentage of completed processing steps for each phase (cf. Figure 6). The necessary counters are efficient to compute.

Convergence. Visualizing the *qualitative* progress exactly is not always possible because measuring the similarity between the current and the exact dendrogram (using WHS) requires the exact target dendrogram as ground-truth, which is usually not available. For this reason, we estimate the dendrogram convergence in an unsupervised fashion: The approximated dendrogram from a progressive algorithm approaches the exact dendrogram and undergoes fewer and fewer changes towards the end of the algorithm. We, thus, measure dendrogram changes over time, which

we call *#CumClusterChanges@k*, as a proxy measure for the dendrogram convergence, i. e., a visual proxy for WHS. To minimize the computational overhead of the estimation, we count the number of time series that change cluster assignments at a single, fixed cut point of the dendrogram and display the cumulative sum scaled to $[0, 1]$ over DendroTime's runtime (cf. Figure 1). The cut point is determined by a user-defined parameter k for the number of distinct clusters (by default $k = n/2$). Figure 1 compares the progression of the exact WHS quality measure and the *#CumClusterChanges@k* convergence indicator for a progressive and a non-progressive algorithm.

6 Evaluation

In this section, we empirically evaluate our heuristic-driven progressive clustering technique DendroTime to demonstrate the expected convergence behavior in different configurations on various datasets, as introduced in Section 6.1. We first compare the runtime and quality of DendroTime to other exact and approximate hierarchical clustering approaches (Section 6.2). Afterward, we analyze the convergence rate of our dissimilarity computation ordering strategies and demonstrate the effectiveness of *ada* and *precl* (Section 6.3). Then, we analyze the limitations of our progressive approach (Section 6.4).

6.1 Experimental setup

Hardware and software. We perform all experiments on a server with an Intel Xeon E5-2630 v4 CPU (10 physical cores at 2.2 GHz) and 64 GiB of memory. We do not enforce any time or memory limits; swap is disabled. DendroTime is implemented in Scala 3.3 and uses the Akka actor programming framework [29]. We run the system on the OpenJDK JVM version 21.0.5 LTS.

Baseline algorithms. We compare DendroTime with three baseline algorithms: Parallel, JET, and HappieClust. Parallel is a multithreaded Scala implementation of HAC. It computes the pairwise time series dissimilarities in parallel and constructs the final, exact stepwise dendrogram in the end. JET [61] and HappieClust [26] are approximate HAC algorithms for time series. JET computes a pre-clustering using the BIRCH [67] algorithm before applying the NN-chain algorithm with *ward* linkage to build the stepwise dendrogram. We extended JET to support other linkages as well. HappieClust computes pseudo-distances in a low-dimensional pivot-space to estimate the time series dissimilarities and construct the stepwise dendrogram. Both algorithms are implemented in Python, executed with Python version 3.9.21, and compute the dissimilarities in parallel.

Metrics. DendroTime eventually produces the same stepwise dendrogram as traditional HAC algorithms. Thus, we do not evaluate the quality of the final dendrogram but DendroTime's progressive convergence using runtime measurements, WHS, and WHS-R-AUC (cf. Section 3.3). We deliberately avoid traditional clustering metrics because they consider only single dendrogram cuts.

Datasets. We perform our experiments on 123 univariate datasets without missing values taken from the UCR time series classification archive [13], and 12 univariate anomaly datasets from the EDENISS project [51]. In the UCR archive, there are 112 datasets with equal-length time series and 11 datasets with variable-length time series. Further characteristics are listed online[1]. The EDENISS datasets contain between 105 and 2,429 time series with varying lengths between 5 and 576 points per time series – every dataset corresponds to a set of extracted anomalies from a specific subsystem and sensor type.

[1] https://timeseriesclassification.com

6.2 Comparison to baselines

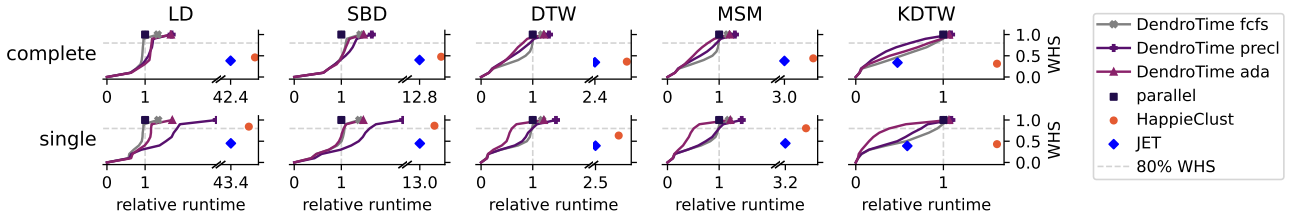
In our first experiment, we compare the performance of DendroTime's *precl*, *ada*, and *fcfs* dissimilarity computation ordering strategies to the baselines Parallel, JET, and HappieClust. To this end, we measure DendroTime's WHS scores over time with different linkage methods (*single*, *complete*, *average*, and *weighted*) and dissimilarity measures (LD, SBD, MSM, DTW, and KDTW), and inspect its convergence behavior. We omit the *tsla* strategy here because it behaves similarly (poorly) to the *fcfs* strategy (cf. Section 6.3); *fcfs* is included as a reference for a non-progressive, incremental algorithm. We use all 135 datasets, and record the algorithms' runtimes and WHS scores. For KDTW, we omit 2 datasets, for which Parallel took longer than 36 h to finish. The WHS calculation costs, i. e., the experiment overhead costs, are measured separately and subtracted from the runtimes for a fair assessment.

