TiVy: Time Series Visual Summary for Scalable Visualization

Gromit Yeuk-Yin Chan, Luis Gustavo Nonato, Themis Palpanas, Cláudio T. Silva, Juliana Freire

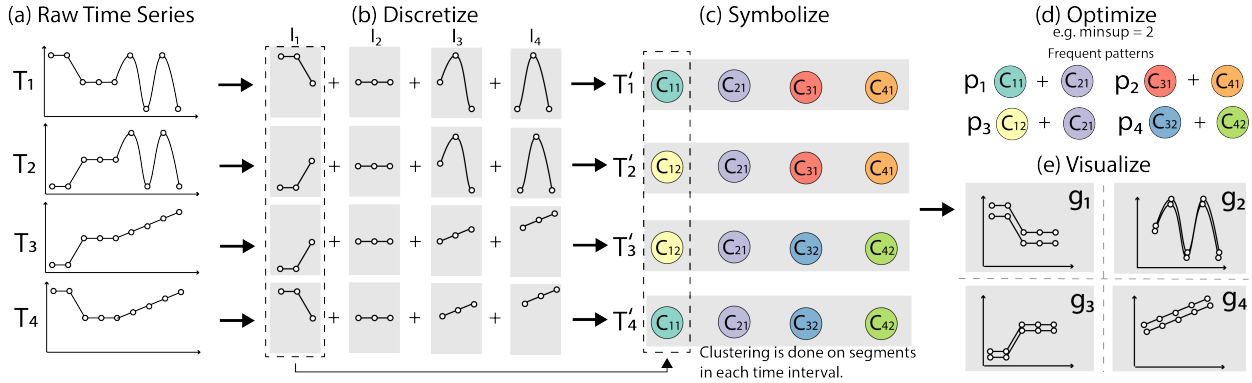

Fig. 1: Pipeline for extracting the time series patterns (g_1, g_2, g_3, g_4) from (a). (b) The series are discretized into sets of time segments with equal length. (c) The segments in each time interval are labeled with the cluster to which they belong. (d) The groupings of time segments are optimized based on the frequent patterns from the sequences in (c). e.g., if $minsup = 2$ (i.e., each output group has to contain at least 2 time series), the results (e) will be subsequences from $C_{11} + C_{21}$, $C_{12} + C_{21}$, $C_{31} + C_{41}$ and $C_{32} + C_{42}$.

Abstract—Visualizing multiple time series presents fundamental tradeoffs between scalability and visual clarity. Time series capture the behavior of many large-scale real-world processes, from stock market trends to urban activities. Users often gain insights by visualizing them as line charts, juxtaposing or superposing multiple time series to compare them and identify trends and patterns. However, existing representations struggle with scalability: when covering long time spans, leading to visual clutter from too many small multiples or overlapping lines. We propose TiVy, a new algorithm that summarizes time series using sequential patterns. It transforms the series into a set of symbolic sequences based on subsequence visual similarity using Dynamic Time Warping (DTW), then constructs a disjoint grouping of similar subsequences based on the frequent sequential patterns. The grouping result, a visual summary of time series, provides uncluttered superposition with fewer small multiples. Unlike common clustering techniques, TiVy extracts similar subsequences (of varying lengths) aligned in time. We also present an interactive time series visualization that renders large-scale time series in real-time. Our experimental evaluation shows that our algorithm (1) extracts clear and accurate patterns when visualizing time series data, (2) achieves a significant speed-up (1000×) compared to a straightforward DTW clustering. We also demonstrate the efficiency of our approach to explore hidden structures in massive time series data in two usage scenarios.

1 Introduction

Time series data analysis is prevalent in many applications. Analysts explore time series to discover patterns in urban activities (e.g., noise, traffic, and weather) [6, 21, 45], identify trends in highly volatile financial markets [62], and group human mobility patterns from wireless telecommunication traffic data series [46, 52–54]. Visual exploration is often used as the first step to obtain insights and formulate hypotheses. Many visualization primitives have been proposed [43, 48, 60, 69] to support temporal data analysis. When exploring large collections of time series data, analysts face a challenging decision [36]. They can either plot each series in separate small multiples, which provides clarity but becomes unwieldy with many series, or superpose all series in a single chart, which scales better but quickly deteriorates into visual clutter when lines overlap and obscure patterns (Figure 2(a) and (b)).

- *Gromit Yeuk-Yin Chan is with Adobe Research (work done while at NYU and visiting Université Paris Cité). E-mail: ychan@adobe.com.*
- *Cláudio T. Silva, Juliana Freire are with New York University. E-mail: {csilva, juliana.freire}@nyu.edu.*
- *L.G. Nonato is with University of Sao Paulo. E-mail: gnonato @icmc.usp.br*
- *Themis Palpanas is with Université Paris Cité. E-mail: themis@mi.parisdescartes.fr.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

TiVy: Combining Juxtaposition and Superposition. Instead of choosing between these extremes, we selectively superpose time series with similar visual shapes while juxtaposing those with different patterns. This creates an effective visual summary that maintains clarity while scaling to large datasets – showing similar trends together in clean, readable charts while separating distinct patterns into different views. To achieve this, we introduce TiVy, an algorithm that automatically identifies which time series should be grouped together. The key insight is that time series with similar visual shapes can be safely superposed without losing readability, while those with different shapes should be separated. TiVy transforms time series into symbolic sequences based on visual similarity using Dynamic Time Warping (DTW), then uses sequential pattern mining to extract groups of similar subsequences that align in time.

TiVy addresses two key challenges. First, the duration and time intervals of similar trends may vary: while some trends are long, others are short. Second, time series with similar trends need not be perfectly synchronized to have their visual properties highlighted in the chart. As some time shifts within a small time interval are acceptable for users to identify the trends, for clustering, it means that we need a distance measure that takes temporal alignment into account. A popular and well-studied distance measure for this purpose is Dynamic Time Warping (DTW) [42], which aligns well with human perception on visual similarity among time series in a line chart. However, as we discuss in Section 4, applying DTW in our scenario is computationally expensive: not only is the search space of possible subsequence clusters with arbitrary sizes and durations is prohibitively large, but also the distance measure to compare visually aligned time series takes quadratic time to

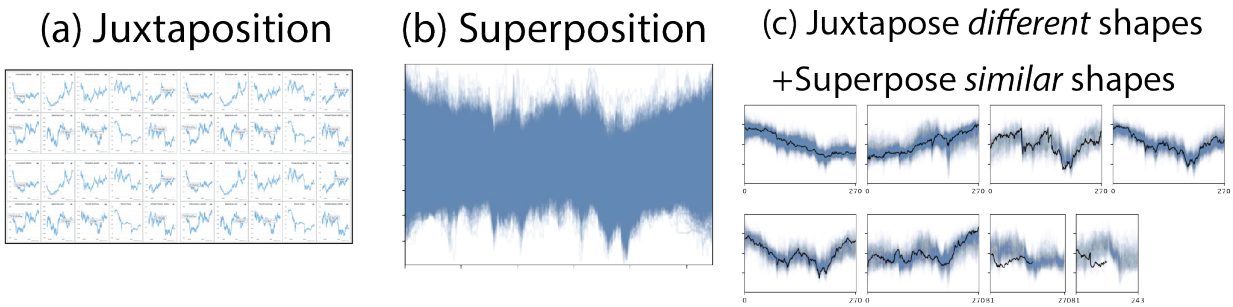

Fig. 2: Illustration of the challenges of visualizing multiple time series. Juxtaposing time series (a) and superposing them (b) have scalability issues with different reasons. Our goal is to (c) perform subsequence clustering that juxtapose similar series and superpose different ones.

compute. For interactive exploration of large datasets, these computational constraints become critical bottlenecks, requiring optimizations that maintain accuracy while achieving real-time performance.

We propose a heuristic technique for extracting time series patterns as subsequence clusters that transforms real-valued series into discrete sequences and extracts the clusters from the sequential patterns. The key components of our approach are: (1) a new symbolic representation for time series that encodes the visual structure of each time series in a discrete format, focusing on the visual shape; and (2) a sequential pattern mining algorithm that interactively groups the discrete sequences to extract time series subsequence clusters for line chart visualizations. Together, they make it possible to display the time series subsequences from each sequential pattern in one small multiple of a line chart with low visual complexity, without the need to change the displayed values (e.g., using signal processing) or visual encoding (e.g., projections).

We propose an interactive system that renders the resulting time series summaries in real-time, enabling exploration of massive datasets that would otherwise be impossible to visualize effectively. We evaluate the performance in terms of speed-ups, qualitative results and limitations. Our main contributions can be summarized as follows: (1) A time series subsequence clustering approach that groups visually and temporally similar time series subsequences from line charts visualization; (2) Efficient algorithms to compute the symbolic representations and extract visual patterns; (3) A usable and scalable visualization interface that leverages the proposed clustering algorithm to explore large volumes of time series data; (4) Usage scenarios with real-world datasets that demonstrate the usefulness of our approach.

2 Related Work

Time Series Visualization. Aigner *et al.* [2] summarize time series visualization techniques based on the data type (i.e., visualizing single or multiple time series) and dimension (i.e., whether it is linear, cyclic, or branching and whether it is a point or an interval). Bach *et al.* [5] see it as operations on a space-time cube including extraction, flattening, filling, and geometric and content transformation.

The first use of time series can be traced back to the 18th century, with the introduction of line charts [69]. Time series data can be encoded in small multiples and sparklines [43] that use (x,y)-positions to encode time. Techniques have also been proposed that include channels such as area or color to provide better scalability and density in the presentation. For example, the horizon chart [8, 60] splits and superimposes a line chart vertically with a few bands of ranges distinguished by color and color fields and uses hues to encode the values. Recently, these techniques have been shown to result in different perceptions of data similarities [28].

Attaining scalability is a major challenge for visualizing multiple time series. Charts can become cluttered even with few time series (i.e., eight in Javad *et al.* [36]). Moritz *et al.* proposed a heat map approach to treat the values in time series as independent pixels and to plot the heat map of lines in one chart [48]. This method reveals the density of lines but hinders patterns from similar time series with small time shifts.

Interactive exploration systems have also addressed the challenge of analyzing multiple time series through various filtering and navigation techniques. TimeSearcher [35] enables users to filter and query similar series via brushing on time intervals and value ranges, while LiveRAC [43] supports batch inspection through a spreadsheet layout. Other approaches use focus+context techniques: ChronoLenses [77] and CareCruiser [30] provide filtering and highlighting, while systems like Continuum [4] and EventRiver [41] employ semantic zooming to navigate different temporal granularities.

