k-Graph: A Graph Embedding for Interpretable Time Series Clustering

Paul Boniol, Donato Tiano, Angela Bonifati, Themis Palpanas

Abstract—Time series clustering poses a significant challenge with diverse applications across domains. A prominent drawback of existing solutions lies in their limited interpretability, often confined to presenting users with centroids. In addressing this gap, our work presents k-Graph, an unsupervised method explicitly crafted to augment interpretability in time series clustering. Leveraging a graph representation of time series subsequences, k-Graph constructs multiple graph representations based on different subsequence lengths. This feature accommodates variable-length time series without requiring users to predetermine subsequence lengths. Our experimental results reveal that k-Graph outperforms current state-of-the-art time series clustering algorithms in accuracy, while providing users with meaningful explanations and interpretations of the clustering outcomes.

Index Terms—Time Series, Clustering, Interpretability

I. INTRODUCTION

Massive collections of time series are becoming a reality in virtually every scientific and social domain [1], [2]. Thus, there is a significant need for multiple relevant applications for proposing methods that can efficiently analyze them [3]. Examples of fields that involve data series are finance, environmental sciences, astrophysics, neuroscience, engineering, multimedia, etc. [3]–[5]. Once collected, the major analysis tasks on time series include pattern matching (or similarity search) [6]–[8], classification [9]–[12], clustering [13]–[15], anomaly detection [16]–[20] and motif discovery [21].

Time series clustering poses a pivotal and complex challenge in data science, garnering substantial attention with many proposed algorithms in recent years. Traditional approaches hinge on distance measures, exemplified by k-Means clustering utilizing Euclidean or Dynamic Time Warping (DTW) distances, and the widely adopted k-Shape algorithm [14] serves as a prominent baseline method. However, recent studies underscore the efficacy of feature-based methods, where clustering operates on extracted time series features, showcasing robust performance in accuracy and execution time [22]. Despite their success, these methods grapple with noteworthy limitations: (i) susceptibility to noisy time series, which can compromise clustering performance, and (ii) a need for more interpretability in most solutions, hindering a comprehensive understanding of the outcomes. The last limitation is a challenging problem, especially when the important features to discover are typical subsequences that could be a grouping criterion for a given cluster. Therefore, identifying such subsequences and providing an interpretable representation of the clustering partitions to the user is essential to enhance the usage and maximize the understanding and trust of unsupervised clustering in time series applications. Unfortunately, none of the existing methods allow a straightforward solution for this problem.
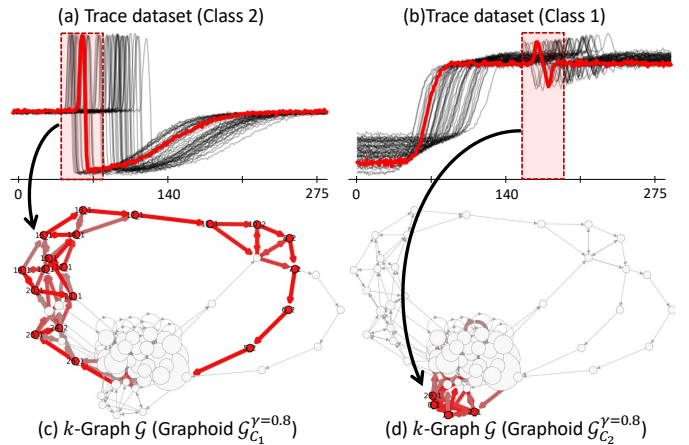

Fig. 1: k-Graph resulting graph $\mathcal{G}$ when applied on the Trace dataset [23]

This paper introduces k-Graph, a novel graph-based univariate time series clustering method that aims to overcome the shortcomings of existing approaches and the problems mentioned above. To the best of our knowledge, k-Graph is the first approach proposing a graph-based representation of the time series for the purpose of clustering. Overall, k-Graph is based on a four-step process: The first step involves **Graph Embedding** where several graphs are computed leveraging an adapted Series2Graph algorithm [17]. Each graph encapsulates groups of similar subsequences within a dataset, with nodes representing these groups and edges carrying weights based on sequence occurrences. Moving forward, in the **Graph Clustering** phase, features are extracted for time series using the graph nodes and edges. The k-Means algorithm is then applied for clustering based on these features. Next, spectral clustering is deployed in **Consensus Clustering** to establish consensus across the multiple partitions obtained in the previous step. This step yields the final labels assigned by k-Graph. Lastly, the most relevant graph is selected in **Interpretability Computation**, and graphoids are computed to enhance interpretability (examples of graphoids for the Trace dataset [23] are depicted in Figure 1). This step contributes to a more profound understanding of the clustering outcomes.

Our contributions are as follows:

1) We provide a new problem formulation for graph embedding for time series clustering (Section II-D). We formalize the concept of interpretability regarding clustering with graph representation of time series and how to measure it.
2) We propose a novel graph embedding method for univariate time series clustering. The latter can be used on

time series of variable length and provides through the graph an interpretable interface for the user to dive into the time series datasets and extract meaningful patterns that compose the different clusters (Section III).

3) We demonstrate that our proposed approach is at least equalling and outperforming under certain scenarios, the current state-of-the-art clustering methods for time series, and being of the same order of magnitude regarding execution time (Section IV).

4) In addition to being as accurate and as expensive to compute, we demonstrate through multiple examples on real world datasets the interpretability power of k-Graph, and the usefulness of the graph for knowledge discovery tasks (Section IV).

5) We provide an open-source implementation of our approach (https://github.com/boniolp/kGraph).

We conclude by discussing the implications of our work and exploring future directions to enhance the accuracy, execution time, and interpretability of our proposed method (Section V).

II. BACKGROUND AND RELATED WORK

We first introduce notations useful for the rest of the paper (Section II-A). Then, we review existing time-series clustering methods and discuss their limitations related to interpretability and user interaction (Section II-B). We then motivate and introduce the usage of Graph embedding for time series clustering (Section II-C). We finally properly define the problem tackled in this paper (Section II-D).

A. Time Series and Graph Notation

Time Series: A univariate time series $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $T_i \in \mathbb{R}$ $[T_1, T_2, ..., T_n]$, where $n = |T|$ is the length of T, and T_i is the ith point of T. In the rest of this paper , we refer to univariate time series as *time series*. We are typically interested in local regions of the time series, known as subsequences. A subsequence $T_{i,\ell} \in \mathbb{R}^\ell$ of a time series T is a continuous subset of the values of T of length ℓ starting at position i. Formally, $T_{i,\ell} = [T_i, T_{i+1}, ..., T_{i+\ell-1}]$. A dataset $\mathcal{D}$ is a set of time series (possibly of different lengths). We define the size of $\mathcal{D}$ as $|\mathcal{D}|$.

Graph: We introduce some basic definitions for graphs, which we will use in this paper. We define a Node Set $\mathcal{N}$ as a set of unique integers. Given a Node Set $\mathcal{N}$, an Edge Set $\mathcal{E}$ is then a set composed of tuples (x_i, x_j), where $x_i, x_j \in \mathcal{N}$. Given a Node Set $\mathcal{N}$, an Edge Set $\mathcal{E}$ (pairs of nodes in $\mathcal{N}$), a Graph $\mathcal{G}$ is an ordered pair $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. A directed graph $\mathcal{G}$ is an ordered pair $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is a Node Set, and $\mathcal{E}$ is an ordered Edge Set. In the rest of this paper, we will only use directed graphs, denoted as $\mathcal{G}$.

B. Time Series Clustering

Time series clustering plays a pivotal role in uncovering meaningful patterns within temporal datasets, where the primary goal is to group similar time series for insightful analysis. The specific challenge addressed by partitional time series clustering involves partitioning a dataset of n time series, denoted as D, into k clusters ($C = \{C_1, C_2, ..., C_k\}$), where the number of clusters, k, is predetermined. Although the choice of k can be determined using wrapper methods, it is assumed to be fixed in advance in many experiments.