Figure 7 (top) plots the measured WHS scores (over time) for all dissimilarities and *single* and *complete* linkage. The other linkage methods behave similarly as *complete* linkage. Parallel, JET, and HappieClust are represented as points because they produce only one result at one specific time, whereas DendroTime outputs (updated) results continuously. Because the datasets in our collection are very heterogeneous in their numbers and lengths of time series, they cause drastically different runtimes. To effectively aggregate the 135 results per setting in Figure 7, we measure the runtime of DendroTime, JET, and HappieClust relative to the runtime of the Parallel baseline. And because the results of Parallel are exact, they also serve as the basis for the quality comparison: For each dataset, we measure the algorithms' WHS scores w. r. t. Parallel's exact dendrogram as $WHS(\widetilde{C}, C_{\text{Parallel}})$.

Figure 7 (bottom) compares the runtime improvement of DendroTime's different dissimilarity ordering strategies for all 20 configurations when they first exceed 80 % quality measured using WHS. Again, we measure the runtime of DendroTime relative to the runtime of the Parallel baseline. If the relative runtime is below 1.0, DendroTime could achieve 0.8 WHS before Parallel computed the exact result.

DendroTime vs. JET/HappieClust. Although JET and HappieClust are approximate algorithms for calculating fast HAC sketches, they are on average slower than DendroTime and even the exact Parallel baseline. This is due to the overhead of their Python implementations on small datasets. For larger datasets and excessively costly dissimilarity measures the overheads diminish: For KDTW, JET is indeed faster than the exact baseline, while HappieClust is just 50 % slower. Nevertheless, DendroTime achieves a better quality after the same runtime for all datasets.

DendroTime vs. Parallel. DendroTime eventually computes the same result as Parallel, but with additional approximation and ordering steps in the process. Hence, it naturally takes more time to finish the exact dendrogram than the Parallel baseline (up to 1.5× for expensive and up to 3× for cheap dissimilarities). The measurements, furthermore, show that DendroTime cannot offer accurate early clusterings for fast to compute dissimilarities, such as LD and SBD, before Parallel finishes the exact dendrogram, because the approximations are nearly as



	LD				SBD				DTW				MSM				KDTW			
	avg	*cmpl*	*sngl*	*wgtd*	*avg*	*cmpl*	*sngl*	*wgtd*	*avg*	*cmpl*	*sngl*	*wgtd*	*avg*	*cmpl*	*sngl*	*wgtd*	*avg*	*cmpl*	*sngl*	*wgtd*
DendroTime *fcfs*	**0.97**	**0.95**	**0.95**	**0.95**	**1.09**	**1.10**	1.07	**1.10**	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.89	0.90	0.93	0.91
DendroTime *ada*	1.20	1.17	1.16	1.16	1.13	1.14	**1.05**	1.14	**0.76**	**0.81**	**0.60**	**0.77**	**0.73**	**0.78**	**0.51**	**0.73**	0.84	0.84	**0.38**	0.84
DendroTime *precl*	1.39	1.20	1.84	1.24	1.38	1.28	1.60	1.34	0.90	0.94	0.99	0.88	0.93	0.95	1.00	0.90	**0.55**	**0.62**	0.75	**0.59**

Figure 7: (Top) Exemplary dendrogram convergence curves for DendroTime, Parallel, JET, and HappieClust with *single* and *complete* linkage and all five dissimilarity measures; other linkage methods show similar behavior as *complete* linkage. The dashed gray lines indicate a WHS of 0.8 and the runtimes of Parallel, respectively. All runtimes were measured relative to the runtime of Parallel and could, thus, be aggregated over all 135 datasets.
(Bottom) Mean relative runtime of DendroTime to reach 80 % quality measured using WHS for all 20 configurations.

costly as the exact computations. For the expensive MSM, DTW, and KDTW dissimilarities, however, DendroTime's *ada* and *precl* strategies can achieve WHSs of above 80 % in shorter runtimes than Parallel despite the continuous calculations of approximate results (cf. Figure 7). This shows that DendroTime provides effective early approximated dendrograms of computationally expensive dissimilarity measures, which are the measures that require these approximations most due to their long calculation times. The measurements also show that the naive *fcfs* strategy converges clearly slower than *ada* and *precl*, which demonstrates the effectiveness of the two more advanced strategies.

DendroTime vs. DendroTime. The overall best progressive dissimilarity ordering strategy is *ada*: It outperforms *precl* in all configurations except with KDTW dissimilarities. Because we later show in Section 6.3 that *precl* actually produces more effective orderings, the advantage of *ada* is its much faster initialization time. For KDTW, *precl* clearly outperforms *ada*. It can even achieve 0.8 WHS below 60 % of Parallel's runtime. Kernel dissimilarities are exceptionally expensive to compute and, thus, *precl*'s overheads are less pronounced and its superior orderings lead to faster convergence. Therefore, we suggest using *precl* for expensive kernel-based measures, such as KDTW, and *ada* for all faster-to-compute dissimilarity measures. For *single* linkage, *ada* converges particularly well because in *single* linkage the minimum pairwise dissimilarity between two clusters determines the clusters' similarity and *ada* computes the (estimated) smallest dissimilarities first. However, the runtime of *precl* with *single* linkage is particularly long because *single* linkage often creates hierarchies that merge single time series into an ever-growing large cluster; the resulting size skew in the pre-clusters of *precl* drastically reduce parallelism and batch sizes.