Time Series Mining and Visualization. While the techniques described above display time series in their original form, to visualize large-scale data, data abstraction techniques have been proposed to mitigate issues with time series occlusion. (see [63] for an in-depth survey).

Approximation techniques have been developed to reduce the complexity of large-scale time series visualization [22]. The most widely adopted approach is Symbolic Aggregate Approximation (SAX) [40], which discretizes time series into symbolic sequences through piecewise aggregate approximation. SAX-based systems like VizTree [39] and SAX Navigator [58] visualize these symbolic representations using hierarchical trees, while Hao *et al.* [33] focus on motif visualization with colored rectangles. Alternative approximation methods include piece-wise linear approaches that preserve gradient information [49] and hierarchical clustering techniques for interactive exploration [59].

Projection and clustering are used to group similar time series together to reveal underlying patterns. Steiger *et al.* visualize the pairwise distance matrix of multiple time series into 2D projections and a Voronoi diagram [65]. Ward *et al.* use N-grams to segment time series and project the data with PCA [75]. Van Goethem *et al.* apply a method that detects trends of time series at the starting time point and visualize the trends and subtrends with a river metaphor [71]. StreamStory [66] transforms time series into a set of states to visualize the relationships among the time series in a directed graph. For the metrics used, many acceleration on DTW are proposed with heuristics on time series properties [3, 34] or approximate algorithms [61].

Instead of considering all the points in a time series, we can explore the statistical properties such as variances and correlations. Pinus [64] is a triangle matrix metaphor that visualizes the variances of time segments in all combinations of time intervals. Kothur *et al.* [38] visualize the time series as color fields that encode the correlations among other time series. TimeSeer [16] visualizes scagnostics with scatter plots of data attributes at each time index to identify anomalies.

We propose a novel time series summarization technique that transforms time series into compact symbolic sequences, considers the visual features of time series, and produces results that can be visualized in line charts without large visual complexities. Our goal is to compute a set of disjoint partitions of real-value data series, which distinguishes our problem setting from overlapping subsequence clusters [13, 17, 19, 29, 37, 51] and clustering discrete event sequences [26, 50]. Our efficient algorithms that achieve significant speed up (from hours to seconds for 100,000 series) in constructing sequences compared to a straightforward clustering with visualization-oriented similarity metrics and attain interactive speed to extract the time series patterns.

Shape Based Subtrajectory Clustering. While one could relate the techniques from subtrajectory clustering due to their similarities on bundling lines on a screen, they are different in terms of objectives,

Pattern	Support	Pattern	Support	Pattern	Support
c_{11}	2	c_{32}	2	$c_{12}+c_{21}$	2
c_{21}	4	c_{42}	2	$c_{21}+c_{32}$	2
c_{31}	2	$c_{11}+c_{21}$	2	$c_{32}+c_{42}$	2
c_{41}	2	$c_{21}+c_{31}$	2		
c_{12}	2	$c_{31}+c_{41}$	2		

Table 1: Patterns in Figure 1(c) having $minsup \geq 2$.

algorithm designs, and metrics definitions. First, most methods' objectives are to find clusters with predefined maximum distance allowed among subtrajectories and a maximum length in a cluster [1, 9, 10], while our work does not. Secondly, they do not aim for interactivity that the greedy algorithms [70] have more than quadratic complexity. Lastly, although the commonly used Fréchet distance [70] aligns the points between two lines like DTW, it only takes distances between *one* aligned pair of points between two lines, while time series distances take the *sum* of all aligned pair of points.

3 Extracting Similar Time Series Subsequences

Our approach frames the time series subsequence clustering into a two-step process: (1) transforming time series into **discrete symbolic sequences** and (2) mining **frequent patterns** from these sequences. Breaking down time series into discrete sequences is a widely-used approach for subsequence similarity search [27,40], treating the time series analysis as a sequence pattern mining problem (Background in Appendix A). For effective visualization, we address two key challenges: 1) constructing discrete subsequences that preserve visual similarity and 2) extracting sets of frequent patterns with visual compactness as the objective. Note that our method focuses on finding similar subsequences within fixed time intervals rather than across arbitrary time periods. While this constrains the search space, it enables the algorithmic efficiency needed for interactive exploration of large datasets, allowing our implementation to process 100,000 time series in seconds (Section 6).

3.1 Time Series Visual Symbolic Representation

Let $\mathscr{T}$ be a set of m real valued time series, where each series $T \in \mathscr{T}$

$$T = t_1, t_2, ..., t_n \quad \forall t_i \in \mathbb{R}, i \in I = \{1, \ldots, n\} \tag{1}$$

has length n. I represents the time intervals in which each T is defined. The proposed approach begins by decomposing the time series T into a sequence of contiguous time series segments (Figure 1(b))

$$T = s_1 \oplus s_2 \oplus \cdots \oplus s_{n'}, \tag{2}$$

where $\oplus$ is the concatenation operator and each s_i has size l. The value of l is defined by users based on the shortest possible intervals of the patterns they want to extract.

Given a subinterval of size l, we split the time interval I where the times series are defined into $n' = \frac{n}{l}$ subintervals I_i such that $I = I_1 \cup I_2 \cup \cdots \cup I_{n'}$. Our goal is to identify groups of similar time series segments within each subinterval I_i, so that each group of similar segments is represented by a symbol. This procedure makes it possible to represent a time series as a sequence of symbols, one per segment I_i, leading to a more compact and easily interpretable representation of the time series (Figure 1(c)). Specifically, let $\mathscr{T}_i$ be the set of time series segments from $\mathscr{T}$ in subinterval I_i. We can cluster the segments in $\mathscr{T}_i$ according to their similarity, assigning a symbol to each cluster. If we denote the clusters in I_i by $c_{i1}, c_{i2}, \ldots, c_{ik_i}$, where k_i is the number of clusters in I_i, we can represent time series T as $T' = c_{1j_1}, c_{2j_2}, \ldots, c_{n'j_{n'}}$ (Figure 1(c)), where c_{ij} is the symbol assigned to the jth cluster of interval I_i (here, we represent the cluster and the symbol assigned to it with the same notation). The symbol c_{ij} is chosen such that the segment $s_i \in T$ defined on I_i is contained in c_{ij} (Figure 1(c)).

Dynamic Time Warping as a Visual Similarity Metric. Choosing the distance metric that reflects visual similarity is essential to obtain good clusters. Despite the abundance of similarity measures available in the literature [20, 47], several studies (including crowdsourcing, user study, and data mining benchmarks) indicate that Dynamic Time Warping (DTW) [7] performs better on average than other measures both in terms of perception [23, 28, 42] and classification accuracy [20]. Unlike Euclidean distance (ED) that only considers the corresponding points in two time series, DTW allows for small shifts on the time axis to minimize the overall sum of distances. For example, although the two time series might have similar patterns (e.g., one peak), ED computes a much larger distance than DTW because it does not perform a shape matching alignment of points before computing the distances (Figure 11 in Appendix). Such an alignment invokes the Gestalt rule of similarity: humans perceive lines with similar slopes as the same group [76]. For these reasons, in our work, we use DTW to calculate the pairwise similarity of each time segment. Other distance measures that align the series like CDTW [11] could be used for the same purposes. Several strategies have been proposed to improve the performance of DTW, including approximation [61], CPU optimization [44] and GPU acceleration [15]. We evaluate their performance under different data sizes and hardware set ups in Section 6.

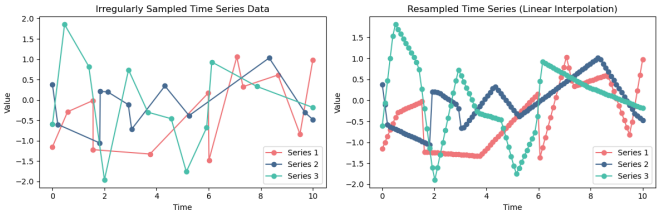

Fig. 3: Limitation on irregularly sampled time series (left) and mitigation by interpolating the series with a regular time grid (right).

Clustering Symbolic Representations. We use agglomerative hierarchical clustering with gap statistics [68] to automatically estimate the number of clusters. To facilitate effective visual analytics, we use a parameter α as clustering strength with Gap statistics to adjust the number of clusters in a $\hat{k}$:

$$\hat{k} = \text{smallest } k \text{ such that } Gap(k) \geq \frac{1}{\alpha} Gap(k+1)$$

Compared to other clustering methods, like k-means or DBSCAN [24], our approach does not need to be given the number of clusters or the best distance thresholds, which are hard to determine among a set of time intervals. Also, since users might want to visualize raw time series without any pre-processing such as normalization, the clustering strength parameter can provide a more flexible control to determine the outcome with human interactions, such as whether or not time series with vertical shifts should be grouped together.

Once the symbolic sequences are established, we can apply sequential pattern mining techniques to retrieve common time series subsequences from the dataset. Before explaining the optimization process and its motivation, we first introduce some nomenclature, mainly the concept of patterns and minimum support (*minsup*) and their implications on the optimization outcome.

3.2 Profiling Sequential Patterns

A *pattern* (i.e., sequential pattern) is a sequence of contiguous symbols and also a set of intersected time series subsequences represented by the symbolic sequences in our case. The length of a pattern is the number of such symbols in the sequence. For instance, in Figure 1, $c_{31}+c_{41}$ is a pattern of length 2. The *support* of a pattern p, denoted $sup(p)$, is defined as the number of time segments that are members of p. For example, support of 3 means that there are three segments that contain the pattern. The minimum support *minsup* is number of time series segments that a pattern must contain to be *frequent*. The *minsup* value can be set by users, based on the requirements of a specific application; intuitively, it specifies how frequent a pattern must be to be significant. The lower the *minsup* value, the greater the chance of obtaining longer frequent patterns, whereas larger *minsup* values produce shorter patterns with more similar segments. Using the example in Figure 1(c), the frequent patterns with $minsup = 2$ are illustrated in Table 1.

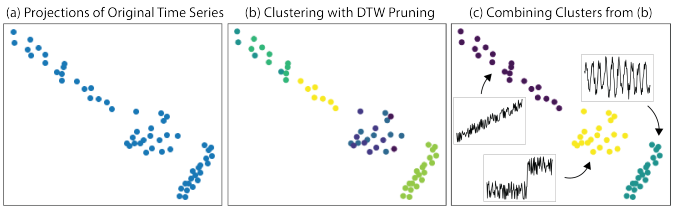

Fig. 4: Clustering time series with DTW distance directly (from (a) to (c)) is an expensive computation. To speed up, we first (b) coarsely partition the data with LSH, then we can sample from each partition to compute DTW clustering with a smaller subset of the data.