1) Raw-Based Approaches: Clustering algorithms for time series can either operate directly on raw data or use transformations to derive features before clustering. Algorithms specifically designed for time series often prefer using raw data [24] for several reasons. First, **Preservation of Information**: raw time series preserve intricate details of temporal patterns, ensuring information fidelity. Second, **Temporal Dependencies**: they capture dynamic patterns and temporal interdependencies that might be lost during feature extraction. Finally, **Data Exploration**: direct analysis of raw time series fosters the discovery of unexpected patterns and trends, supporting an exploratory, discovery-driven approach.

In the raw-based approach, the k-Means algorithm is commonly used to identify patterns and similarities within temporal data. Each time series is treated as a multidimensional vector, with data points representing observations over time. The goal is to partition the time series into k clusters, optimizing the assignment of series to cluster centers.

k-Shape [25] is a well-known algorithm for time series clustering in this approach. It starts by randomly selecting initial cluster centers and uses a specialized distance metric to measure dissimilarity based on shapes, addressing temporal dynamics. The algorithm iteratively assigns each time series to the cluster with the smallest shape-based distance and updates cluster centers based on the mean shape. This process continues until convergence, resulting in clusters that highlight shape similarities and provide insights into temporal dynamics.

A recent study [26] has compared k-Shape with a large amount of clustering baselines and has demonstrated that none of the recent baselines outperform significantly k-Shape.

Finally, several raw-basd approach are using the Dynamic Time Warping (DTW), a well-known algorithm for measuring the similarity between two time series by aligning them through time axis warping [27]. This process minimizes the distance between sequences, accommodating shifts and stretches in the time dimension. Although DTW is highly accurate, it is computationally intensive, often necessitating optimizations for improved efficiency. SOMTimeS [28] is an algorithm that addresses this challenge. It is a self-organizing map (SOM) designed for clustering time series data, utilizing DTW as a distance measure to effectively align and compare time series. SOMTimeS enhances computational efficiency by implementing a pruning strategy that significantly reduces the number of DTW calculations required during the training.

However, it is important to note that raw-based approaches may face challenges when dealing with noise in raw time series, potentially obscuring meaningful patterns. Additionally, the direct clustering of raw time series might result in clusters lacking clear distinctions or meaningful insights, making extracting valuable information from the data challenging.

2) Feature-Based Approaches: Adopting a feature-based [29] approach offers several advantages that address the challenges associated with raw-based methods: **Dimensionality Reduction**, feature extraction often reduces dimensionality. Indeed, lower-dimensional feature representations enhance computational efficiency and reduce the risk of the curse of dimensionality; **Enhanced Discrimination**, feature selection can be tailored to emphasize specific characteristics crucial for discrimination, enhancing the ability of clustering algorithms to distinguish subtle differences between time series, leading

to more accurate clustering; and **Interpretability**, clusters derived from features often yield more interpretable results than those directly from raw time series. Interpretability is crucial for extracting insights, and features-based approaches provide clearer explanations for grouping.

TS3C [30] is an example of feature-based clustering approach. The latter segments the time series and extracts a set of statistical features to represent them. The features are then used to cluster time series. FeatTS [31] emerges as a feature-based algorithm tailored for univariate time series clustering. It leverages the TSFresh library and employs Principal Features Analysis to extract salient features from time series data. Building upon this foundation, the authors introduce Time2Feat (T2F) [22], an algorithm designed explicitly for multivariate time series clustering. T2F distinguishes itself by adopting two distinct feature extraction approaches named intra and inter-signal feature extraction. Furthermore, the authors enhance the feature selection process by introducing a grid search method for selecting optimal features for clustering multivariate time series. FeatTS and Time2Feat are two approaches that accommodate unsupervised and semi-supervised clustering scenarios. Notably, Time2Feat stands out as the current state-of-the-art solution for multivariate time series clustering, showcasing the advancements in feature extraction and selection methodologies.

Nevertheless, feature-based approaches may suffer from the information loss induced by the transformation of the original time series (i.e., sequential patterns) into features. Handling high-dimensional feature spaces poses a challenge, especially when dealing with numerous derived features. This complexity can compromise the interpretability of the algorithm, making it difficult for users to examine all the extracted features. Additionally, the sensitivity of feature engineering poses a challenge, where selecting inappropriate features or applying unsuitable transformations may affect clustering quality.

3) Deep Learning Approaches: In recent years, deep learning strategies [32] have emerged to address the challenges of time series clustering, demonstrating strong performance. These approaches leverage deep learning's ability to directly interface with time series data, marking a significant shift from traditional methods. A key advantage is the **Hierarchy of Abstraction**, where deep learning architectures capture complex relationships within temporal data, revealing intricate patterns essential for effective clustering. Additionally, these methods exhibit **High Efficacy**, surpassing traditional techniques with superior accuracy and efficiency.

The Deep Auto-Encoder (DAE) [33] is a popular solution, serving as an unsupervised model for representation learning. It transforms raw input data into new space representations, extracting valuable features through encoding. The DAE architecture, characterized by seven fully connected layers, effectively harnesses learned features through an internal layer. These features are subsequently input into a clustering loss function, minimizing the distance between data points and their respective assigned cluster centers.

Another approach is Deep Temporal Clustering (DTC) [34], which uses the DAE for feature representation and clustering. The DTC's clustering layer optimizes the Kullback-Leibler (KL) divergence objective, aligning with a self-training target distribution. The encoding process influences clustering performance, with learned representations fed into the k-Means algorithm for final clustering.

In conclusion, despite the high efficacy and the ability to uncover complex patterns, deep learning approaches encounter fundamental challenges related to the interpretability of their decision-making processes. The inherent complexity of these machine learning models often results in a lack of transparency in understanding obtained results. Furthermore, these approaches may struggle to integrate domain-specific knowledge seamlessly, presenting obstacles to guiding the clustering process based on expert insights.

C. Graph Embedding of Time Series

We argue that to address the challenges of maintaining information preservation while maximizing interpretability, a viable solution is to represent time series as a (suitably constructed) graph. Constructing such a graph involves processing subsequences of the time series dataset. These subsequences represent various types of patterns and their temporal succession. This approach furnishes clustering methods with substantial information, contributing to the sustenance of high accuracy. Additionally, this approach facilitates user-friendly navigation through the time series.

Various methods have been proposed to convert time series into graphs for specific analytical tasks to overcome the challenges mentioned earlier. Series2Graph embeds univariate time series into a directed graph [17], [35], primarily employed for anomaly detection. Similarly, approaches like Time2Graph [36] utilize graph representations of time series to address time series classification. The advantages of such time series graph representations are threefold. First, these graph representations are easily interpretable by any user. Second, constructed directly from subsequences of the time series, they preserve essential information. Lastly, a unified embedding can significantly reduce execution time, as evidenced in anomaly detection scenarios [35].

However, prior works on graph embedding for time series were often proposed either under supervised settings [36], simplifying the graph construction task, or in an unsupervised manner but for continuous time series [17]. For the specific task of time series clustering, a graph embedding of time series necessitates the unsupervised construction of a single, comprehensive graph for an entire dataset, encompassing multiple continuous time series. While a straightforward solution would be building one graph per time series, this approach diminishes the interpretability advantage of having a single, concise graph.