For datasets with many time series, measuring WHS for the intermediate results of DendroTime becomes prohibitively expensive. Hence, we cannot measure the quality of their approximate results fast enough to plot the convergence curves. As an example, Figure 8 shows the convergence behavior and the runtime breakdown of DendroTime with DTW and *weighted* linkage for two worst-case datasets: Crop with 24,000 time series of size 46 and ElectricDevices with 16,637 time series of size 96. For these two datasets, we can compute the quality for only

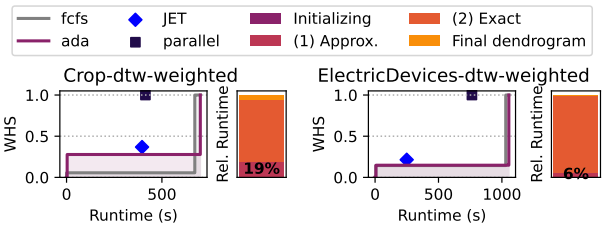


Figure 8: DendroTime's convergence and runtime breakdown for the worst-case datasets Crop and ElectricDevices with thousands of very short (< 100 points) time series.

two dendrograms, leading to a poor-looking, but not necessarily poor-performing convergence behavior (and low WHS-R-AUC); the user-facing dendrogram visualizations still converge progressively. Although these configurations influence DendroTime's convergence curves negatively, we still included them in our aggregated results in Figure 7.

From all strategies in this experiment, the *ada* strategy performs the best. *precl*'s initialization overhead increases with the number of time series in the dataset, leading to a worse convergence rate compared to *ada* on average. Next, we analyze the differences in the DendroTime strategies in more detail.

6.3 Progressive dissimilarity computation

In this experiment, we evaluate DendroTime's convergence effectiveness for each of its dissimilarity computation ordering strategies (*fcfs*, *precl*, *ada*, and *tsla*) and 1,000 random orderings, demonstrating the difficulty of choosing effective orders for progressive dissimilarity computations and the strategies' capabilities. Because PrAC's first phase behaves the same for all our strategies, we consider only its second phase in this experiment, and start with an existing approximated dissimilarity vector. For a consistent and fair assessment, we uniformly sample for every dataset a maximum of 10,000 steps, at which we construct the stepwise dendrogram and record the runtime and WHS; in this way, we can analyze the convergence behavior of the different strategies using WHS-R-AUC. Due to the many random

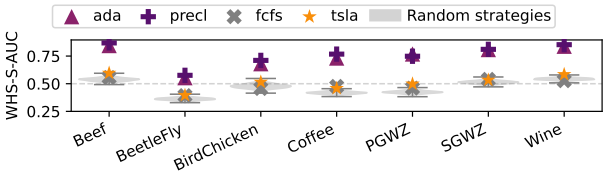


Figure 9: DendroTime's convergence rate measured using WHS-R-AUC for the *fcfs*, *precl*, *ada*, and *tsla* ordering strategies and 1,000 random orderings, all with MSM and *average* linkage for seven UCR datasets. Measurements with other dissimilarity-linkage combinations yield similar results.

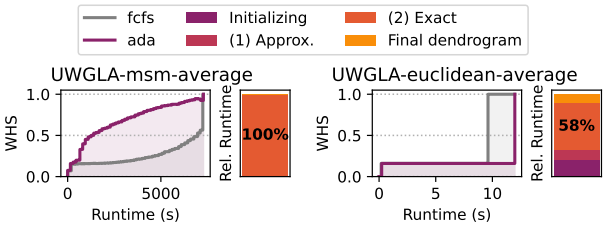


Figure 10: DendroTime's convergence over time and runtime breakdowns for the UWaveGestureLibraryAll dataset with MSM or Euclidean dissimilarities, and *average* linkage.

executions, this evaluation uses only the seven smallest UCR datasets.

Figure 9 shows the WHS-R-AUC of *fcfs*, *precl*, *ada*, and *tsla* as points and the WHS-R-AUC distribution of the 1,000 random orderings as violin plots for each of the seven datasets. The random orderings follow skewed normal distributions around 0.5 or lower, with even their max values being far away from progressive. The fact that none of the 7 × 1,000 random orders is even close to being progressive demonstrates the difficulty of finding a progressive ordering for dissimilarity computations. The simple *fcfs* and *tsla* ordering strategies also fall within the random distributions and cannot be considered effective. The *precl* and *ada* strategies, however, significantly outperform the other strategies: Their orders result in rapidly converging dendrograms and, hence, high WHS-R-AUC scores. Overall, *precl* has the best convergence over all selected datasets, but its initialization is expensive to compute and does not scale to larger time series collections (cf. Section 6.2). The *ada* strategy is slightly less effective, but with its more light-weight initialization and our effective optimizations, it scales better to larger time series collections (cf. Section 6.2). Both *ada* and *precl*, though, clearly provide progressive dissimilarity computation orders.

6.4 Limitations of DendroTime

For the progressive and interactive computation of the dendrogram, DendroTime requires three times more memory than a one-off computation for storing the extra dissimilarity vectors: working set vector, computation ordering vector, and NN-chain vector.

While many time series clustering approaches use expensive, elastic dissimilarity measures, the usage of Euclidean distances is still widespread. Euclidean distances can be computed very efficiently. In this experiment, we look at how DendroTime deals with such efficient dissimilarity measures, for which it was not designed.