3.3 Computing Effective Groups from Sequential Patterns

While these patterns can be treated as subsequence clusters since the time segments in each pattern are similar to each other, directly using frequent patterns for visualization presents two main problems. First, the number of frequent patterns can grow drastically when the *minsup* is set to a small number because of the combinatorial explosion [14]. Even if we restrict the results to only include those who are minimal or closed [32], there might still be many patterns. Second, these patterns might overlap with each other, resulting in many repeated time segments displayed. Since our goal is to find subsequence clusters that best represent the trends in the dataset (Figure 1(d)), we leverage the patterns to find an effective set of time series partitions. We define g as an unique *subset* of time series belonging to p as the cluster to be visualized. Given a *minsup* value, our resulting set of clusters $\mathsf{G} = \{g_1, g_2, \cdots\}$ obeys the following properties:

$$\begin{gathered} g_i \subset p_x, g_i \nsubseteq p_y, x \neq y \\ g_i \cap g_j = \emptyset \text{ and } sup(g_i) \geq minsup \end{gathered} \tag{3}$$

for any clusters $g_i, g_j \in \mathsf{G}$.

As a result, each possible set of clusters G is a partition of the original time series data. Since a "concise" visual representation of the time series should display most data with minimum number of sets (i.e., charts), we could define the set $\hat{\mathsf{G}}$ as the one that has the least number of clusters from the cluster sets G_{max} that contains the most number of time series data from G:

$$\hat{\mathsf{G}} = \underset{\mathsf{G} \in \mathsf{G}_{max}}{\arg\min} |\mathsf{G}|, \mathsf{G}_{max} = \left\{ \underset{\mathsf{G} \in \mathbf{G}}{\arg\max} \sum_{g \in \mathsf{G}} |g| \right\} \tag{4}$$

where $|\cdot|$ accounts for the number of elements in the set.

Intuitively, we use frequent patterns to identify significant and similar time series subsequences of arbitrary lengths and discard those without enough support. We further minimize the number of groups needed to partition the similar contiguous subsequences. Thus, we provide a concise overview of unique temporal behaviors along the time period, dramatically reducing the visual complexity and allowing users to quickly grasp the dominant trends within a large number of time series.

3.4 Method Limitations and Mitigation

While TIVY effectively addresses challenges in time series visualization, some limitations must be considered for practical applications.

1. *Irregularly sampled time series.* DTW clustering requires regularly sampled time series, making irregularly sampled data incompatible with our approach (Figure 3(a)). A simple strategy to standardize the sampling rate is to set up a uniform grid and interpolate the values of each series at the grid points (Figure 3(b)).
2. *Window Size Sensitivity.* If l is too small, it discards the shifts beyond the sub-interval, and if l is too large, subsequence clusters with short durations are not grouped together. For time series data with short patterns and long shifts, subsequence clusters without temporal constraints will serve better outcomes. To mitigate this, as discussed in Section 4, we could implement adaptive window sizing, though this increases computational complexity.

ALGORITHM 1: TIVY (**T**ime Series **V**isual Summar**y**)

Input : T – a set of time series as a $m \times n$ matrix
window_size – window size
α – clustering strength
minsup – minimum support
Output : $\hat{\mathsf{G}}$ – A list of subsequence groups

```
P ← construct_sequences(T, window_size, α)      /* Sec. 3.1 */
D ← {}; prefix_scan(P,minsup,null,D)            /* Sec. 3.2 */
Ĝ ← extract_groups(D,minsup)  /* Patterns from Sec. 3.3 */
```

ALGORITHM 2: construct_sequences

Input : T – a set of time series
w – window size
α – clustering strength
Output : P – A set of symbolic sequences

```
m ← T.shape[0]                          /* number of time series */
n ← T.shape[1]                  /* length of each time series */
S ← array_split(T,w)                    /* split T into segments */
P ← empty_matrix(m, n/w)
for i=0; i < n / w; i++ do
    s ← S[i]
    cluster_labels ← []
    for j=0; j < m; j++ do
        cluster_labels[j] ← LSH(s[j])  /*  coarse clustering  */
    end
    sample ← []   /*  sampling from each LSH cluster  */
    for label in unique(cluster_labels) do
        sample.push(random_select(s[cluster_labels == label,:]))
    end
    sample_labels ←
      hierarchical_clustering(sample, metric = 'DTW', strength = α)
    /*  assign final labels to the coarse clusters  */
    for label, sample_label in unique(cluster_labels), sample_labels do
        cluster_labels[cluster_labels == label] ← sample_label
    end
    P[:,i] ← cluster_labels
end
```

4 THE TIVY ALGORITHM

The TIVY (**T**ime Series **V**isual Summar**y**) algorithm constructs the symbolic representations and derives a grouping of time series subsequences (see Algorithm 1). The algorithm consists of three key steps: (1) clustering time series at each sub-interval, (2) profiling the supports of sequential patterns, and (3) computing the groupings of the sequential patterns. Since it is heuristics-driven, we will discuss the effects on the parameters and later demonstrate the effectiveness and parameters' influences on visualizing synthetic and two real datasets in Section 6.

Clustering Time Series at each Sub-interval (Section 3.1). As illustrated in Figure 1(c), to transform a real-valued time series to a discrete symbolic representation, we need to cluster segments using the DTW metric. However, a straightforward clustering with DTW is infeasible even for moderately-sized data, since the distance computation has a time and space complexity of $O(m^2)$, where m is the length of the time series. While m could be small if the window size l is small, the bottleneck comes from the hierarchical clustering that automatically choose the optimal number of clusters, since it requires a pairwise distance matrix resulting in $O(n^2)$ time complexity, where n is the number of time series. Thus, the total time complexity to construct symbolic sequences from time series is $O(m^2n^2)$. Even though this approach leads to an effective perceptual result, the lack of scalability hinders its usage for interactive exploration.

To address this limitation, we propose a more efficient clustering methods. The method splits the clustering process into two stages. It first coarsely groups the time series using Locality-Sensitive Hashing (LSH) [18]. Basically, we hash each time series $t \in T$ into an integer bucket using the following hash function:

$$h(t) = \left\lfloor \frac{t \cdot x + b}{w} \right\rfloor \tag{5}$$

ALGORITHM 3: prefix_scan

```
Input  : P̄ – an m̄ × n̄ submatrix containing m̄ symbolic sequences with length n̄
         minsup – minimum support
         prefix – prefix sequence of the submatrix
         D – A reference of dictionary P → { Time Segments }
1  if P̄ is empty then
2      return
3  end
4  first_col ← P̄[1 : m, 1]      /* clusters on the current level */
5  for symbol, idx in unique(first_col) do
6      if Size(idx)<minsup then
           /* skip all support calculations on patterns
              with prefix idx                                  */
7          continue
8      end
9      D[prefix + symbol] ← D[prefix][idx, 2 : n̄]
10     prefix_scan(P̄[idx, 2 :], minsup, prefix + symbol, D)
11 end
```

where x is a random vector with each element sampled from Normal distribution $x_i \sim N(0,1)$, w is a width representing the quantization bucket, and b is a random variable sampled from the Uniform Distribution $b \sim unif[0,w]$. The hashing function ensures that if two time series are similar in terms of Euclidean distance, which is the upper bound of DTW distance, the probability of being assigned to the same bucket (i.e. $P(h(t_i) = h(t_j))$) will be high. Such grouping only requires a linear scan on the series and each scan only contains a hashing on the values of the series (i.e. $O(mn)$ time complexity). Then, instead of running the DTW clustering on the whole dataset, we can run it with one or more samples from each hash bucket, substantially reducing the number of DTW computations, while still leveraging the benefits of DTW to group visually similar series. The outline of the algorithm `construct_sequences` (Algorithm 2 and Figure 4) is as follows:

1. The clustering starts on segments in each time interval (line 5-6).
2. We first run LSH (line 8-10) to coarsely cluster the time series.
3. To further combine these coarse clusters, we run hierarchical clustering with DTW distances (line 15) with only *one* random sample from each coarse cluster (line 12-14). α is used to determine the optimal number of clusters from Gap statistics.
4. We propagate the cluster labels obtained from each sample to the rest of its coarse cluster members (line 18-20) to return the final result (Figure 4(c)).

Profiling the supports of sequential patterns (Section 3.2). Unlike traditional sequence mining algorithms like PrefixSpan and others [25, 55, 73] that require expensive pattern projections as the bottleneck, our symbolic sequence T' is a special case that contains ordered cluster labels that never appear in more than one time interval. For example, the time series represented in $c_{12} + c_{21} + c_{31}$ are subsets from the ones represented in $c_{12} + c_{21}$ Thus, we could efficiently extract patterns by grouping the time series symbol-by-symbol from the beginning of the symbolic sequences to the end (Algorithm 3). It handles one time interval in each recursion (line 4). The time series indices (as rows in the input) are grouped together to the next recursion if they have the same symbol at the current interval (as column), or be split into different recursions if not (line 5). These indices will also be stored in the dictionary if their patterns' supports are not smaller than *minsup*; else, the recursion will stop (line 6-9). Such an approach allows the pattern mining procedure to be completed within seconds, as opposed to minutes in other pattern mining approaches (Section 6.2).

Extracting Effective Groupings via Greedy Search (Section 3.3). Since the number of frequent patterns is exponential to the number of symbols [31], and the number of possible sets of groupings G is the binomial sums (i.e., $O(2^n)$), finding the set of effective groupings is hard. Thus, the proposed algorithm follows a greedy approach that evaluates the frequent patterns one by one, with heuristics that prioritize long frequent patterns. The intuition is that if long subsequence groups are chosen, it is likely to reduce shorter patterns, which requires more groupings to cover the same number of time intervals. The outline of the algorithm `extract_groups` (Algorithm 4) is as follows:

ALGORITHM 4: extract_groups

```
Input  : D: A dictionary of P → {Time Segments}
         minsup: Minimum support
Output : Ĝ – A list of subsequence groups
1  Ĝ ← []
2  while D ≠ ∅ do
3      delete D[p] ∀p in D if Support(D[p]) < minsup
       /* Prioritize longest frequent patterns          */
4      Candidate← {p | p in D} , where length(p) = maximum length of patterns
         in D
5      while Candidates ≠ ∅ do
6          p_candidate ← random_select(Candidates)
7          overlap_patterns ← []
8          for p in Candidates \ p_candidate do
9              if Support(D[p] − D[p_candidate]) < minsup then
10                 overlap_patterns.push(p)
11             end
12         end
           /* Maximize retrieval on longest patterns    */
13         if Size(overlap_patterns)> 1 then
14             delete D[p_candidate]
15             Candidates.remove(p_candidate)
16             continue
17         end
18         Ĝ.push(p_candidate)
19         for p in D do
20             D[pattern] ← D[pattern] − D[p_candidate]
21         end
22         for p in Candidates do
23             if Support(D[p]) < minsup then
24                 Candidates.remove(p)
25             end
26         end
27     end
28 end
```

1. We initialize the grouping candidates and the contained series as the pattern profile obtained from `extract_groups` as D. We iteratively extract (line 19) and prune (line 3) the candidates until the candidate list becomes empty (line 2).
2. In each iteration, the algorithm first shortlists the candidates with the longest patterns (*Candidates*) (lines 4-5) and tries to extract as many candidates from this set as possible (lines 6-31).
3. Each shortlisted candidate is evaluated one by one in a random order (line 7). During the evaluation, we calculate the reduction of item in *Candidates* if the candidate is selected (line 10). A reduction happens only when (1) there are series removed due to the overlapping of both series and time interval between the comparisons, and; (2) the removal leads to the support of the item being below *minsup*.
4. If the reduction is greater than 1, it means taking this candidate will not maximize the retrieval of the set and we should remove it (line 14-18). Otherwise, we export it to the final result (line 19) and remove the overlapped series in D (line 20-22) and update the shortlisted candidates correspondingly (line 23-27).