D. Problem Formulation

We propose a novel approach to time series clustering by employing a graph embedding. The essence of this methodology lies in transforming a time series dataset into a sequence of abstract states corresponding to different subsequences within the dataset. These states are represented as nodes, denoted by $\mathcal{N}$, in a directed graph, $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. The edges, $\mathcal{E}$, encode the frequency with which one state occurs after another [17]. We define this graph as follows:

Definition 1 (Graph Embedding): Let a time series dataset be defined as $\mathcal{D} = \{T_1, T_2, ..., T_n\}$. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a directed graph. $\mathcal{G}$ is the graph embedding of $\mathcal{D}$ if there
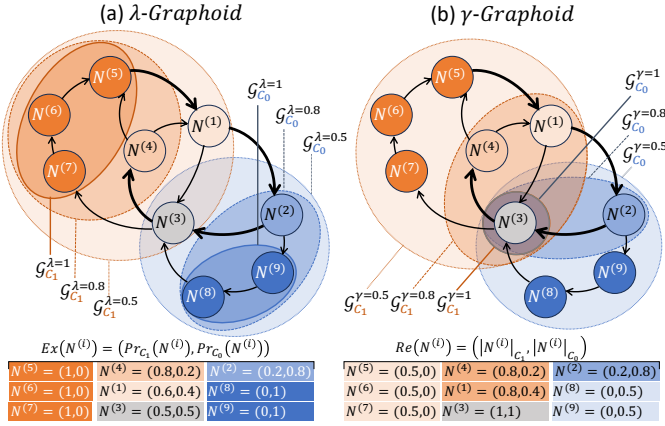
Fig. 2: λ-Graphoids and γ-Graphoids for different λ and γ.

exists a function $\mathcal{M}$ such that for any $T \in \mathcal{D}$, $\mathcal{M}(T) = \langle N^{(1)}, N^{(2)}, ..., N^{(m)} \rangle$, and $\forall i \in [1, m], N^{(i)} \in \mathcal{N}$, and $\forall i \in [1, m-1], (N^{(i)}, N^{(i+1)}) \in \mathcal{E}$.

The rest of the section considers $\mathcal{M}(T)$ as a subgraph. For the sake of interpretability, it is imperative that different sections of the graph distinctly capture the similarity between time series. Nodes, representing similar patterns from various time series, and edges, denoting possible transitions between these patterns, play a pivotal role in this context. Consequently, a comparable time series is found to be placed within the same region of the graph. As a result, in a given dataset $\mathcal{D} = \{T_0, T_1, ..., T_n\}$, a designated cluster of time series, denoted as $C_i \subset \mathcal{D}$, corresponds to a discernible subgraph $\mathcal{G}_{C_i} \subset \mathcal{G}$. This subgraph is formally called a "*Graphoid*."

Definition 2 (Graphoid): Let $\mathcal{D}$ be a time series dataset and $C_i \subset \mathcal{D}$ a given cluster such as $C_i = \{T_1, T_2, ..., T_{k'}\}$. Let $\mathcal{G}$ be the graph embedding of $\mathcal{D}$ resulting from a function $\mathcal{M}$. We define the *Graphoid* of C_i as:

$$\mathcal{G}_{C_i} = \bigcup_{T \in C_i} \mathcal{M}(T)$$

In Definition 2, the *Graphoid* of a given cluster contains all the nodes and edges that at least one time series of that cluster crossed. Therefore, a node of $\mathcal{G}$ may belong to multiple graphoids. As a consequence, no distinction is made between nodes that contain subsequences of all time series of one cluster (i.e., **representativity**) or time series subsequences of one cluster only (i.e., **exclusivity**) and nodes that are contained in all clusters. We thus introduce the two following definitions:

Definition 3 (Node representativity): Let $\mathcal{D}$ a time series dataset and $C = \{C_1, C_2, ..., C_k\}$ a clustering partition. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be the graph embedding of $\mathcal{D}$ resulting from a function $\mathcal{M}$. We define the *Representativity* of node $N \in \mathcal{N}$ as $Re(N) = (|N|_{C_1}, ..., |N|_{C_k}$ with $|N|_{C_i}$ defined as:

$$|N|_{C_i} = \frac{1}{|C_i|} \sum_{T \in C_i} 1_{[N \in \mathcal{M}(T)]}$$

Definition 4 (Node Exclusivity): Let $\mathcal{D}$ a time series dataset and $C = \{C_1, C_2, ..., C_k\}$ a clustering partition. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be the graph embedding of $\mathcal{D}$ resulting from a function $\mathcal{M}$. We define the *Exclusivity* of node $N \in \mathcal{N}$ as $Ex(N) = (Pr_{C_1}(N), ..., Pr_{C_k}(N))$ with $Pr_{C_i}(N)$ defined as:

$$Pr_{C_i}(N) = \frac{|C_i||N|_{C_i}}{\sum_{T \in \mathcal{D}} 1_{[N \in \mathcal{M}(T)]}}$$

In other words, the **representativity** of a node is the number of time series of a given cluster that crossed the node divided by the total number of time series within that cluster. The **exclusivity** of a node is the number of time series of a given cluster that crossed the node divided by the total number of time series that crossed that same node. The same definitions can be used for edges. Based on the above definitions, we can restrict the definition of a *Graphoid* based on exclusivity and representativity. We thus introduce λ-*Graphoid* and γ-*Graphoid* defined as follows:

Definition 5 (λ-Graphoid): For a given dataset $\mathcal{D}$, $\mathcal{G}$ the graph embedding of $\mathcal{D}$, and a given cluster C_i. The λ-Graphoid of C_i is defined as $\mathcal{G}_{C_i}^{\lambda} = (\mathcal{N}_{C_i}^{\lambda}, \mathcal{E}_{C_i}^{\lambda})$ such as $\forall N \in \mathcal{N}_{C_i}^{\lambda}, \forall E \in \mathcal{E}_{C_i}^{\lambda}, Pr_{C_i}(N) \geq \lambda$ and $Pr_{C_i}(E) \geq \lambda$.

Definition 6 (γ-Graphoid): For a given dataset $\mathcal{D}$, $\mathcal{G}$ the graph embedding of $\mathcal{D}$, and a given cluster C_i. The γ-Graphoid of C_i is defined as $\mathcal{G}_{C_i}^{\gamma} = (\mathcal{N}_{C_i}^{\gamma}, \mathcal{E}_{C_i}^{\gamma})$ such as $\forall N \in \mathcal{N}_{C_i}^{\gamma}, \forall E \in \mathcal{E}_{C_i}^{\gamma}, |N|_{C_i} \geq \gamma$ and $|E|_{C_i} \geq \gamma$.

The concepts introduced above can be better illustrated using Figure 2. The λ-*Graphoid* and γ-*Graphoid* are influenced by the chosen values of λ and γ. As demonstrated in Figure 2, higher values of λ and γ lead to more restrictive graphoids. In the illustration, nodes $N^{(5)}, N^{(6)}, N^{(7)}$ are exclusively crossed by time series from cluster C_1, highlighting unique patterns specific to this cluster. On the other hand, node $N^{(3)}$ is traversed by all time series of C_1 and is considered the most representative pattern for this cluster. However, it also represents a common pattern for C_0.

This scenario emphasizes the trade-off between representativity and exclusivity. While $N^{(3)}$ provides a comprehensive representation of C_1, it lacks exclusivity to this cluster. In contrast, $N^{(5)}, N^{(6)}, N^{(7)}$ offer exclusive patterns for C_1 but might not be present in all time series of this cluster, limiting the interpretability. Thus, finding an optimal balance between higher values of λ and γ is crucial for maximizing the interpretability of a clustering partition through graph embedding. Based on the above definitions, we can state the following:

Lemma 1: For a given clustering partition $C = \{C_1, C_2, ..., C_k\}$, if $\lambda \leq k$, then $\bigcup_{C_i \in C} \mathcal{G}_{C_i}^{\lambda} = \mathcal{G}$. if $\lambda > 0.5$, then $\bigcap_{C_i \in C} \mathcal{G}_{C_i}^{\lambda} = \emptyset$.

Lemma 2: For a given clustering partition $C = \{C_1, C_2, ..., C_k\}$, if $\forall C_i \in C, \mathcal{G}_{C_i}^{\lambda=1} = \mathcal{G}_{C_i}^{\gamma=\frac{1}{|C_i|}}$, then $\bigcap_{C_i \in C} \mathcal{G}_{C_i} = \emptyset$

Lemma 2 corresponds to the perfect partition, in which each graph region exclusively represents one cluster. However, we do not need to have all the nodes of a *Graphoid* exclusively representing only one cluster. It is sufficient to have one node for each cluster with $|N|_{C_i} = Pr_{C_i}(N) = 1$. Therefore, the problem we want to solve is the following.