Figure 10 shows DendroTime's convergence behavior with runtime breakdowns for MSM dissimilarity (left) and Euclidean distance (right) for a dataset with 4,478 time series of size 945. DendroTime assumes that the dissimilarity computations dominate its runtime and, hence, runs these comparisons progressively. Here, with a cheap dissimilarity measure, such as the Euclidean distance, DendroTime could finish the clustering 607 × faster compared to MSM. Only with Euclidean distances, reading the time series from disk (*Initializing*), computing the approximations (*(1) Approx.*), and constructing the final dendrogram (*Final dendrogram*) could even be measured as overheads. The overall runtime of DendroTime with Euclidean distance is so short that even the quality measure cannot be computed fast enough to visualize the progressive improvements, as discussed in Section 6.2. The usage of Euclidean distances in DendroTime showed the same effect also for other datasets. We conclude that for efficient time series dissimilarity measures, our two-phased approach is ineffective because initialization and approximation eat away most progressive gains of a short exact calculation phase.

7 Anomaly clustering case study

Detecting anomalous subsequences in time series data is one of the most important tasks in time series analytics for domains, such as environmental monitoring [10], preventive healthcare [3], or predictive maintenance [63]. Domain-specialized anomaly detection algorithms have shown to *detect* the location, magnitude, and length of anomalous subsequences accurately [40, 42, 66]. Semi- and unsupervised anomaly detection algorithms, however, cannot *distinguish* or *classify* different types of anomalies. For this reason, anomaly detection pipelines often apply an additional downstream classification step to distinguish different anomaly types, such as sensor failures, environmental influences, false predictions, or actual domain events. The anomaly classification step is usually implemented as unsupervised time series clustering [23, 51, 56, 61] because labeled training data is often of insufficient quantity.

Clustering time series *anomalies* efficiently is especially challenging because (i) the anomaly extraction process often extracts similar anomalies with varying lengths and shifted positions, (ii) the same anomaly type can appear with different amplitudes and frequencies, and (iii) the number of anomaly types (clusters) is not known in advance. DendroTime is particularly useful in this scenario because it can leverage elastic dissimilarity measures to cluster variable length input time series, does not require users to specify the number of clusters upfront, and allows users to inspect the approximate results and stop the clustering early.

In this section, we demonstrate the usage of DendroTime in a real-world anomaly clustering scenario for bio-regenerative life support system telemetry from the EDEN ISS project [65].

EDEN ISS project. EDEN ISS was a research greenhouse operated by the German Aerospace Center in Antarctica between 2018 and 2021 to study plant growth in harsh environments. We focus on the ICS subsystem that records temperatures around the growing lamps above each growth tray. Rewicki et al. [51] identified nine anomaly types in the temperature recordings using an interactive analysis process: Anomaly detection and extraction were performed in an ensemble consisting of the algorithms Maximally Divergent Intervals (MDI) [4] with an anomaly score

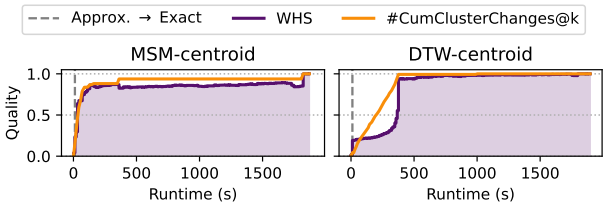


Figure 11: DendroTime's convergence using MSM dissimilarities (left) and DTW dissimilarities (right) with *centroid* linkage on the EDENISS-ICS-full dataset.

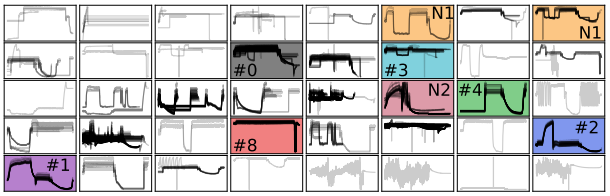


Figure 12: Anomaly types (clusters) found in ICS temperatures using the partial execution of DendroTime. Named anomaly types are annotated in the same color and with the same number as in Rewicki et al. [51, Figure 5, Table A3] (six out of nine), newly discovered anomaly types are *N1* and *N2*.

threshold of 0.5 and Discord Aware Matrix Profile (DAMP) [30], which have been shown to detect complementary anomalies [50, 51]. All extracted temperature anomalies were subsequently clustered using k-Means or HAC to identify recurring anomalous behaviors.

Because larger data sizes could not be explored interactively, Rewicki et al. [50] restricted their analysis to a single year (2020) of the four-year mission and used a resolution of only 1/300 Hz. This resulted in 1,303 anomalous subsequences with an average length of 162 points (ICS-restricted). For our experiment, we use *all* available data ranging from 2018 to 2020 in its original recorded frequency of 1/60 Hz. In this data, we find 2,103 anomalous subsequences with an average length of 954 points (ICS-full).

DendroTime study. To demonstrate that DendroTime allows the ICS-full dataset in the interactive clustering process, we first execute the baseline algorithm Parallel on both datasets and measure its runtimes. Then, we execute DendroTime's *ada* strategy on the datasets and stop the clustering process when the convergence indicator signals a good convergence (indicated by leveling out around 1.0). We use DTW and *centroid* linkage for both algorithms because this was the configuration used in the original analysis.