Adaptive Sub-Interval Sizes and Grid Search. Fixing the sub-interval size l (Equation 2) makes the computation efficient, but it can affect the outcome (Section 3.4). Therefore, instead of decomposing each segment s_i with size l, we enable each segment's size to be a multiple of l, and then generate all the valid contiguous partitions. Among these partitions, we run a grid search approach to identify the partition that provides the best subsequence clusters from Algorithm 1. To evaluate cluster quality, we propose a metric inspired by the Silhouette index [57] that balances cluster separation and cohesion: $Q = \frac{Q_{inter}}{Q_{intra}}$. Q_{inter} is the inter-cluster distance (i.e. sum of distances among the medoids in each cluster) and Q_{intra} is the intra-cluster distance (i.e. sum of distances between the medoids and the series in each's clusters). The distances takes DTW distances and also the overlap of time intervals for inter-cluster and the length of series for intra-cluster into account. Intuitively, higher values indicate better clustering with well-separated, cohesive groups. This allows us to capture subsequence

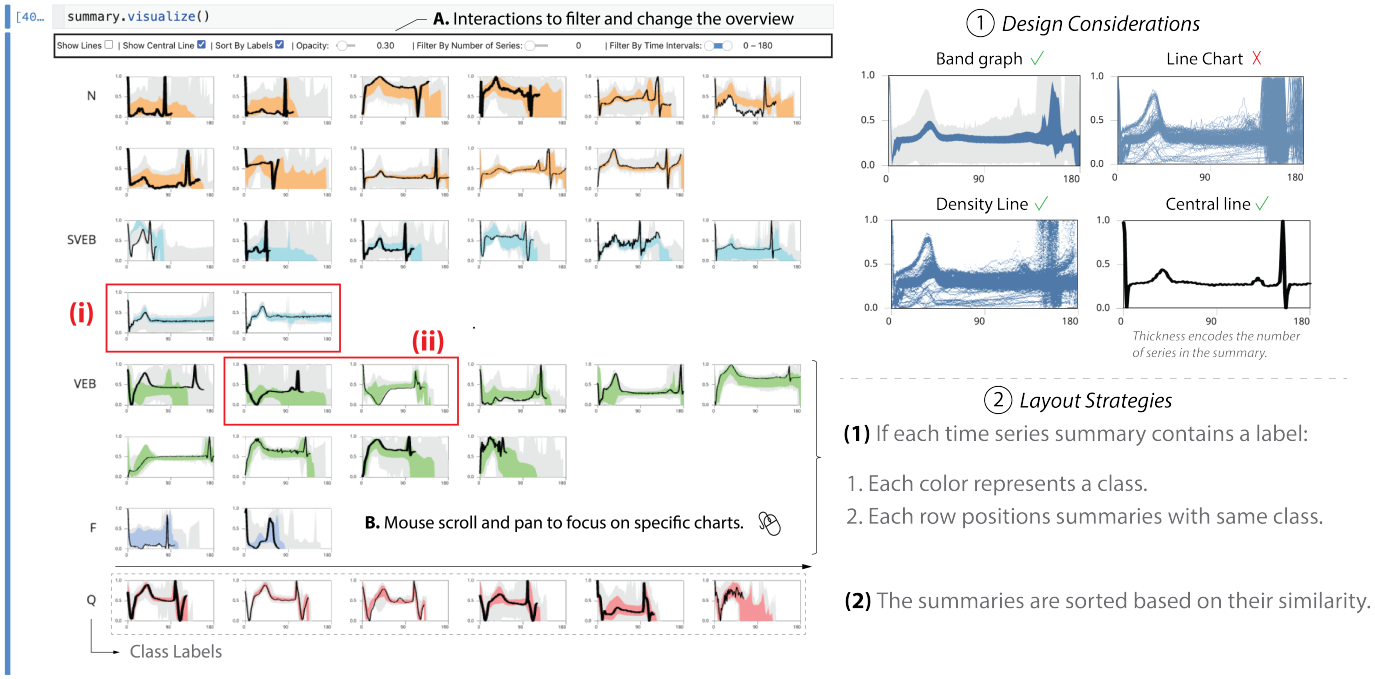

Fig. 5: Time series summary for 100,000 ECG signals in a WebGL based widget in Jupyter notebook. Summaries of different shapes are visualized in different small multiples for categorizing different waveforms to help proper diagnosis and treatment (i, ii). ① Visual Design: Each chart is shown with a bold line encoding the representation and size of the summary. The time series inside can be encoded as a band graph or density line chart. ② Layout: Colors and positions encode the labels of the summary, and the charts are sorted based on their pairwise similarity.

clusters with varying sub-interval sizes more easily, as demonstrated in Figure 7. However, the drawback is that there will be $2^{\frac{n}{l}-1}$ times of running the whole algorithm. For example, a $l = \frac{n}{10}$ in Figure 7 results in 512 combinations that makes the computation finished in around 8 minutes, provided that we use dynamic programming to save the clustering results on the same time intervals among the partitions.

Tradeoff between Speed and Quality. While our pipeline provides various speed ups, exploiting the parameters for maximal speed might affect the quality, leading to original problems of time series visualization like visual clutter in a chart or too many charts. We now describe settings that will incur such a tradeoff.

1. *Increase bucket size in LSH.* Increasing the width w in Equation 5 will make series more prone to fall into the same bucket that reduces the number of series undergoing the DTW clustering routine (i.e., faster clustering), but also increases the variances of series inside the buckets, resulting in more visual clutters. Conversely, a smaller w will align the results to DTW clustering with slower computation. More samples might improve cluster assignment but could not separate similar series inside the same bucket.
2. *Increase minsup.* As it increases minimum number of series needed to be defined as a pattern in the final output, the cluster candidates and the search space for the groupings decrease. However, if there are many clusters left for the final output, it also implies that the output misses a lot of original data.
3. *Decrease window size l.* Small l speeds up the DTW clustering as it is quadratic to l, but might produce the problems in Section 3.4.

5 Visualizing Time Series At Scale

We propose an efficient open-source implementation for scalable time series visualization using our algorithm. Efficiency means both effective visual encoding and computationally accessible rendering of thousands of time points at a decent frame rate on a laptop. We outline the analytical tasks supported by time series summaries and present a WebGL-based implementation as a reusable Jupyter notebook widget.

5.1 Tasks and Requirements

A visual summary of time series effectively addresses the removal of visual clutters and optimization of layout compactness. We reference the tasks from Aigner *et al.* [2], which describes a set of tasks involved in the time series data analysis. The overall tasks include classification, clustering, search and retrieval, pattern discovery, and prediction. Our visual summary falls into the category of clustering. The book references the classical paper regarding time series clustering by Van Wijk and Van Selow [72]. Also, a recent paper related to visualizing trends in time series [71] summarizes a set of clustering tasks from the book. Therefore, we base our design considerations on the clustering tasks from these three references. Furthermore, we include the system design requirements to maximize the real-world impact of our software. The tasks are summarized below. **T.1** and **T.2** focus on the tasks gathered from the surveyed time series clustering analysis and **T.3** and **T.4** focus on the interactions involved when visualizing clusters with additional attributes. Last, **T.5** is related to the requirement for building a system from the algorithm and focuses on the scalability and accessibility of the system.

- **T.1** Understand subsequences' behavior in large time series including:
  - a. The support of each cluster.
  - b. The time interval and range of each cluster.
- **T.2** Understand the distribution of each cluster including:
  - a. The quality of each cluster.
  - b. The shape differences among clusters.
- **T.3** Filter and zoom for specific clusters to:
  - a. Highlight clusters by their temporal values.
  - b. Inspect individual time series within a cluster.
- **T.4** Compare clusters with different attributes.
- **T.5** Integrate to the real working environment.
  - a. Implement as an interactive widget inside the computation notebook with low hardware requirement.
  - b. Render visualization with low latency and scale to real-world datasets.

5.2 Visualizing Time Series Summary

Designs to visualize multiple time series have been thoroughly studied [36]. However, when it comes to high volumes of temporal information, the main challenge is the technical scalability to render millions of data points representing the lines while performing various interactions seamlessly. We need to consider different tradeoffs to balance the

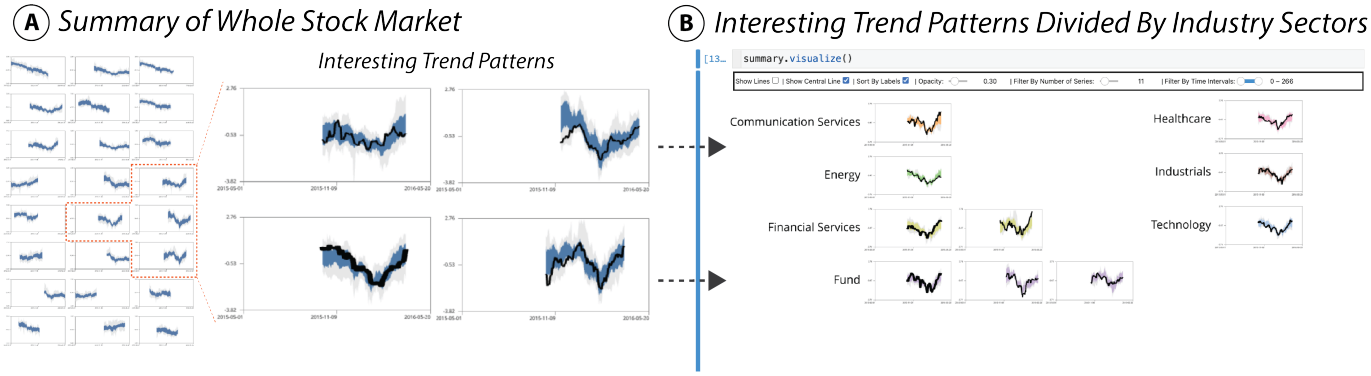

Fig. 6: A time series visual summary is a set of time series subsequence clusters of varying lengths that groups visually similar time segments together. In a stock market use case, we run TIVY to explore visual summaries of 4,470 stock market time series in 2015-16 for portfolio construction. Ⓐ The algorithm creates subsequence clusters that cover main trends in the dataset. Ⓑ After identifying different trends in the market, a common "v" shape pattern is shown among stocks in the first three months of 2016 and splits the subsequence clusters by sectors to observe which sectors contain this pattern.

software efficiency and accessibility. Thus, we now propose our time series visualization system (Figure 5), and discuss the visual encodings that consider both visual clarity and rendering efficiency.