Problem 1 (Time Series Graph Clustering): Given a dataset $\mathcal{D}$, automatically construct the graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, and compute a clustering partition $C = \{C_1, ..., C_k\}$ of $\mathcal{D}$, such that:

$$\forall C_i \in C, |\mathcal{G}_{C_i}^{\lambda=1} \cap \mathcal{G}_{C_i}^{\gamma=1}| > 0$$

As this problem is impossible to solve in some use cases, the objective is to find the largest possible values of λ and γ, such that the condition in Problem 1 holds and the values of
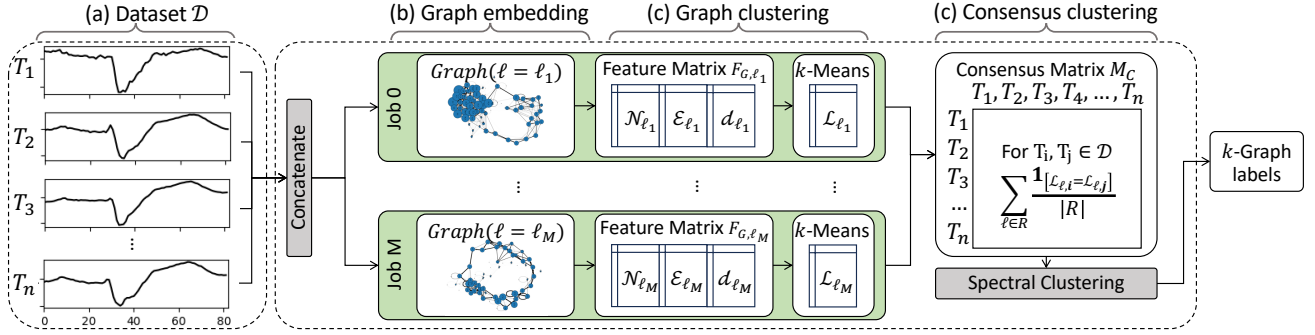
Fig. 3: k-Graph pipeline.

Symbol	Description
T	a time series (of length $|T|$)
ℓ	subsequence length
$\mathcal{D}$	a dataset of time series
C_i	a cluster of a clustering partition C
$\mathcal{L}$	labels generated by clustering
k	number of clusters
$\mathcal{N}, \mathcal{E}$	set of nodes and edges
d	degrees of a set of nodes
$\mathcal{G}$	directed graph
$\mathcal{M}$	function that transform T into $\mathcal{G}$
$\mathcal{G}_\ell$	graph embedding built with length ℓ
$F_{\mathcal{G},\ell}$	feature matrix for graph $\mathcal{G}_\ell$
M_C	consensus matrix
$\mathcal{G}_{C_i}, \mathcal{G}_{C_i}^\lambda, \mathcal{G}_{C_i}^\gamma$	graphoid, λ-graphoid and γ-graphoid of C_i
$|N|_{C_i}, Pr_{C_i}(N)$	representativity, exclusivity of node N for C_i

TABLE I: Table of symbols

λ and γ indicate the quality of the clustering interpretability. Table I summarizes the symbols used in this paper.

III. PROPOSED APPROACH

In this section, we describe k-Graph, an approach for time series clustering, as a solution to tackle the problem described in the previous section. For a given dataset $\mathcal{D}$, the overall k-Graph process is divided into three main steps as follows:

1) **Graph Embedding** (Section III-A): for M different subsequence lengths, we compute M graphs. For a given subsequence length ℓ, The set of nodes represent groups of similar subsequences of length ℓ within the dataset $\mathcal{D}$. The edges have weights corresponding to the number of times one subsequence of a given node has been followed by a subsequence of the other node.

2) **Graph Clustering** (Section III-B): For each graph, we extract a set of features for all time series of the dataset $\mathcal{D}$. These features correspond to the nodes and edges that the time series crossed. Then, we use these features to cluster the time series using k-Means for scalability reasons.

3) **Consensus Clustering** (Section III-C): At this point we have M clustering partitions (i.e., one per graph). We build a consensus matrix M_C. We then cluster this matrix using spectral clustering in the objective of grouping time series that are highly connected (i.e., grouped in the same cluster in most of the M partitions). The output of this clustering step is the labels provided by k-Graph.

4) **Interpretability Computation** (Section III-D): after obtaining the clustering partition, we select the most relevant graph (among the M graphs), and we compute the interpretable graphoids.

Algorithm 1: k-Graph

input : Dataset $\mathcal{D}$, Number of clusters k, Number of length M (default is 30), Maximum length rate rml (default is 0.4), Sample rate $smpl$ (default is 10)

output: Clustering labels $\mathcal{L}$

1   $all\text{-}\mathcal{L} \leftarrow []$;
2   $R \leftarrow M$ random lengths in $[5, (min_{T \in \mathcal{D}}|T|) * rml]$;
3   **foreach** $\ell \in R$ **do**
4     $\mathcal{G}_\ell \leftarrow Graph(\mathcal{D}, \ell, smpl)$ // Section III-A
5     $F_{\mathcal{G},\ell} \leftarrow extractFeature(\mathcal{D}, \mathcal{G}_\ell)$ // Section III-B
6     $\mathcal{L}_\ell \leftarrow k\text{-}Means(F_{\mathcal{G},\ell}, k)$ // Section III-B
7   add $\mathcal{L}_\ell$ in $all\text{-}\mathcal{L}$;
8   $M_C \leftarrow Consensus(all\text{-}\mathcal{L})$ // Section III-C
9   $\mathcal{L} \leftarrow SpectralClustering(M_c, k)$ // Section III-C
10   **return** $\mathcal{L}$

Figure 3 illustrates the different steps of k-Graph, detailed in Algorithm 1. Below, we describe each step in detail.

A. Graph Embedding

As described in Section II-D, the general objective is to build one graph representing the time series dataset on which we can perform clustering. For this step, a solution could be to use the Series2Graph algorithm [17]. Nevertheless, Series2Graph presents three limitations. First, it accepts a continuous time series as input, rather than a dataset, that may contain several series of different lengths. Second, the user must specify a subsequence length ℓ. This becomes problematic in interpretable clustering, as determining an optimal length in advance is challenging. Third, the embedding employed by Series2Graph does not scale to large dataset sizes, which would be a significant drawback in our case. In k-Graph, we propose the following graph embedding procedure.

For a dataset $\mathcal{D}$, we build M different graphs $\mathcal{G}_\ell = (\mathcal{N}_\ell, \mathcal{E}_\ell)$ for M different subsequence length ℓ. Algorithm 2 outlines the procedure for constructing a single graph for a given dataset and subsequence length. The procedure is the following:

1) **Subsequence Embedding**: For each time series $T \in \mathcal{D}$, we collect all the subsequences of a given length ℓ into an array called $Proj(T, \lambda)$. We then concatenate all the computed $Proj(T, \lambda)$ into $Proj$ for all the time series in the dataset (Line 3). We then sample $Proj$ (user-defined parameter $smpl$) and keep only a limited number of subsequences stored in $Proj_{smpl}$. We use the latter to train a Principal Component Analysis (PCA) (Line 5). We then use the trained PCA and a rotation step to project all the subsequences into a two-dimensional space that only

---

Algorithm 2: Graph Embedding $Graph(\mathcal{D}, \ell)$

input : Dataset $\mathcal{D}$, input length ℓ, sample $smpl$
output: graph $\mathcal{G}_\ell$

    // Concatenante subsequences
1 $Proj \leftarrow [];$
2 **foreach** $T \in \mathcal{D}$ **do**
3 | add $Proj(T, \ell)$ in $Proj$;
    // Reduce to a two-dimensional space
4 $Proj_{smpl} \leftarrow$ randomly select $\lfloor \frac{|Proj|}{smpl} \rfloor$ elements in $Proj$;
5 $pca \leftarrow PCA.fit(Proj_{smpl});$
6 $SProj \leftarrow Reduce(Proj, pca);$
    // Extract the nodes and edges
7 $\mathcal{N}_\ell, \mathcal{E}_\ell \leftarrow NodeCr(SProj_{smpl}), EdgeCr(SProj, \mathcal{N}_\ell);$
8 return $\mathcal{G}_\ell = (\mathcal{N}_\ell, \mathcal{E}_\ell)$

---

preserves the shapes of the subsequences. The result is denoted as $SProj$. We denote the PCA and rotation steps $Reduce(Proj, pca)$, where pca is the trained PCA.