Parallel can cluster the anomalies of the ICS-restricted dataset in 18 s. For the ICS-full dataset, Parallel requires 31 min, which is too slow for an interactive process. DendroTime, however, can achieve convergence after 6.8 min and with 543,414 of 2,210,253 (exact) dissimilarities computed. This partial execution of DendroTime can achieve a WHS of 93.6 %, as DendroTime's convergence curves for ICS-full in Figure 11 show. DendroTime converges better with MSM than with DTW dissimilarities, confirming our previous results in Section 6.2. However, to compare DendroTime's results to the original study [51], we need to execute it with DTW for this use case. While DendroTime's partial runtime on the full dataset is significantly longer than the original algorithm's runtime on the restricted dataset, it is still short enough to be used in an interactive analysis process.

The result of DendroTime's partial execution allows us to perform the same analysis as in [50] and to identify existing and even some new anomaly types: As already pointed out in [51], HAC algorithms produce unbalanced dendrograms on EDEN ISS data, leading to few large clusters and many tiny clusters for many cut points. This prevents us from selecting a good number of clusters k and the respective cut point automatically (too few clusters). After visual inspection of the dendrogram, we manually select a cut point that results in 40 clusters, which is not too many clusters with not too many instances. The 40 clusters (anomaly types) are shown in Figure 12. We can easily identify six out of the nine original anomaly types: *Anomalous Night Phase (#0)*, *Long Peak (#1)*, *Short Peak (#2)*, *Near Flat Noisy or Flat Signal (#3)*, *Missing / Delayed Warmup (#4)* and *Flat and Drop (#8)*. Additionally, we can identify two previously unknown anomaly types: *Long Daydrop (N1)* and *Interrupted Peak (N2)*. *N1* was not detected previously, despite appearing on two different days in 2020. *N2* can be observed only in 2019.

8 Related work

Time series clustering. DendroTime is the first *progressive* HAC algorithm for time series. Many non-progressive algorithms have been proposed to cluster time series, which can broadly be categorized into two classes [1, 22]: *sequence-based* clustering methods that work directly with the time series using time series-specific dissimilarity measures, and *feature-based* clustering methods that extract time series properties in a preprocessing step and then apply standard (tabular) clustering algorithms on these features.

Sequence-based clustering algorithms, such as k-Means [22], k-Shapes [41], OPTICS [2], HAC [1, 24, 41], or DTCR [32], achieve competitive performances only if they utilize a time series-specific dissimilarity measure, such as SBD or MSM. Holder et al. [22] analyzed elastic dissimilarity measures for clustering time series with k-Means and k-Medoids. Javed et al. [24] benchmarked partition-based, density-based, and hierarchical time series clustering algorithms, while Lafabregue et al. [27] benchmarked various deep learning network architectures for time series clustering.

Feature-based approaches to time series clustering can utilize any traditional tabular clustering algorithm, but rely on an effective set of time series features. Examples are using statistical features with k-Means to cluster customer-specific electricity usage [48] or applying k-Medoids on time series bit representations extracted with piecewise aggregate approximation (PAA) [28]. The `tsfresh` [11] and `catch22` [31] libraries provide time series-specific feature extraction methods that are based on statistical characteristics. Time2Feat [8] uses learned time series features.

Hierarchical agglomerative clustering. HAC is a family of bottom-up hierarchical clustering algorithms that is often used to cluster time series for three reasons: It can leverage elastic dissimilarities to cluster variable-length time series, it does not require the user to specify the number of clusters upfront, and it produces easy to visualize results [1, 14, 24, 41]. Most approaches were proposed with the ED in mind. However, ED does not work well for variable-length time series, and many popular time series dissimilarity measures, such as DTW and SBD, do not satisfy

metric properties. The widely adopted NN-chain algorithm [38] for HAC still works as long as the chosen linkage method is compatible with the dissimilarity measure.

Murtagh and Contreras [39] give a general overview of HAC for ED. Schubert [54] proposes optimized versions of the NN-chain algorithm for ED using an incremental nearest neighbor search approach that can achieve sub-quadratic runtime. Unfortunately, these algorithms cannot be applied to non-metric dissimilarities.

Existing parallel versions of HAC [15, 35, 58, 59, 64] focus on parallelizing the clustering step that expects a complete distance matrix as input. For sliding and elastic time series dissimilarities, though, computing the distance matrix takes most of the runtime and is, therefore, subject to our parallel and progressive strategies. Existing approaches are, for this reason, orthogonal to our approach, but they could be used to speed up the final dendrogram construction.

Approximate hierarchical agglomerative clustering. To control HAC's computational complexity, a few non-progressive, approximate algorithms have been presented:

Wenig et al. [61] propose the unsupervised approximate algorithm JET for clustering large collections of variable-length time series. It first computes a coarse-grained pre-clustering using a cheap feature-based dissimilarity measure before building a dendrogram using the expensive elastic dissimilarity measure SBD, similar to our *precl* strategy. We use JET as a non-progressive baseline.

Kull and Vilo [26] propose the approximate HAC algorithm HappieClust for clustering biological gene expressions. HappieClust uses pseudo-distances within a low-dimensional pivot-space to identify pairs of similar objects. After computing the exact dissimilarities between these similar objects, it utilizes the triangle inequality principle to approximate the missing dissimilarities and perform agglomerative clustering. HappieClust is another non-progressive baseline.

Cochez and Mou [12] combine the twister tries data structure with locality-sensitive hashing to implement an approximate HAC algorithm for average linkage. The algorithm places strict requirements on the dissimilarity measure: It must be a distance metric with a corresponding *proportionally sensitive family* of locality-sensitive hash functions. Because many time series dissimilarity measures do not even fulfill the metric requirements, twister tries are challenging to apply to time series clustering. This also applies to Koga et al.'s LSH-Link algorithm [25], which also uses locality-sensitive hashing but for single linkage. DendroTime's approach is independent of the used dissimilarity measure and linkage method.