5.2.1 Visual Encodings

Since TIVY allows a holistic high-level summary of all subsequence clusters in a time series dataset, each group could be visualized with a precise shape (**T.2**) and displayed with the main statistics (**T.1**) such that users can quickly acquire an overview of multiple clusters plotted on the same screen. Moreover, it becomes feasible to use aggregated visual encodings and superposition visualization in Figure 2(b) and (c) to present a more significant number of time series inside the cluster. Overall, our design considerations include the usage of band graph, line chart, density line chart, and a central line to encode the time series inside each summary (Figure 5①).

Band graph. We provide two bands (i.e., range + 90% quantile) to visualize both the extrema and tighter bounds of the time series inside the cluster to balance between the visual clarity and accuracy. The advantage is that since the time series within each chart is homogenous, the band can accurately depict the trends and limit the amount of noise (**T.2**). Also, in terms of rendering, we can use two polygons to render two bands, which are much simpler than rendering millions of points (**T.5**). We acknowledge the fact that further user study is needed to evaluate the tradeoffs between clarity and uncertainty when choosing the number of quantiles in uncertainty visualization.

Density line chart. To reveal original data points in the chart, we can use line charts to visualize the time series. However, direct input of data points to the rendering pipeline will easily hinder the system's interactive performance. When users pan and zoom the whole graph, each data point in the time series has to undergo several linear transformations to define its new location in the screen. For example, 100,000 time series with a cardinality of 180 will require 18 million transformation operations. Some platforms might not support rendering lines with widths that we need to triangulate the lines as polygons that results in more vertices. For a common laptop and latency in milliseconds allowed only, users will easily experience lagging during the interactions (**T.5**).

To address the increasing data points on rendering the line charts, we need to avoid having the rendering complexity be linearly proportional to the number of data points. One way to do so is to aggregate the input time series to 2-D density maps, which are bounded to a fixed number of bins. By having a slight overhead to compute the density map, we can instead input much smaller meshes for real-time interactions. The color opacity encodes the density such that we can inspect the distribution of temporal values inside the cluster (**T.2**).

Center line. To improve the reflection of the main statistics in the aggregated plot (i.e., band graph), we use the medoid [67] in the cluster as the main shape, which is an sample t_m inside the cluster $g = \{t_1, t_2, \cdots, t_n\}$ that has the minimum sum of pairwise distances with other lines:

$$t_m = \arg\min_{t \in g} \sum_{i=1}^{n} d(t, t_i) \tag{6}$$

Choosing a line instead of using time-invariant averaging methods such as soft-DTW [15] or DTW Barycenter Averaging [56] avoids computationally expensive algorithms ($O(N \cdot T^3 + N^2 \cdot T^2)$) for useful interactive analysis. We also encode the line width with the number of time series inside the summary to better estimate cluster size (**T.1**).

5.2.2 Layout Strategies

To allocate the time series in a compact layout, we position each time series chart as small multiples that fills the entire canvas from left to right and top to down (Figure 5②). Besides, when users supply an attribute to each summary (e.g., class labels), we can encode the summaries with colors and group the same ones by vertical positions (Figure 5(i)) to present clearer comparisons (**T.4**). Also, summaries might share similar shapes among each other as well. Thus, we can sort the summaries by their similarity. It can be done by first sampling one series from each summary, then build a hierarchical cluster (i.e., linkage) with the samples and obtain the order from the leaves in the hierarchy (Figure 5(ii)).

5.2.3 Interactions

Our widget provides the interactions to facilitate the exploration of time series summaries (Figure 5A).

Pan and zoom. Since our rendering pipeline provides great computation efficiency on the linear transformation on the graphics, users can easily pan and zoom the whole canvas to focus on a subset of time summaries in real-time (**T.2**).

Filtering. Since each summary contains statistics such as the number of time series and time intervals, our system provides two sliders that filter the summaries based on that. These allow the users to explore the important trends effectively within a specific time interval (**T.1**).

Toggles for different visual encodings and layouts. Our system provides different toggles to reveal different visual encodings on demand. Users can either display the summaries as band graphs or density lines and select whether they want the central line or not. Furthermore, users can select whether they want the summaries to be separated by the attributes or not (**T.4**).

5.2.4 Implementation

Our algorithm and system are implemented with Python and can be used in Jupyter notebook and Jupyter Lab. WebGL is required to render the time series visualization. We use NumPy for most of our

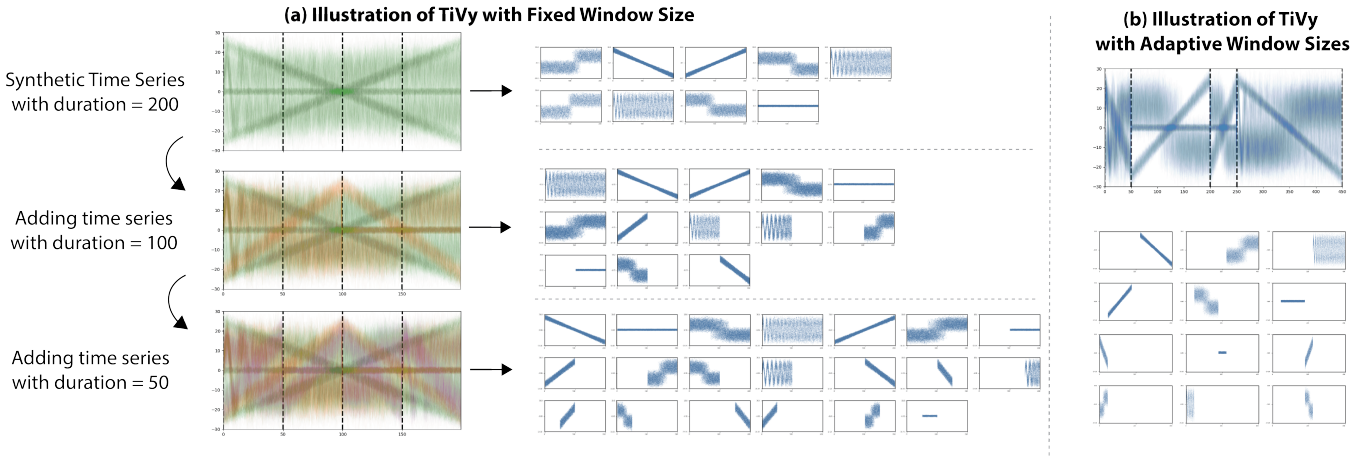

Fig. 7: Evaluation of TIVY's visual quality using synthetic dataset composing of size main trends with different durations. Our algorithm successfully extracts complex patterns hidden in different parts of the dataset into accurate subsequence clusters, and supports adaptive sub-interval sizes (dashed lines on irregular intervals on the top right chart) when needed.

computations in TIVY and VisPy for rendering the meshes representing different visuals in the time series views.

To reduce the latencies during visualization interactions, such as filtering or switching the charts, we first compute the vertices of all possible visuals (i.e., polygons of band graphs, centerlines, and density lines) and store them in a buffer object. Then, when each visualization is selected and shown, we can directly import the locations to the rendering pipeline without creating the vertexes on the fly. Our visualization interface is available at `https://github.com/GromitC/Jupyter-Time-Series-Visualization`.

6 EVALUATION

6.1 Datasets and Apparatus

All of our experiments are conducted in a MacBook Pro with 2.4 GHz 8-Core Intel Core i9 CPUs and 32GB RAM, except the one requiring GPUs (we use 8 A100-40GB). We use the following datasets for our experiments:

Synthetic Data. We design a synthetic dataset with different shapes in multiple time intervals. The whole dataset contains six classes of shapes: cyclic; normal; increasing; decreasing; upward shift; and downward shift. In each class, the time series have some deviations in temporal alignment and amplitudes but plotting all of them within one plot would not cause too much perception differences. We also generate these data with different durations and combine them together to form the datasets shown in Figure 7. The main purpose of using synthetic data is to evaluate the expected behavior of our algorithm with known data characteristics.

Stock Time Series. The dataset contains 4,470 company daily adjusted stock prices in NASDAQ between the year of 2015-16. The industry sector for each stock price are also provided.

ECG dataset. We use the MIT-BIH Arrhythmia ECG dataset [1] that records 100,000 patients' heart beat signals. Each signal has a cardinality of around 200 and we trim the trailing zeros for each signal. The heartbeats are annotated by five groups of heart conditions.

6.2 Quantitative Evaluation

In this section, we report the effects of visual outcomes by different design choices in the algorithm, and the performances with different scalability and alternatives.

Visual Quality. To verify that our algorithm can extract the time series patterns with different sizes. We present the visual summarization results of the synthetic data under different settings in Figure 7(A), as well as the patterns extracted from the real-world datasets in Figure 5 and Figure 6Ⓐ. First, for the synthetic data, we run our algorithm on three datasets with different combinations of durations of time series patterns. For each unique pattern and combination, there exists 100 time series. We fix the clustering strength to 1, minimum support to 50, 30 LSH with $w = 1$, and select the result with recall greater than 0.95 among time window sizes of {25, 50, 100, 200} for three datasets. We also create a dataset that contains shapes with different lengths at different time intervals to evaluate the adaptive window sizing capability (Figure 7(B)). Figure 7 shows that TIVY is able to separate the patterns regardless of different shapes and durations existed in the dataset. We noticed that adding series with different shapes makes grouping similar shapes more easily. We hypothesize that increasing shape distinctiveness is beneficial for the gap statistics approach, as suggested by related evidence [74]. For the patterns extracted for the real-world datasets, we will present them later in Section 6.4.