2) **Node Creation**: Create a node for each of the densest parts of the above two-dimensional space [17]. In practice, we perform a radial scan of $SProj_{smpl}$. For each radius, we collect the intersection with the trajectories of $SProj_{smpl}$, and we apply kernel density estimation on the intersected points: each local maximum of the density estimation curve is assigned to a node. These nodes can be seen as a summarization of all the major length patterns ℓ that occurred in $\mathcal{D}$. For this step, we only consider the sampled collection of subsequences $SProj_{smpl}$. We denote this step $NodeCr(SProj_{smpl})$ (Line 7).

3) **Edge Creation**: Retrieve all transitions between pairs of subsequences represented by two different nodes: each transition corresponds to a pair of subsequences, where one occurs immediately after the other in a time series T of the dataset $\mathcal{D}$. We represent transitions with an edge between the corresponding nodes. We note this step $EdgeCr(SProj, \mathcal{N}_\ell)$ (Line 7). $\mathcal{N}_\ell$ is the node set extracted in the above step.

Despite the linear aspect of the PCA (in the subsequence embedding step), we observed in practice that using the first three components was sufficient to embed enough shape-based information [17]. Furthermore, PCA is renowned for its ability to identify and emphasize principal patterns while diminishing the impact of noise [37], [38]. The latter guarantees a graph-based representation less affected by the presence of noise in the time series subsequences.

As mentioned above, the graph embedding requires a parameter $smpl$ that controls the number of subsequences used to train the PCA and to create the nodes. We set by default the parameter $smpl = 10$ and evaluate its influence in Section IV. Moreover, we generate the graph embedding using Algorithm 2 for M different lengths randomly selected from a predefined interval. We denote the set of randomly selected lengths as R, where the user can specify the number of lengths by setting $M = |R|$. Our observations show that accuracy and execution time tend to improve with an increase in M. However, the accuracy reaches a plateau after $M = 30$, as demonstrated in Section IV.

Since the subsequence embedding step necessitates subsequences of at least 5 points, the length interval can be con-

strained to $[5, \min_{T \in \mathcal{D}} |T|]$. However, empirical observations indicate that this interval can be further optimized. We define this interval as $[5, (\min_{T \in \mathcal{D}} |T|) \times rml]$, where rml (rate maximum length) is a user-defined parameter with a default value of 0.4 (evaluated in Section IV).

Figure 3, we show the graphs for 3 different lengths (ℓ equal to 10, 30, and 50) for the TwoLeadECG dataset of the UCR-Archive [23]. The variation in the graph's topology emphasizes the significance of subsequence length. Building multiple graphs ensures that k-Graph does not rely on a predetermined subsequence length, thus avoiding situations where an erroneous or unsuitable choice of the single subsequence length leads to a suboptimal graph, potentially failing to capture critical data patterns.

B. Graph Clustering

At this stage, we have M distinct graphs corresponding to M different subsequence lengths. To perform a clustering partition of the dataset $\mathcal{D}$ using these graphs, we opt for a feature-based approach for execution time and interpretability. Our goal is to maximize information extraction from the graph. For this purpose, we extract three types of features for each time series: (i) node-based, (ii) edge-based features, and (iii) degree-based features. Specifically, for two first types of features, we quantify how many times a given time series intersects each node and edge in the graph. For the third type, we measure the degree for each node in the subgraph corresponding to each time series. For example, the same node can have a high degree for one time series (corresponding to a specific subgraph in $\mathcal{G}_\ell$) and a low degree for another time series (corresponding to another subgraph). Formally, for a given dataset $\mathcal{D}$ and its graph embedding $\mathcal{G}_\ell = (\mathcal{N}_\ell, \mathcal{E}_\ell)$ computed using a subsequence length ℓ, we define for a time series $T \in \mathcal{D}$, $\mathcal{M}_\ell(T) = \langle N^{(1)}, N^{(2)}, ..., N^{(m)} \rangle$ the path of T in $\mathcal{G}_\ell$, and $\mathcal{G}_\ell(T)$ the corresponding subgraph. Thus, we define the feature matrix $F_{\mathcal{G}, \ell} \in \mathbb{R}^{(|\mathcal{D}|, |\mathcal{N}_\ell| + |\mathcal{E}_\ell|)}$ as follows:

$$\forall T_i \in \mathcal{D}, F_{\mathcal{G}, \ell}[i] = [f_{T_i, N_1}, ..., f_{T_i, N_{|\mathcal{N}_\ell|}}, f_{T_i, E_1}, ..., f_{T_i, E_{|\mathcal{E}_\ell|}},$$
$$f^d_{T_i, N_1}, ..., f^d_{T_i, N_{|\mathcal{N}_\ell|}}]$$

With $\forall N, E \in \mathcal{N}_\ell, \mathcal{E}_\ell$:

$$f_{T_i, N} = \sum_{N^{(j)} \in \mathcal{M}_\ell(T_i)} 1_{[N = N^{(j)}]}$$

$$f_{T_i, E} = \sum_{(N^{(j)}, N^{(j+1)}) \in \mathcal{M}_\ell(T_i)} 1_{[E = (N^{(j)}, N^{(j+1)})]}$$

$$f^d_{T_i, N} = deg_{\mathcal{G}_\ell(T)}(N) * 1_{[N = (N^{(j)}, N^{(j+1)})]}$$

In the equation above, $deg_{\mathcal{G}_\ell(T)}(N)$ denotes the degree of the node N within the subgraph $\mathcal{G}_\ell(T)$. As described above, $F_{\mathcal{G}, \ell}$ is a sparse matrix with values greater than zero only if a time series crosses a node or an edge. However, as time series can be of different lengths, the number of nodes and edges crossed can vary significantly from one dataset instance to another. As a consequence, we normalize each row $F_{\mathcal{G}, \ell}[i]$ by subtracting the its mean $\mu(F_{\mathcal{G}, \ell}[i])$ and dividing by its standard deviation $\sigma(F_{\mathcal{G}, \ell})$. We can then apply any clustering algorithm on $F_{\mathcal{G}, \ell}$.

We use k-Means with Euclidean distance to produce a partition $\mathcal{L}_\ell$ for a given graph $\mathcal{G}_\ell$. Normalization is applied only during the clustering step, while the unnormalized feature
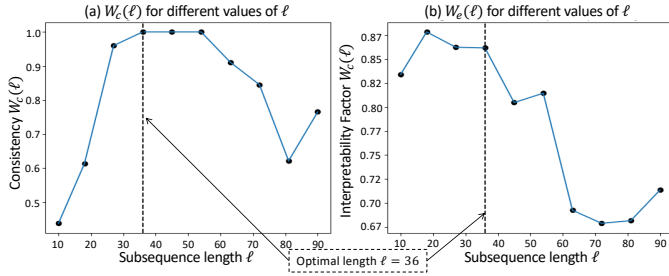
Fig. 4: W_c and W_e for k-Graph (with $k = 4$ and $M = 10$) applied on the Trace dataset of the UCR-Archive.

matrix is retained for computing graphoids. The choice of k-Means is motivated by its scalability for large datasets and its centroid-based approach, which aids interpretability. The centroids from graph clustering are crucial for computing graphoids, which are key to interpreting the clustering results of k-Graph (see Section III-D).

C. Consensus Clustering

At this point, we have one clustering partition $\mathcal{L}_\ell$ per graph built in the graph embedding step (M in total, for ℓ in the set of randomly selected lengths R). The objective is to compute a consensus from all these partitions. The problem of consensus clustering, or ensemble clustering, is a well-studied problem with several methods proposed in the literature [39], [40].