9 Conclusion

DendroTime is a parallel, progressive clustering system for large collections of time series. It creates and continuously improves an approximate dendrogram, which eventually converges to the exact solution. DendroTime's two-phased approach and its dissimilarity computation ordering strategies are most efficient for expensive dissimilarity measures, which are necessary (and popular) to effectively cluster time series. We compared DendroTime to an exact and an approximate baseline algorithm with excellent results. In future work, we aim to develop a strategy to calculate the individual time series dissimilarities incrementally, so that partial results from the approximation can be re-used by the exact calculations to minimize DendroTime's overhead.

Artifacts

The source code, data, and other artifacts have been made available at https://github.com/hpi-information-systems/dendrotime.

References

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. 2015. Time-series clustering – A decade review. *Information Systems*, 53, 16–38. DOI: 10.1016/j.is.2015.04.007.

[2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: ordering points to identify the clustering structure. *SIGMOD Record*, 28, 2, 49–60. DOI: 10.1145/304181.304187.

[3] Sardar Ansari, Negar Farzaneh, Marlena Duda, Kelsey Horan, Hedvig B. Andersson, Zachary D. Goldberger, Brahmajee K. Nallamothu, and Kayvan Najarian. 2017. A Review of Automated Methods for Detection of Myocardial Ischemia and Infarction Using Electrocardiogram and Electronic Health Records. *IEEE Reviews in Biomedical Engineering*, 10, 264–298. DOI: 10.1109/RBME.2017.2757953.

[4] Björn Barz, Erik Rodner, Yanira Guanche Garcia, and Joachim Denzler. 2018. Detecting regions of maximal divergence for spatio-temporal anomaly detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41, 1088–1101, 5.

[5] Lukas Berg, Tobias Ziegler, Carsten Binnig, and Uwe Röhm. 2019. ProgressiveDB: progressive data analytics as a middleware. *Proceedings of the VLDB Endowment (PVLDB)*, 12, 12, 1814–1817. DOI: 10.14778/3352063.3352073.

[6] Donald J. Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 359–370.

[7] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proceedings of the VLDB Endowment (PVLDB)*, 11, 3, 311–323. DOI: 10.14778/3157794.3157800.

[8] Angela Bonifati, Francesco Del Buono, Francesco Guerra, and Donato Tiano. 2022. Time2Feat: learning interpretable representations for multivariate time series clustering. *Proceedings of the VLDB Endowment (PVLDB)*, 16, 2, 193–201. DOI: 10.14778/3565816.3565822.

[9] Yanhong Chen, Haibin Cai, Yiqing Gong, Kun Lu, Jingqiao Mao, Weiyu Chen, Kang Wang, Huan Gao, and Mingming Tian. 2025. Diurnal distribution of phytoplankton in large shallow lakes based on time series clustering. *Ecological Informatics*, 90. DOI: 10.1016/j.ecoinf.2025.103250.

[10] Haibin Cheng, Pang-Ning Tan, Christopher Potter, and Steven Klooster. 2009. Detection and Characterization of Anomalies in Multivariate Time Series. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 413–424. DOI: 10.1137/1.9781611972795.36.

[11] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. 2018. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing*, 307, 72–77. DOI: 10.1016/j.neucom.2018.03.067.

[12] Michael Cochez and Hao Mou. 2015. Twister Tries: Approximate Hierarchical Agglomerative Clustering for Average Distance in Linear Time. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 505–517. DOI: 10.1145/2723372.2751521.

[13] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6, 6, 1293–1305. DOI: 10.1109/JAS.2019.1911747.

[14] Hoang Anh Dau, Diego Furtado Silva, François Petitjean, Germain Forestier, Anthony Bagnall, Abdullah Mueen, and Eamonn Keogh. 2018. Optimizing dynamic time warping's window width for time series data mining applications. *Data Mining and Knowledge Discovery*, 32, 4, 1074–1120. DOI: 10.1007/s10618-018-0565-y.

[15] Laxman Dhulipala, David Eisenstat, Jakub Łącki, Vahab Mirronki, and Jessica Shi. 2022. Hierarchical Agglomerative Graph Clustering in Poly-Logarithmic Depth. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 22925–22940.

[16] Vladimir Estivill-Castro. 2002. Why so many clustering algorithms: a position paper. *SIGKDD Explorations Newsletter*, 4, 1, 65–75. DOI: 10.1145/568574.568575.

[17] Brian S. Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. 2011. *Cluster Analysis*. (5th ed.). John Wiley & Sons, Ltd. 352 pp. ISBN: 978-0-470-97781-1. DOI: 10.1002/9780470977811.

[18] Simon Fong. 2012. Using hierarchical time series clustering algorithm and wavelet classifier for biometric voice classification. *Journal of Biomedicine and Biotechnology*, 2012. DOI: 10.1155/2012/215019.

[19] Lincheng Han, Shun Cheng, Weiru Yuan, Xiuyu Zhang, and Jianguo Wang. 2026. A novel nonparametric bayesian model for time series clustering: application to electricity load profile characterization. *Future Generation Computer Systems - The International Journal of Escience*, 174. DOI: 10.1016/j.future.2025.107929.

[20] Pierre Hansen and Brigitte Jaumard. 1997. Cluster analysis and mathematical programming. *Mathematical Programming*, 79, 1, 191–215. DOI: 10.1007/BF02614317.