[1] `https://www.physionet.org/content/mitdb/1.0.0/`

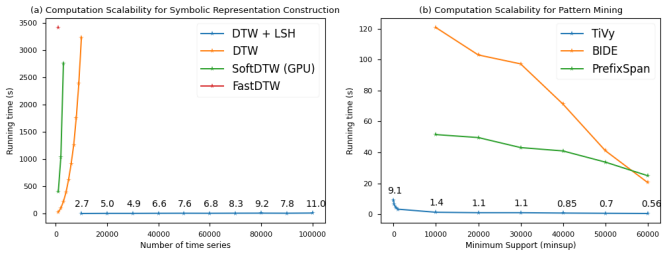

Fig. 8: Run time (> 1 hr omitted) improvement with LSH and indexing.

Computation Scalability. We report the run time using the 100,000 ECG data series in Figure 8. To highlight the effect of LSH in reducing the quadratic time complexity of DTW distance metric and hierarchical clustering (Section 3.1), we sample the ECG data and compare the run time between the algorithms with and without the optimization [44], and other variants on DTW (i.e. SoftDTW on GPUs [15] and FastDTW [61]). We fix our parameters on time windows and clustering strengths. In Figure 8(a), it shows that our LSH optimization is able to speed up the process from hours to within a couple of seconds, and cpu optimization for DTW plays the most important role on the efficiency. For the discrete pattern mining process (Section 4), we compare with general purpose pattern mining approach [73]. In Figure 8(a), it shows that given our unique pattern structures, we can speed up the computations from long time to seconds. For the rendering pipeline, we report the preparation time of the buffers input to the system and the interaction latencies during the filter, pan, and zoom operations (Figure 9). We can see that by addressing the linear overhead of preparing the visual buffers to the system which is an one-off computation only (Figure 9(a)), we can achieve a seamless exploration of time series data in the interactive interface with good FPS (Figure 9(b)).

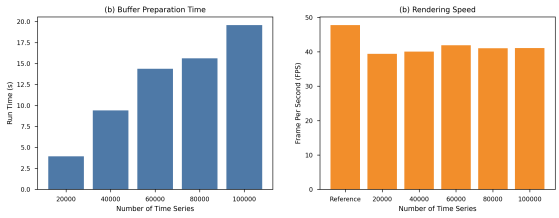

Fig. 9: Rendering preparation time and FPS.

6.3 Qualitative Evaluation

Our goals in this section are to visually illustrate the qualitative impacts of various optimizations in the pipeline to the final results to help readers understand how to spot unsatisfactory results on a dataset.

Over Clustering with long *w*. Our LSH buckets series with similar distances, where the range is defined by w (Equation 5). Thus, if w is too large (e.g. 10× in Section 4), then different shapes will be shown in one chart visibly. (Figure 10(a)). More variations of w and number of samples are shown in Figure 13 in the Appendix. Overall, w plays a more critical role than sampling as discussed in Section 4.

Redundant Plots with small window size *l*. If l is small, there would be a lot of charts with similar shapes (Figure 10(b)) since it limits the alignment capability for DTW clustering (Section 3.4).

Under Plotting with high *minsup*. If the *minsup* is too high, lots of data points will be abandoned in the final output due to the high number of series required to be in the plot, resulting in a display with many missing shapes (Figure 10(c)).

6.4 Use Cases

We present two usage scenarios to demonstrate the effectiveness of TIVY. First, we show our approach can summarize meaningful patterns for correct categorization of the ECG data. Second, we apply our technique to help understand important trends in the financial stock market. We fix the time window size to one-tenth of the total durations, clustering strength to 1, and minimum support to 50. They are chosen since the summarization outcomes for both visualization capture more than 95% data points in the datasets and produce visible trends with few clutters.

6.4.1 ECG Signal Classification

We demonstrate whether TIVY is able to visualize large amounts of time series effectively. We use the MIT-BIH Arrhythmia ECG dataset [2]. ECG is widely used in medical practices to monitor cardiac health. Understanding the waveforms and attributing them to the correct categorization is important for proper diagnosis and treatment. Each row in the data consists of heartbeat signals annotated by at least two cardiologists. The annotations are mapped to five groups suggested by Association for the Advancement of Medical Instrumentation (AAMI): Normal (N), Supraventricular Ectopic Beat (SVEB), Ventricular Ectopic Beat (VEB), Fusion Beat (F) and Unknown Beat (Q). We removed the trailing zeros since they represent the end of the beat.

Exploring patterns among 100,000 time series. The overall summarization result is shown in Figure 5. There are some interesting patterns shown. For example, the patterns in Figure 5(i) are long flat lines of dropped beats, which are clear characteristics of junctional escape beat belonging to the SVEB group. Also, Figure 5(ii) shows strong contraction beats with a long pauses afterwards, which are symptoms related to premature ventricular contractions. These illustrate that TIVY successfully extracts visually meaningful patterns among 100,000 ECG signals, which corroborates the verification from two independent cardiologists.

6.4.2 Financial Time Series Analysis

The second use case involved the use of TIVY on stock market data to construct a portfolio. The dataset contains 4,470 company daily stock price series between the year of 2015-16. The goal is to study the trends occurred in the financial market to construct a portfolio that balances the risks in different situations. The time series are normalized with zero mean and unit variance to calibrate the trends with different numerical prices. We use TIVY to explore and select stocks through understanding different visual behavior of the stock time series.

[2] https://www.physionet.org/content/mitdb/1.0.0/

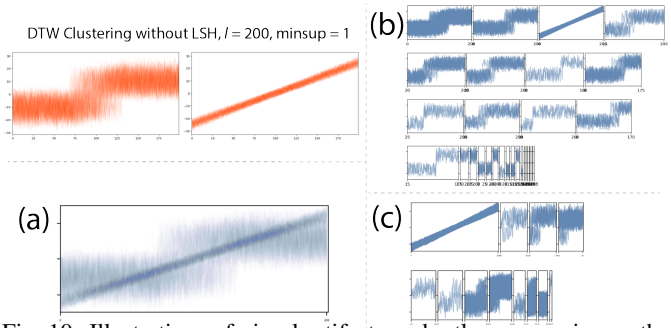

Fig. 10: Illustrations of visual artifacts under three scenarios on the synthetic data (orange): (a) long w; (b) high *minsup*; (c) small l.

Observing overall trends. To begin with, the algorithm computes a visual summary that shows 23 trends that, in total, cover most of the time series data (Figure 6Ⓐ). By inspecting the shapes of the trends, most companies prices' were decreasing throughout the period. However, some increasing trends were involved as well as the ones with zig-zag shapes. As the zig-zagging stocks are the ones that are being actively bought and sold (i.e., a price war). For risk-averse (i.e., afraid of risks) situations, they were not recommended for purchases.

Focusing on selected trends. Next, users might want to know if there are any more risky trends, so the time series subsequence clusters are filtered with the "v" shapes (Figure 6(b)). The visualization reveals that this pattern happens in early 2016. Since the trend exists in many such companies, there must be a global economy issue that affects the whole market. For portfolio managers, they could be aware that actively trading the stocks impose a higher risk in this situation.

Exploring industry sectors. The portfolio manager may try to know if there are any relationships between the trend and the sectors, so he may further split the clusters by the sector. The result shows that the trends mainly happen in four sectors represented by four thick lines: Industrial, Technology, Healthcare, and mutual funds (non-sector) (Figure 6(b)). As a result, he or she can research passive trading strategies for these four categories of companies. Overall, the algorithm provides a visual evidence to explore the stock trends freely and leverage the system to perform multi-model data analysis.

7 Conclusion and Future Work

We presented TiVy, a novel approach to time series visualization that addresses the trade-off between scalability and visual clarity through selective superposition based on visual similarity. By combining LSH-accelerated DTW clustering with specialized sequential pattern mining, we achieve 1000× performance improvement over straightforward DTW clustering while maintaining visual quality, enabling interactive exploration of datasets with 100,000+ time series. Our experimental evaluation demonstrates both the technical performance and practical utility of the approach. There are promising directions we would like to pursue in future work: 1) Extend TiVy to multivariate time series to enable analysis of complex phenomena with multiple interdependent variables; 2) Support interactive refinement like sketching, allowing domain experts to guide pattern discovery; 3) Explore modern vision models as replacements for the DTW clustering pipeline, learning visual similarity directly from time series projections and potentially attaining greater scalability; 4) Apply our techniques to other domains.

Acknowledgments

This work is part of Chan's PhD dissertation [12]. Supported by EU Horizon projects TwinODIS (101160009), DataGEMS (101188416), and by *ΥΠΑΙΘΑ* & NextGenerationEU project HARSH (*ΥΠ*3*TA* – 0560901); partially supported by Fapesp (#2022/09091-8) and CNPq (#307184/2021-8) funding agencies.

REFERENCES

[1] P. K. Agarwal, K. Fox, K. Munagala, A. Nath, J. Pan, and E. Taylor. Subtrajectory clustering: Models and algorithms. In *Proc. PODS*, PODS '18, p. 75–87. ACM, New York, NY, USA, 2018. doi: 10.1145/3196959.3196972

[2] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Springer Science & Business Media, 2011. doi: 10.1007/978-1-4471-7527-8

[3] G. Al-Naymat, S. Chawla, and J. Taheri. Sparsedtw: A novel approach to speed up dynamic time warping. In *Proc. Australasian Data Mining Conference*, pp. 117–127, 2009. doi: 10.5555/2449360.2449384

[4] P. André, M. L. Wilson, A. Russell, D. A. Smith, A. Owens, et al. Continuum: Designing timelines for hierarchies, relationships and scale. In *Proc. UIST*, pp. 101–110. ACM, 2007. doi: 10.1145/1294211.1294229

[5] B. Bach, P. Dragicevic, D. Archambault, C. Hurter, and S. Carpendale. A review of temporal data visualizations based on space-time cube operations. *Comput. Graphics Forum*, 33(3):61–81, 2014. doi: 10.1111/cgf.12365

[6] J. P. Bello, C. Silva, O. Nov, R. L. Dubois, A. Arora, J. Salamon, C. Mydlarz, and H. Doraiswamy. Sonyc: A system for monitoring, analyzing, and mitigating urban noise pollution. *Commun. ACM*, 62(2):68–77, January 2019. doi: 10.1145/3224204

[7] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAIWS*, vol. 10, pp. 359–370. Seattle, WA, 1994. doi: 10.5555/3000850.3000887

[8] D. Braun, R. Borgo, M. Sondag, and T. von Landesberger. Reclaiming the horizon: Novel visualization designs for time-series data with large value ranges. *IEEE Trans. Vis. Comput. Graph.*, 30(1):1161–1171, 2024. doi: 10.1109/TVCG.2023.3326576

[9] F. Brüning, H. Akitaya, E. Chambers, and A. Driemel. Subtrajectory clustering: Finding set covers for set systems of subcurves. *Computing in Geometry and Topology*, 2(1):1:1–1:48, Feb. 2023. doi: 10.57717/cgt.v2i1.7

[10] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. In *Proc. ISAAC*, pp. 644–655. Springer, 2008. doi: 10.1007/978-3-540-92182-0_57

[11] K. Buchin, A. Nusser, and S. Wong. Computing continuous dynamic time warping of time series in polynomial time. In *Proc. SoCG*, pp. 22:1–22:16. Schloss Dagstuhl, 2022. doi: 10.4230/LIPIcs.SoCG.2022.22

[12] G. Y.-Y. Chan. *Data Summaries for Scalable Visual Analysis*. PhD thesis, New York University, 2021.