Following established practice [39], we build a consensus matrix, which we employ to measure how many times two time series have been grouped in the same cluster for two graphs built with two different lengths. Formally, we define $M_C \in \mathbb{R}^{(|\mathcal{D}|,|\mathcal{D}|)}$ a matrix computed as follows:

$$\forall T_i, T_j \in \mathcal{D}, M_C[i,j] = \frac{1}{|R|} \sum_{\ell \in R} 1_{[\mathcal{L}_{\ell,i} = \mathcal{L}_{\ell,j}]} \tag{1}$$

M_C can be seen as a similarity matrix about the clustering results obtained on each graph. More specifically, for two time series T_i and T_j, if $M_C[i,j]$ is high, they have been associated in the same cluster for several subsequence lengths and can be grouped in the same cluster. On the contrary, if $M_C[i,j]$ is low, the two time series were usually grouped in different clusters regardless of the subsequence length. Therefore, the M_C matrix can be seen as the adjacency matrix of a graph. In this graph, nodes are the time series of the dataset, and an edge exists if two time series have been clustered together in the same cluster (the weights of these edges are the number of times these two time series have been clustered together). As the objective is to find communities of highly connected nodes (i.e., time series that were grouped multiple times in the same cluster), we use spectral clustering (with M_C used as a pre-computed similarity matrix). The output of the spectral clustering is the final labels $\mathcal{L}$ of k-Graph.

D. Interpretability and Explainability

The previous section elucidated k-Graph's approach to the time series clustering problem. Now, we delve into the steps necessary for solving the interpretable clustering problem. As outlined in Section II-D, we formalized the interpretability problem through the computation of graphoids. The challenge lies in the assumption that there is a unique graph. However,

k-Graph operates with M graphs to produce one dataset partition. We are presented with two options: (i) compute the graphoids for each graph, or (ii) select one graph that is most representative of the final labeling $\mathcal{L}$.

Considering option (i), we assume equal contributions from each graph to the labels $\mathcal{L}$. However, this assumption falters due to the consensus clustering step. For instance, among the M graphs, one-third might generate one type of partition, while the remaining two-thirds yield another. Consequently, the partition from the latter two-thirds would be selected as the final labels $\mathcal{L}$ of k-Graph. As a result, one-third of the graphs might be irrelevant for the final labels, yielding graphoids that do not correctly interpret the clustering.

Hence, we opt for option (ii), choosing one of the M graphs: the one that is most relevant to the clustering labels $\mathcal{L}$ produced by k-Graph. To make this selection, we employ two criteria: the consistency of $\mathcal{L}_\ell$ with $\mathcal{L}$, and the interpretability factor of $\mathcal{G}_\ell$ associated with $\mathcal{L}$.

[Consistency] To gauge the consistency between the final labels $\mathcal{L}$ and those generated by each graph $\mathcal{G}_\ell$ (for $\ell \in R$), we utilize the Adjusted Rand Index (ARI). The latter is particularly suitable for unbalance datasets as it accounts for expected similarity due to chance (See Section IV). Formally, the consistency is expressed as $W_c(\ell) = ARI(\mathcal{L}, \mathcal{L}_\ell)$, measuring the agreement between the clustering labels produced by k-Graph and those corresponding to a specific subsequence length ℓ. This index provides a quantitative similarity measure, with higher values indicating greater consistency.

For a practical illustration, Figure 4(a) displays the values of W_c across different subsequence lengths (ℓ) in the range $R = [10, 28, 27, 36, 45, 54, 63, 72, 81, 90]$. In this example, k-Graph, configured with $k = 4$ and $M = 10$, is applied to the Trace dataset from the UCR-Archive. The graph generated with subsequence lengths between 27 and 54 demonstrates high consistency (ARI values above 0.9) with the final labels $\mathcal{L}$, indicating that these lengths are more suitable for interpretation. Conversely, other lengths result in lower consistency, suggesting that labels produced under these conditions are less aligned with the final clustering labels of k-Graph. Therefore, for optimal interpretability, it is advisable to consider subsequence lengths within the range of 27 to 54.

[Interpretability Factor] The second criterion focuses on selecting the most interpretable graphs in tandem with a given clustering partition. As elucidated in Section II-D, interpretability is achieved by ensuring that each cluster has precisely one node, characterized by $|N|_{C_i} = Pr_{C_i}(N) = 1$. To quantify interpretability, we calculate the interpretability factor as the maximum $Pr_{C_i}(N)$ across all nodes N in the graph $\mathcal{G}_\ell$. Formally, let $\mathcal{D}$ be a dataset, and $\mathcal{G}_\ell = (\mathcal{N}_\ell, \mathcal{E}_\ell)$ be its graph embedding computed with a subsequence length ℓ. We denote $C = C_1, ..., C_k$ as the clustering partition associated with labels $\mathcal{L}$ obtained using k-Graph. The interpretability factor $W_e(\ell)$ of $\mathcal{G}_\ell$ is defined as:

$$W_e(\ell) = \frac{1}{k} \sum_{C_i \in C} \max_{N \in \mathcal{N}_\ell} Pr_{C_i}(N) \tag{2}$$

In practical terms, the $\mathcal{M}$ function in Definition 4 is instantiated as follows: for a time series $T \in \mathcal{D}$, $\mathcal{M}_\ell(T) = \langle N^{(1)}, N^{(2)}, ..., N^{(m)} \rangle$, representing the path of T in $\mathcal{G}_\ell$.

Figure 4(b) illustrates W_e for the Trace dataset of the UCR-Archive, revealing that graphs computed with shorter lengths exhibit a higher interpretability factor. To select the optimal length $\bar{\ell}$ and, consequently, the graph $\mathcal{G}_\ell$ that maximizes both W_c and W_e, we employ the following criterion:

$$\bar{\ell} = \operatorname*{argmax}_{\ell \in R} \left[W_e(\ell).W_c(\ell) \right] \qquad (3)$$

In Figure 4, the sole length that maximizes the product $W_e(\ell) \cdot W_c(\ell)$ is $\bar{\ell} = 36$. Subsequently, the graph associated with this length is utilized to compute the graphoids.

To compute the graphoids, we use the feature matrix $F_{G,\bar{\ell}}$ (described in Section III-B). In practice, we compute the representativity and the exclusivity of a node N in $\mathcal{G}_\ell$. We can compute the two previous measures for all nodes in $\mathcal{G}_\ell$, and then select only those that satisfy the property of λ-Graphoid and γ-Graphoid for a desired value of λ and γ. As one highly representative and exclusive node per cluster is enough, we automatically return one node $\bar{N}_{C_i}$ per cluster such that:

$$\bar{N}_{C_i} = \operatorname*{argmax}_{N \in \mathcal{N}_\ell} \left[|N|_{C_i}.Pr_{C_i}(N) \right] \qquad (4)$$

The node $\bar{N}_{C_i}$ is considered the most interpretable node for a given cluster C_i. Finally, for each node, we store the timestamps of the subsequences in $\mathcal{D}$ associated with it. Therefore, we compute the centroid of all the subsequences that compose $\bar{N}_{C_i}$. Figure 10 and Figure 12 show the most interpretable nodes (and the corresponding centroids) identified by k-Graph for several example datasets.

In our approach, interpretability (i.e., graphoid for each cluster) is integrated during the training phase rather than applied post-hoc. By embedding interpretability into the clustering process, we ensure that the selected graph consistently represents the clustering logic during both training and subsequent predictions. This ensures reliable interpretability, even when incoming time series exhibit variability or contain multiple relevant subsequences.

E. Complexity Analysis

This section analyzes the computational complexity of different steps in k-Graph. For the Graph Embedding step, the sequence embedding complexity is bounded by the PCA (using a randomized SVD solver) with complexity $O(|\mathcal{D}|^2 \cdot |T|^2)$. The node creation and edge creation steps are both $O(|\mathcal{D}| \cdot |T|)$ on average. Following the graph embedding step, the complexity for feature matrix creation and k-Means clustering is $O(|\mathcal{D}| \cdot (|\mathcal{N}| + |\mathcal{E}|))$ and $O(k.|\mathcal{D}| \cdot (|\mathcal{N}| + |\mathcal{E}|))$ respectively. In practice, the number of nodes and edges is significantly smaller than the number of subsequences in the dataset, making this step negligible compared to the graph embedding step. Finally, building the consensus matrix requires $O(M \cdot |\mathcal{D}|^2)$. As the consensus matrix is already built, spectral clustering requires, in the worst case, $O(|\mathcal{D}|^2)$. Overall, since the first two steps are executed M times (for M different subsequence lengths), the k-Graph complexity is $O(M \cdot |\mathcal{D}|^2 \cdot |T|^2)$. However, as the computations of the M graphs are independent, they can be executed in parallel, significantly reducing execution time. Moreover, the $smpl$ parameter significantly reduces $|\mathcal{D}|$. Section IV evaluates the influence of M and $smpl$ on execution time.