[21] Christopher Holder, Anthony Bagnall, and Jason Lines. 2024. On time series clustering with k-means. arXiv: 2410.14269 [cs]. Pre-published.

[22] Christopher Holder, Matthew Middlehurst, and Anthony Bagnall. 2024. A review and evaluation of elastic distance functions for time series clustering. *Knowledge and Information Systems*, 66, 2, 765–809. DOI: 10.1007/s10115-023-01952-0.

[23] Satoshi Iwata and Kenji Kono. 2012. Clustering performance anomalies based on similarity in processing time changes. *Information and Media Technologies*, 7, 1, 141–152. DOI: 10.11185/imt.7.141.

[24] Ali Javed, Byung Suk Lee, and Donna M. Rizzo. 2020. A benchmark study on time series clustering. *Machine Learning with Applications*, 1, 100001. DOI: 10.1016/j.mlwa.2020.100001.

[25] Hisashi Koga, Tetsuo Ishibashi, and Toshinori Watanabe. 2004. Fast Hierarchical Clustering Algorithm Using Locality-Sensitive Hashing. In *Discovery Science*, 114–128. DOI: 10.1007/978-3-540-30214-8_9.

[26] Meelis Kull and Jaak Vilo. 2008. Fast approximate hierarchical clustering using similarity heuristics. *BioData Mining*, 1, 1, 9. DOI: 10.1186/1756-0381-1-9.

[27] Baptiste Lafabregue, Jonathan Weber, Pierre Gançarski, and Germain Forestier. 2022. End-to-end deep representation learning for time series clustering: a comparative study. *Data Mining and Knowledge Discovery*, 36, 1, 29–81. DOI: 10.1007/s10618-021-00796-y.

[28] Guiling Li, Olli Bräysy, Liangxiao Jiang, Zongda Wu, and Yuanzhen Wang. 2013. Finding time series discord based on bit representation clustering. *Knowledge-Based Systems*, 54, 243–254. DOI: 10.1016/j.knosys.2013.09.015.

[29] [SW] Lightbend Inc., Akka: Build Powerful Reactive, Concurrent, and Distributed Applications More Easily version 2.6.3, 2020.

[30] Yue Lu, Renjie Wu, Abdullah Mueen, Maria A Zuluaga, and Eamonn Keogh. 2022. Matrix Profile XXIV: Scaling Time Series Anomaly Detection to Trillions of Datapoints and Ultra-fast Arriving Data Streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 49.

[31] Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. 2019. Catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery*, 33, 6, 1821–1852. DOI: 10.1007/s10618-019-00647-x.

[32] Qianli Ma, Jiawei Zheng, Sen Li, and Gary W Cottrell. 2019. Learning representations for time series clustering. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[33] Jakub Maciejewski, Konstantinos Nikoletos, George Papadakis, and Yannis Velegrakis. 2025. Progressive Entity Matching: A Design Space Exploration. *Proceedings of the ACM on Management of Data*, 3, 1, 1–25. DOI: 10.1145/3709715.

[34] Pierre-François Marteau and Sylvie Gibet. 2015. On Recursive Edit Distance Kernels With Application to Time Series Classification. *IEEE Transactions on Neural Networks and Learning Systems*, 26, 6, 1121–1133. DOI: 10.1109/TNNLS.2014.2333876.

[35] Nicholas Monath et al. 2021. Scalable Hierarchical Agglomerative Clustering. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 1245–1255. DOI: 10.1145/3447548.3467404.

[36] Daniel Müllner. 2011. Modern hierarchical, agglomerative clustering algorithms. arXiv: 1109.2378 [cs]. Pre-published.

[37] Fionn Murtagh. 1984. Complexities of Hierarchic Clustering Algorithms: State of the Art. *Computational Statistics Quarterly (CSQ)*, 1, 2, 101–113.

[38] Fionn Murtagh. 1985. *Multidimensional Clustering Algorithms*. Physica-Verlag HD. 131 pp. ISBN: 3-7051-0008-4.

[39] Fionn Murtagh and Pedro Contreras. 2017. Algorithms for hierarchical clustering: an overview, II. *WIREs Data Mining and Knowledge Discovery*, 7, 6, 1219. DOI: 10.1002/widm.1219.

[40] Antonios Ntroumpogiannis, Michail Giannoulis, Nikolaos Myrtakis, Vassilis Christophides, Eric Simon, and Ioannis Tsamardinos. 2023. A meta-level analysis of online anomaly detectors. *The VLDB Journal*, 32, 4, 845–886. DOI: 10.1007/s00778-022-00773-x.

[41] John Paparrizos and Luis Gravano. 2017. Fast and Accurate Time-Series Clustering. *ACM Transactions on Database Systems*, 42, 2, 8:1–8:49. DOI: 10.1145/3044711.

[42] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. 2022. TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection. In *Proceedings of the VLDB Endowment (PVLDB)*, 2150–8097. DOI: 10.14778/3529337.3529354.

[43] John Paparrizos, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2020. Debunking Four Long-Standing Misconceptions of Time-Series Distance Measures. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1887–1905. DOI: 10.1145/3318464.3389760.

[44] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. 2015. Progressive Duplicate Detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27, 5, 1316–1329. DOI: 10.1109/TKDE.2014.2359666.

[45] S. Parthasarathy. 2002. Efficient progressive sampling for association rules. In *Proceedings of the International Conference on Data Mining (ICDM)*, 354–361. DOI: 10.1109/ICDM.2002.1183923.

[46] Darius Pfitzner, Richard Leibbrandt, and David Powers. 2009. Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, 19, 3, 361–394. DOI: 10.1007/s10115-008-0150-6.