[13] H. Chen, A. D. Beachnau, P. Thomas, P. Maneriker, J. Kimball, and R. A. Rossi. Live-its: Lsh-based interactive visualization explorer for large-scale incomplete time series. In *Proc. BigData*, pp. 840–849. IEEE, 2024. doi: 10.1109/BigData62323.2024.10825096

[14] J. Cheng, Y. Ke, and W. Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowl. Inf. Syst.*, 16(1):1–27, 2008. doi: 10.1007/s10115-007-0092-4

[15] M. Cuturi and M. Blondel. Soft-dtw: A differentiable loss function for time-series. In *Proc. ICML*, pp. 894–903. JMLR. org, 2017.

[16] T. N. Dang, A. Anand, and L. Wilkinson. Timeseer: Scagnostics for high-dimensional time series. *IEEE Trans. Vis. Comput. Graph.*, 19(3):470–483, 2013. doi: 10.1109/TVCG.2012.128

[17] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proc. KDD*, pp. 16–22. AAAI Press, 1998. doi: 10.5555/3000292.3000296

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. SoCG*, pp. 253–262, 2004. doi: 10.1145/997817.997857

[19] A. Denton. Kernel-density-based clustering of time series subsequences using a continuous random-walk noise model. In *Proc. ICDM*, pp. 8–pp. IEEE, 2005. doi: 10.1109/ICDM.2005.84

[20] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, 2008. doi: 10.14778/1454159.1454226

[21] H. Doraiswamy, J. Freire, M. Lage, F. Miranda, and C. Silva. Spatio-temporal urban data analysis: A visual analytics perspective. *IEEE Comput. Graphics Appl.*, 38(5):26–35, 2018. doi: 10.1109/MCG.2018.053491728

[22] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *Proc. VLDB Endow.*, 12(2):112–127, 2018. doi: 10.14778/3282495.3282498

[23] P. Eichmann and E. Zgraggen. Evaluating subjective accuracy in time series pattern-matching using human-annotated rankings. In *Proc. IUI*, pp. 28–37. ACM, 2015. doi: 10.1145/2678025.2701379

[24] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD*, vol. 96, pp. 226–231, 1996. doi: 10.5555/3001460.3001507

[25] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng. Vmsp: Efficient vertical mining of maximal sequential patterns. In *Proc. Canadian Conference on Artificial Intelligence*, pp. 83–94. Springer, 2014. doi: 10.1007/978-3-319-06483-3_8

[26] M. C. Frith, M. C. Li, and Z. Weng. Cluster-buster: Finding dense clusters of motifs in dna sequences. *Nucleic Acids Res.*, 31(13):3666–3668, 2003. doi: 10.1093/nar/gkg540

[27] Y. Gao and J. Lin. Efficient discovery of time series motifs with large length range in million scale time series. In *Proc. ICDM*, pp. 1213–1222. IEEE, 2017. doi: 10.1109/ICDM.2017.8356939

[28] A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos. Comparing similarity perception in time series visualizations. *IEEE Trans. Vis. Comput. Graph.*, 25(1):523–533, 2019. doi: 10.1109/TVCG.2018.2865077

[29] D. Goldin, R. Mardales, and G. Nagy. In search of meaning for time series subsequence clustering: Matching algorithms based on a new distance measure. In *Proc. CIKM*, pp. 347–356, 2006. doi: 10.1145/1183614.1183666

[30] T. Gschwandtner, W. Aigner, K. Kaiser, S. Miksch, and A. Seyfang. Carecruiser: Exploring and visualizing plans, events, and effects interactively. In *Proc. PacificVis*, pp. 43–50. IEEE, 2011. doi: 10.1109/PACIFICVIS.2011.5742371

[31] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003. doi: 10.1145/777943.777945

[32] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, 2007. doi: 10.1007/s10618-006-0059-1

[33] M. C. Hao, M. Marwah, H. Janetzko, U. Dayal, D. A. Keim, D. Patnaik, N. Ramakrishnan, and R. K. Sharma. Visual exploration of frequent patterns in multivariate time series. *IV*, 11(1):71–83, 2012. doi: 10.1177/1473871611430769

[34] M. Herrmann and G. I. Webb. Early abandoning pruneddtw and its application to similarity search. *arXiv preprint arXiv:2010.05371*, 2020. doi: 10.48550/arXiv.2010.05371

[35] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Inf. Vis.*, 3(1):1–18, 2004. doi: 10.1057/palgrave.ivs.9500061

[36] W. Javed, B. McDonnel, and N. Elmqvist. Graphical perception of multiple time series. *IEEE Trans. Vis. Comput. Graph.*, 16(6):927–934, 2010. doi: 10.1109/TVCG.2010.162

[37] E. Keogh and J. Lin. Clustering of time-series subsequences is meaningless: Implications for previous and future research. *Knowl. Inf. Syst.*, 8(2):154–177, 2005. doi: 10.1007/s10115-004-0172-7

[38] P. Köthur, C. Witt, M. Sips, N. Marwan, S. Schinkel, and D. Dransch. Visual analytics for correlation-based comparison of time series ensembles. In *Comput. Graphics Forum*, vol. 34, pp. 411–420. Wiley Online Library, 2015. doi: 10.1111/cgf.12653

[39] J. Lin, E. Keogh, and S. Lonardi. Visualizing and discovering non-trivial patterns in large time series databases. *Inf. Vis.*, 4(2):61–82, 2005. doi: 10.1057/palgrave.ivs.9500089

[40] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 2–11. ACM, 2003. doi: 10.1145/882082.882086

[41] D. Luo, J. Yang, M. Krstajic, W. Ribarsky, and D. Keim. Eventriver: Visually exploring text collections with temporal references. *IEEE Trans. Vis. Comput. Graph.*, 18(1):93–105, 2012. doi: 10.1109/TVCG.2011.169

[42] M. Mannino and A. Abouzied. Expressive time series querying with hand-drawn scale-free sketches. In *Proc. CHI*. ACM, 2018. doi: 10.1145/3173574.3173962

[43] P. McLachlan, T. Munzner, E. Koutsofios, and S. North. Liverac: Interactive visual exploration of system management time-series data. In *Proc. CHI*, pp. 1483–1492. ACM, 2008. doi: 10.1145/1357054.1357286

[44] W. Meert, K. Hendrickx, and T. Van Craenendonck. wannesm/dtaidistance v2.0.0. Zenodo, 2020. doi: 10.5281/zenodo.3981067

[45] F. Miranda, M. Lage, H. Doraiswamy, C. Mydlarz, J. Salamon, Y. Lockerman, J. Freire, and C. T. Silva. Time lattice: A data structure for the interactive visual analysis of large time series. *Comput. Graphics Forum*, 37(3):23–35, 2018. doi: 10.1111/cgf.13398

[46] K. Mirylenka, V. Christophides, T. Palpanas, I. Pefkianakis, and M. May. Characterizing home device usage from wireless traffic time series. In *Proc. EDBT*, pp. 539–550, 2016. doi: 10.5441/002/edbt.2016.51

[47] K. Mirylenka, M. Dallachiesa, and T. Palpanas. Data series similarity using correlation-aware measures. In *Proc. SSDBM*, pp. 11:1–11:12, 2017. doi: 10.1145/3085504.3085515

[48] D. Moritz and D. Fisher. Visualizing a million time series with the density line chart. *arXiv preprint arXiv:1808.06019*, 2018. doi: 10.48550/arXiv.1808.06019

[49] P. K. Muthumanickam, K. Vrotsou, M. Cooper, and J. Johansson. Shape grammar extraction for efficient query-by-sketch pattern matching in long time series. In *Proc. VAST*, pp. 121–130. IEEE, 2016. doi: 10.1109/VAST.2016.7883518

[50] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997. doi: 10.1613/jair.374

[51] T. Oates. Identifying distinctive subsequences in multivariate time series by clustering. In *Proc. KDD*, pp. 322–326, 1999. doi: 10.1145/312129.312268

[52] T. Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Rec.*, 44(2):47–52, 2015. doi: 10.1145/2814710.2814719

[53] T. Palpanas and V. Beckmann. Report on the first and second interdisciplinary time series analysis workshop (itisa). *SIGMOD Rec.*, 48(3):36–40, 2019. doi: 10.1145/3377391.3377400

[54] P. Paraskevopoulos, T.-C. Dinh, Z. Dashdorj, T. Palpanas, L. Serafini, et al. Identification and characterization of human behavior patterns from mobile phone data. In *D4D Challenge session, NetMob*, 2013.