Dataset Type	# Dataset	Avg. # TS	Avg. Length	Avg. Classes
AUDIO	1	2110.00	1024.00	39.00
DEVICE	11	2428.18	809.27	3.55
ECG	7	2313.71	513.29	14.14
EOG	2	724.00	1250.00	12.00
EPG	2	288.50	601.00	3.00
H.DYN.	3	312.00	2000.00	52.00
IMAGE	31	1713.81	338.45	10.61
MOTION	6	343.83	899.00	3.83
OTHER	1	204.00	201.00	18.00
SENSOR	14	2213.64	348.79	3.07
SIMUL.	9	1566.33	255.44	3.89
SOUND	11	1982.27	308.18	6.91
SPECTRO	12	448.17	980.75	3.42
TRAFFIC	1	365.00	24.00	2.00

TABLE II: UCR-Archive Dataset category Statistics

IV. EXPERIMENTAL EVALUATION

[Setup] We implemented our algorithms in Python 3.9. The evaluation was conducted on a server Intel(R) Xeon(R) Gold 6242R CPU 3.10GHz (80 CPUs), with 512GB RAM. Our code is publicly available: https://github.com/boniolp/kGraph

[Datasets] We conducted an experimental evaluation utilizing real datasets from the UCR-Archive [23] to assess the classification performance of various methods. While our proposed approach is versatile and can be evaluated on variable-length time series, the baseline methods do not share this flexibility. Therefore, out of the initial 128 datasets, we excluded 15 containing variable-length time series. Our comparative analysis involves assessing our proposed approach against the baselines on a subset of 113 real-time series datasets. Table II summarizes the characteristics of the adopted datasets, which represent several different scenarios.

[Baselines] In our experimental evaluation, we consider the 15 state-of-the-art (SOTA) algorithm from each category:

- **Traditional Clustering:** We include the k-Means algorithm, which uses the Euclidean distance as a similarity measure. We also include eight additional traditional clustering approaches for completeness which are Mean-Shift [41] (M.Sh. in Table III), Gaussian Mixture [42] (G.M. in Table III), BIRCH [43], MiniBatch-k-Means [42], OPTICS [44], HDBSCAN [45], DBSCAN [46] and Agglomerative Clustering with a average linkage [42] (Aggl. in Table III). For all these baselines, we use their corresponding scikit-learn [42] implementation, and we select their default parameters.
- **Raw-based Approaches:** We select the k-Shape algorithm [25]. This algorithm identifies the most discriminative sub-shapes of the time series to form the final cluster. In addition, we adopt SomTimeS, an algorithm that uses a self-organizing map (SOM) framework to align and compare time series with Dynamic Time Warping (DTW) as the distance measure. It enhances computational efficiency through a pruning strategy that reduces the number of DTW calculations during training.
- **Symbolic Approaches:** We select the Symbolic Pattern Forest algorithm (SPF) [47]. The approach checks if some randomly selected symbolic patterns exist in the time series to partition the dataset. This partition process is executed multiple times, and the ensemble combines the partitions to generate the final partition.
- **Features-based Approaches:** We choose Time2Feat [22] (T2F in Table III). This approach extracts and selects
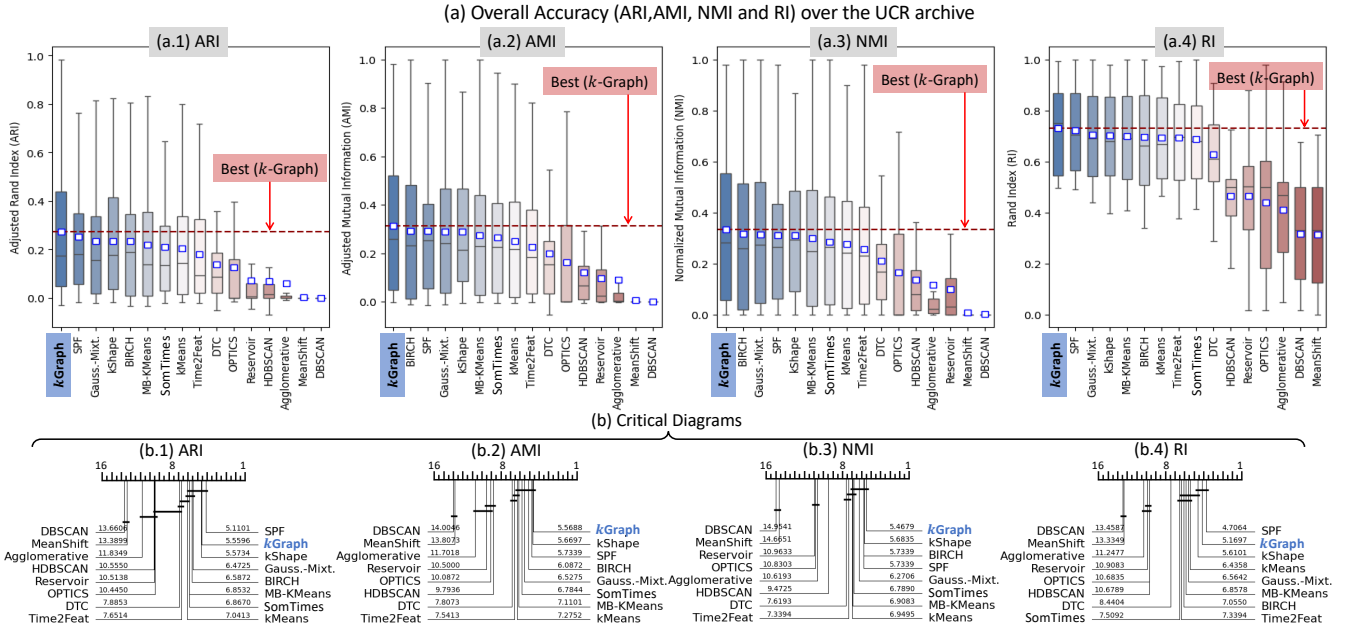
Fig. 5: Experimental comparison of k-Graph versus the baselines on the UCR archive. In (a), the mean values are represented as a white square. The horizontal red dotted line represents the best mean.

the best features from the time series, providing a fully interpretable and effective solution.

- **Deep Learning Approaches:** We opt for two baselines. First, the Deep Temporal Clustering (DTC) algorithm [34]. DTC uses feature representation extracted from the Deep Auto-Encoder and the Kullback-Leibler (KL) divergence objective for self-labeling of the dataset. The second baseline considered is the Reservoir Model Space [48] (called Reservoir in the rest of the paper and Res. in Table III). Reservoir Computing (RC) is a family of Recurrent Neural Network (RNN) models whose recurrent part is generated randomly and then kept fixed. The representation learned with Reservoir is then fed into a traditional clustering.

[Metrics] We compare k-Graph against state-of-the-art approaches using metrics that assess how well clusters align with predefined or expected data structures by comparing them with external information like labels. We focus on four metrics: the Adjusted Rand Index (ARI) [49], Adjusted Mutual Information (AMI), Normalized Mutual Information (NMI), and Rand Index (RI).