[47] Andrea Pietracaprina, Matteo Riondato, Eli Upfal, and Fabio Vandin. 2010. Mining top-K frequent itemsets through progressive sampling. *Data Mining and Knowledge Discovery*, 21, 2, 310–326. DOI: 10.1007/s10618-010-0185-7.

[48] Teemu Räsänen and Mikko Kolehmainen. 2009. Feature-based clustering for electricity use time series data. In *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA)*, 401–412.

[49] Chotirat Ann Ratanamahatana and Eamonn Keogh. 2005. Three Myths about Dynamic Time Warping Data Mining. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 506–510. DOI: 10.1137/1.9781611972757.50.

[50] Ferdinand Rewicki, Joachim Denzler, and Julia Niebling. 2023. Is It Worth It? Comparing Six Deep and Classical Methods for Unsupervised Anomaly Detection in Time Series. *Applied Sciences*, 13, 3, 1778. DOI: 10.3390/app13031778.

[51] Ferdinand Rewicki, Jakob Gawlikowski, Julia Niebling, and Joachim Denzler. 2024. Unraveling Anomalies in Time: Unsupervised Discovery and Isolation of Anomalous Behavior in Bio-Regenerative Life Support System Telemetry. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD) - Applied Data Science Track*, 207–222. DOI: 10.1007/978-3-031-70378-2_13.

[52] F. James Rohlf. 1973. Algorithm 76. Hierarchical Clustering Using the Minimum Spanning Tree. *The Computer Journal*, 16, 93–95.

[53] Sebastian Schmidl, Felix Naumann, and Thorsten Papenbrock. 2024. AutoTSAD: Unsupervised Holistic Anomaly Detection for Time Series Data. *Proceedings of the VLDB Endowment (PVLDB)*, 17, 11, 2987–3002. DOI: 10.14778/3681954.3681978.

[54] Erich Schubert. 2025. Hierarchical Clustering Without Pairwise Distances by Incremental Similarity Search. *Similarity Search and Applications*, 15268, 238–252. DOI: 10.1007/978-3-031-75823-2_20.

[55] Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. 2018. Schema-Agnostic Progressive Entity Resolution. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 53–64. DOI: 10.1109/ICDE.2018.00015.

[56] Catherine Stamoulis. 2020. A spectral clustering approach for the classification of waveform anomalies in high-dimensional brain signals. In *Proceedings of the International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 50–53. DOI: 10.1109/EMBC44109.2020.9176369.

[57] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. 2013. The Move-Split-Merge Metric for Time Series. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25, 6, 1425–1438. DOI: 10.1109/TKDE.2012.88.

[58] Baris Sumengen, Anand Rajagopalan, Gui Citovsky, David Simcha, Olivier Bachem, Pradipta Mitra, Sam Blasiak, Mason Liang, and Sanjiv Kumar. 2021. Scaling Hierarchical Agglomerative Clustering to Billion-sized Datasets. arXiv: 2105.11653 [cs]. Pre-published.

[59] Yue Wang, Vivek Narasayya, Yeye He, and Surajit Chaudhuri. 2022. PACk: an efficient partition-based distributed agglomerative hierarchical clustering algorithm for deduplication. *Proceedings of the VLDB Endowment (PVLDB)*, 15, 6, 1132–1145. DOI: 10.14778/3514061.3514062.

[60] B. P. Welford. 1962. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4, 3, 419–420. DOI: 10.1080/00401706.1962.10490022.

[61] Phillip Wenig, Mathias Höfgen, and Thorsten Papenbrock. 2024. JET: Fast Estimation of Hierarchical Time Series Clustering. In *Proceedings of the International Conference on Time Series and Forecasting (ITISE)*, 37. DOI: 10.3390/engproc2024068037.

[62] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. 2013. Pay-As-You-Go Entity Resolution. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25, 5, 1111–1124. DOI: 10.1109/TKDE.2012.43.

[63] Mark Woike, Ali Abdul-Aziz, and Michelle Clem. 2014. Structural health monitoring on turbine engines using microwave blade tip clearance sensors. In *Proceedings of the International Conference on Smart Sensor Phenomena, Technology, Networks, and Systems Integration (SPIE)*. DOI: 10.1117/12.2044967.

[64] Shangdi Yu, Yiqiu Wang, Yan Gu, Laxman Dhulipala, and Julian Shun. 2021. ParChain: a framework for parallel hierarchical agglomerative clustering using nearest-neighbor chain. *Proceedings of the VLDB Endowment (PVLDB)*, 15, 2, 285–298. DOI: 10.14778/3489496.3489509.

[65] Paul Zabel, Matthew Bamsey, Conrad Zeidler, Vincent Vrakking, Daniel Schubert, and Oliver Romberg. 2017. Future exploration greenhouse design of the eden iss. In *Proceedings of the International Conference on Environmental Systems (ICES)*.

[66] Aoqian Zhang, Shuqing Deng, Dongping Cui, Ye Yuan, and Guoren Wang. 2024. An Experimental Evaluation of Anomaly Detection in Time Series. *Proceedings of the VLDB Endowment (PVLDB)*, 17, 3, 483–496. DOI: 10.14778/3632093.3632110.

[67] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25, 2, 103–114. DOI: 10.1145/235968.233324.

[68] Xin Zhang and Ahmed Eldawy. 2023. Less is More: How Fewer Results Improve Progressive Join Query Processing. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 1–12. DOI: 10.1145/3603719.3603728.