[55] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. ICDE*, p. 0215. IEEE, 2001. doi: 10.1109/ICDE.2001.914830

[56] F. Petitjean, A. Ketterlin, and P. Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognit.*, 44(3):678–693, 2011. doi: 10.1016/j.patcog.2010.09.013

[57] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20:53–65, 1987. doi: 10.1016/0377-0427(87)90125-7

[58] N. Ruta, N. Sawada, K. McKeough, M. Behrisch, and J. Beyer. Sax navigator: Time series exploration through hierarchical clustering. In *Proc. VIS*, pp. 236–240. IEEE, 2019. doi: 10.1109/VISUAL.2019.8933618

[59] D. Sacha, M. Kraus, J. Bernard, M. Behrisch, T. Schreck, Y. Asano, and D. A. Keim. Somflow: Guided exploratory cluster analysis with self-organizing maps and analytic provenance. *IEEE Trans. Vis. Comput. Graph.*, 24(1):120–130, 2017. doi: 10.1109/TVCG.2017.2744805

[60] T. Saito, H. N. Miyamura, M. Yamamoto, H. Saito, Y. Hoshiya, and T. Kaseda. Two-tone pseudo coloring: Compact visualization for one-dimensional data. 2005. doi: 10.1109/INFOVIS.2005.35

[61] S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, 2007. doi: 10.3233/IDA-2007-11508

[62] T. Schreck, T. Tekušová, J. Kohlhammer, and D. Fellner. Trajectory-based visual analysis of large financial time series data. *ACM SIGKDD Expl. Newsl.*, 9(2):30–37, 2007. doi: 10.1145/1345448.1345454

[63] G. Shurkhovetskyy, N. Andrienko, G. Andrienko, and G. Fuchs. Data abstraction for visualizing large time series. In *Comput. Graphics Forum*, vol. 37, pp. 125–144. Wiley Online Library, 2018. doi: 10.1111/cgf.13237

[64] M. Sips, P. Kothur, A. Unger, H.-C. Hege, and D. Dransch. A visual analytics approach to multiscale exploration of environmental time series. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2899–2907, 2012. doi: 10.1109/TVCG.2012.191

[65] M. Steiger, J. Bernard, S. Mittelstädt, H. Lücke-Tieke, D. Keim, T. May, and J. Kohlhammer. Visual analysis of time-series similarities for anomaly detection in sensor networks. In *Comput. Graphics Forum*, vol. 33, pp. 401–410. Wiley Online Library, 2014. doi: 10.1111/cgf.12396

[66] L. Stopar, P. Skraba, M. Grobelnik, and D. Mladenic. Streamstory: Exploring multivariate time series on multiple scales. *IEEE Trans. Vis. Comput. Graph.*, 25(4):1788–1802, 2019. doi: 10.1109/TVCG.2018.2825424

[67] A. Struyf, M. Hubert, P. Rousseeuw, et al. Clustering in an object-oriented environment. *J. Stat. Softw.*, 1(4):1–30, 1997. doi: 10.18637/jss.v001.i04

[68] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 63(2):411–423, 2001. doi: 10.1111/1467-9868.00293

[69] E. Tufte and P. Graves-Morris. The visual display of quantitative information, 1983. doi: 10.1119/1.14057

[70] I. van der Hoog, L. Ost, E. Rotenberg, and D. Rutschmann. Efficient Greedy Discrete Subtrajectory Clustering. In *Proc. SoCG*, vol. 332 of *LIPIcs*, pp. 78:1–78:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2025. doi: 10.4230/LIPIcs.SoCG.2025.78

[71] A. Van Goethem, F. Staals, M. Löffler, J. Dykes, and B. Speckmann. Multi-granular trend detection for time-series analysis. *IEEE Trans. Vis. Comput. Graph.*, 23(1):661–670, 2017. doi: 10.1109/TVCG.2016.2598619

[72] J. J. Van Wijk and E. R. Van Selow. Cluster and calendar based visualization of time series data. In *Proc. InfoVis*, pp. 4–9. IEEE, 1999. doi: 10.1109/INFVIS.1999.801851

[73] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proc. ICDE*, pp. 79–90. IEEE, 2004. doi: 10.1109/ICDE.2004.1319986

[74] M. Wang, Z. B. Abrams, S. M. Kornblau, and K. R. Coombes. Thresher: Determining the number of clusters while removing outliers. *BMC Bioinformatics*, 19:1–15, 2018. doi: 10.1186/s12859-017-1998-9

[75] M. O. Ward and Z. Guo. Visual exploration of time-series data with shape space projections. In *Comput. Graphics Forum*, vol. 30, pp. 701–710. Wiley Online Library, 2011. doi: 10.1111/j.1467-8659.2011.01919.x

[76] C. Ware. *Information Visualization: Perception for Design*. Elsevier, 2012. doi: 10.5555/2285540

[77] J. Zhao, F. Chevalier, E. Pietriga, and R. Balakrishnan. Exploratory analysis of time-series with chronolenses. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2422–2431, 2011. doi: 10.1109/TVCG.2011.195

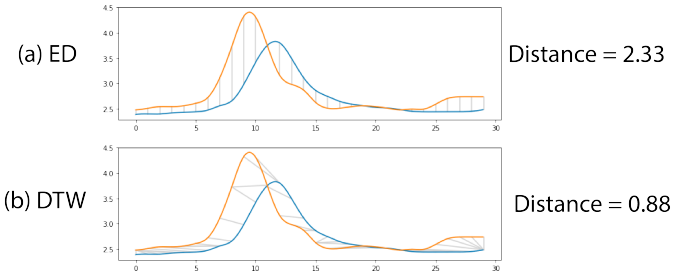

Fig. 11: (a) Euclidean Distance (ED) sums up the L_2 distance between the points of two time series at the same temporal positions. (b) Dynamic Time Warping (DTW) matches the points (i.e., the grey lines) first even though they are not aligned on the time axis.

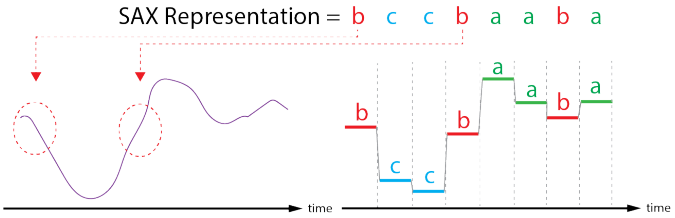

Fig. 12: Illustration of Symbolic Aggregated Approximation (SAX). Some time segments with different shapes may belong to the same symbol due to averaging from Piecewise Aggregated Approximation (PAA).

APPENDIX A BACKGROUND ON TIME SERIES SUBSEQUENCE MINING

We give an overview of time series subsequence clustering to provide the intuition behind our approach to the problem. To identify common time series subsequences, techniques have been proposed that transform the real-valued time series into *discrete sequences*. Figure 12 illustrates a popular technique called Symbolic Aggregate approXimation (SAX) [40]. Although SAX is not designed for visualizing the real-valued data series since it aggregates the series as a sequence of flat lines (Figure 12), we are inspired by it for discretizing the time series for subsequence clusterings. First, the time series is split into w segments with equal durations. Then, for each segment, an alphabet representing a range with equal probable distribution is assigned based on the average of the values within the segment (i.e., piece-wise aggregate approximation (PAA)). Since the time series becomes a discrete sequence, pattern mining techniques can be applied to obtain the repetitive substrings among all sequences in the dataset. Subsequence clusters are then created from these substrings.

Our method is similar to SAX in terms of discretizing the real-valued series to symbols, but we focus on creating symbols that represent series with similar shapes instead of similar average values in a time window. Also, we propose a pattern mining approach that derives clusters to attain scalability in time series visualization by reducing the number of small multiples.

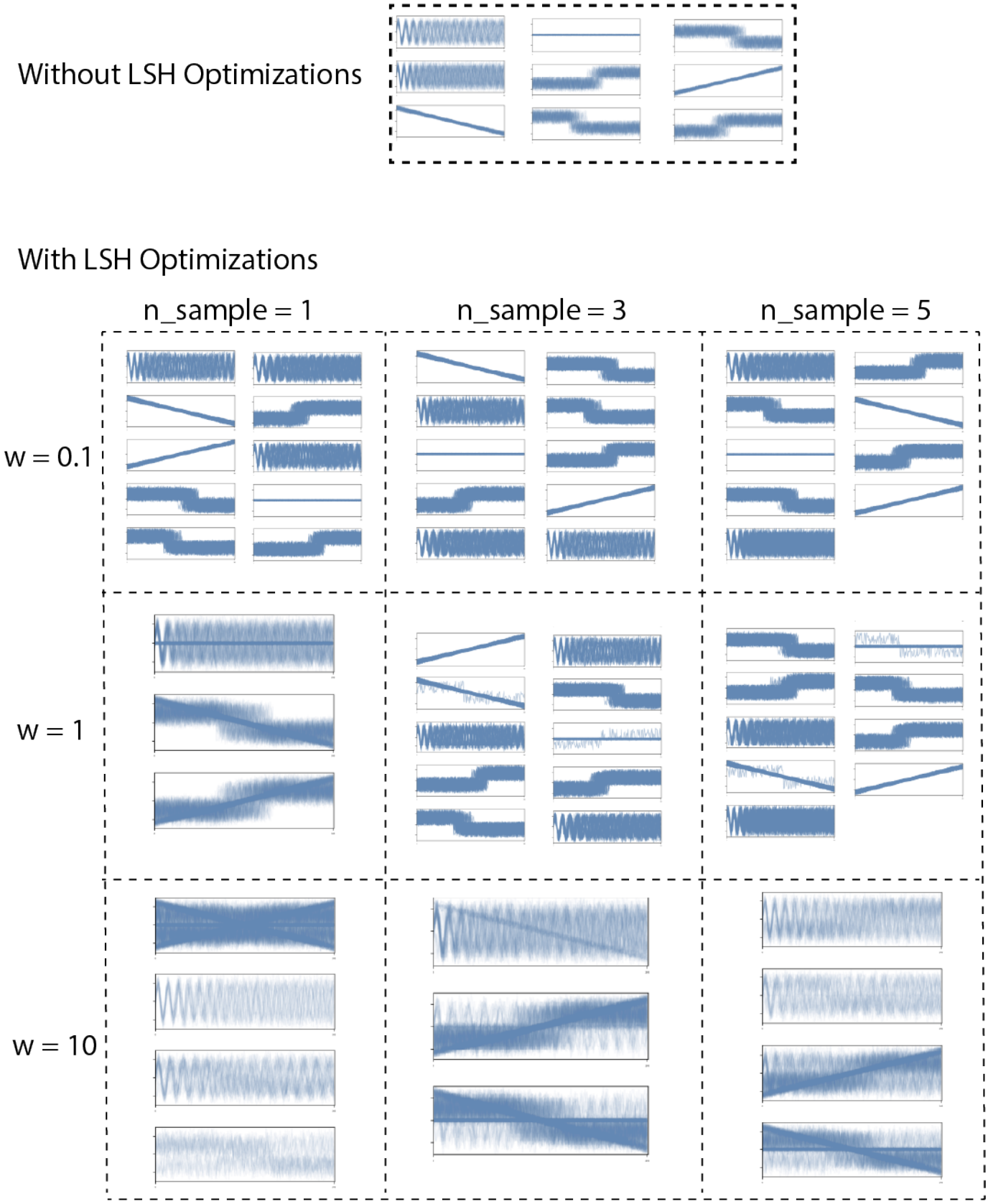

Fig. 13: Qualitative results of LSH accelerated clustering with increasing widths and number of samples. Overall, increasing *w* plays a more critical role than increasing the number of samples.