The Adjusted Rand Index (ARI), derived from the Rand Index (RI), measures the similarity between two clusterings by counting pairs of data points in the same or different clusters. Unlike the RI, the ARI accounts for expected similarity due to chance, which is crucial for accurate results, especially in imbalanced datasets. This correction provides a more reliable measure of clustering performance, with a scale from -0.5 to 1. A value of 1 indicates perfect agreement, 0 suggests chance-level agreement and negative values indicate disagreement. By considering class balance, the ARI offers a more robust assessment of clustering similarity compared to the RI. To enhance evaluation robustness, we also use Adjusted Mutual Information (AMI) and Normalized Mutual Information (NMI). Mutual Information (MI) measures the mutual dependence between two clusterings, but cluster sizes and numbers can influence it. NMI normalizes MI to a scale
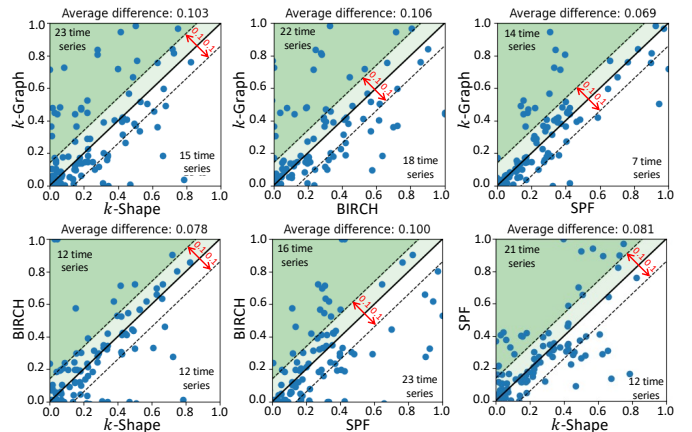

Fig. 6: Pairwise comparisons (ARI) of the top performing methods, namely, k-Graph, k-Shape, SPF, and BIRCH.

from 0 (no mutual information) to 1 (perfect correlation). AMI further adjusts MI for chance, providing a normalized measure that is useful for comparing clustering results across datasets with varying cluster sizes.

A. Accuracy Evaluation

Presentation Table III showcases the performance of k-Graph compared to state-of-the-art approaches, expressed in terms of Adjusted Rand Index (ARI). The datasets are categorized as introduced in Section IV. We also include the average ARI, NMI, AMI, and RI (for the entire UCR-Archive), as well as the number of wins and the average rank (for each measure). The results demonstrate k-Graph's superior performance across various dataset types, achieving the best performance in 5 out of 13 cases and the second-best results in 2 out of 8 cases. Moreover, we observe that k-Graph outperforms the baselines on average across the entire UCR-Archive for all measures. We also note that k-Graph has the best average rank for AMI and

	k-**Graph**	k-Shape	SPF	BIRCH	G.M.	MB-k-M.	k-Means	SomTimes	T2F	DTC	OPTICS	Reservoir	Aggl.	HDBSC.	M.Sh.	DBSC.
	Averaged ARI per category of the UCR-Archive datasets															
AUDIO	0.226	0.223	0.262	**0.392**	<u>0.386</u>	0.329	0.343	0.074	0.118	0.320	0.000	0.000	-0.000	0.012	-0.000	0.000
DEVICE	0.101	0.072	0.080	0.063	<u>0.108</u>	0.094	0.058	**0.117**	0.083	0.058	0.007	0.016	0.003	0.015	0.000	0.001
ECG	<u>0.315</u>	**0.363**	0.248	0.285	<u>0.279</u>	0.243	0.251	0.224	0.166	0.190	0.112	0.082	0.051	0.011	0.000	0.000
EOG	<u>0.129</u>	0.152	0.126	0.156	**0.188**	0.163	<u>0.170</u>	0.146	0.139	0.139	0.000	0.065	0.000	0.021	0.000	0.000
EPG	0.445	0.035	0.416	**1.000**	**1.000**	**1.000**	0.182	**1.000**	0.538	0.109	<u>0.772</u>	0.759	**1.000**	**1.000**	0.000	0.000
H.DYN.	**0.373**	0.037	<u>0.235</u>	0.109	0.092	0.087	0.043	0.066	0.048	-0.000	<u>0.001</u>	0.000	0.018	0.005	0.000	0.000
IMAGE	0.290	<u>0.297</u>	<u>0.292</u>	**0.299**	0.283	0.249	0.259	0.238	0.191	0.176	0.157	0.105	0.024	0.064	0.014	-0.000
MOTION	**0.170**	<u>0.154</u>	<u>0.156</u>	0.155	0.123	0.138	0.141	0.124	0.090	0.139	0.048	0.033	0.034	0.050	0.000	0.000
OTHER	**0.945**	0.414	<u>0.308</u>	<u>0.713</u>	0.627	0.668	0.708	0.212	0.478	0.295	0.242	0.000	0.705	0.056	0.000	0.000
SENSOR	**0.352**	0.283	<u>0.337</u>	<u>0.224</u>	0.238	0.215	0.234	0.226	0.191	0.174	0.166	0.074	0.097	0.093	0.001	-0.005
SIM.	<u>0.403</u>	0.353	<u>0.376</u>	0.315	0.351	0.274	0.261	0.292	**0.418**	0.209	0.110	0.113	0.070	0.034	0.000	0.000
SOUND	**0.044**	0.035	<u>0.041</u>	0.004	0.011	0.010	0.016	0.031	0.028	0.000	0.000	0.016	0.001	0.000	0.000	0.000
SPECTRO	0.208	0.167	**0.234**	0.133	0.130	0.172	<u>0.212</u>	0.170	0.171	0.063	0.157	-0.001	0.020	0.054	0.000	0.000
TRAFFIC	0.492	0.676	0.136	0.461	<u>0.749</u>	**0.757**	<u>0.047</u>	0.260	-0.018	0.054	0.000	0.000	0.008	0.098	0.008	0.000
	Averaged Accuracy for each dataset of UCR-Archive															
RI	**0.729**	0.702	<u>0.721</u>	0.697	0.707	0.700	0.695	0.688	0.693	0.630	0.438	0.467	0.411	0.464	0.314	0.317
ARI	**0.275**	0.237	<u>0.252</u>	0.238	0.239	0.223	0.208	0.209	0.181	0.145	0.124	0.075	0.061	0.069	0.004	-0.001
AMI	**0.315**	0.291	<u>0.291</u>	0.296	<u>0.293</u>	0.278	0.256	0.264	0.230	0.204	0.161	0.098	0.093	0.119	0.005	0.001
NMI	**0.335**	0.312	0.312	<u>0.320</u>	<u>0.317</u>	0.302	0.284	0.259	0.217	0.165	0.102	0.118	0.133	0.006	0.001
	Number of win on the UCR-Archive															
RI	<u>22</u>	8	**26**	8	4	4	6	5	8	4	2	1	1	2	0	1
ARI	**25**	11	<u>17</u>	9	5	7	8	6	8	4	3	1	0	2	0	0
AMI	**21**	11	<u>12</u>	<u>13</u>	12	1	7	4	7	3	7	4	0	3	1	0
NMI	**21**	12	12	<u>13</u>	11	1	6	4	7	3	7	4	0	4	1	0

TABLE III: Effectiveness (ARI, RI, AMI, and NMI) per category of the UCR-Archive, the average (of measures) across all UCR-Archive, and the number of wins (i.e., the number of time methods achieved best accuracy) on the UCR-Archive. In bold, the best value per dataset type. The underlined represents the second best result.

ARI and is second best for RI and ARI. Finally, k-Graph has the most significant number of wins (i.e., number of times k-Graph has the best performance) for ARI, AMI, and NMI, and the second largest number of wins for RI. Figure 5(a) presents ARI, AMI, NMI, and RI across all datasets, demonstrating k-Graph's overall improvement compared to other approaches. Figure 5(b) shows the critical difference diagram using a pairwise Wilcoxon sign rank test, with methods ranked along the horizontal axis. Figure 6 compares k-Graph's ARI with top-performing methods (k-Shape, SPF, and BIRCH). Each point represents a dataset's performance between two methods, with the green area indicating superior performance for y-axis methods and the white area for x-axis methods. Dotted lines mark 0.1 performance difference intervals, with points outside these bands indicating significant performance differences. The average distance from the identity line indicates the magnitude of performance differences between methods.

Discussion The results indicate an enhancement in accuracy with the implementation of k-Graph. This improvement is evident when evaluating performance across different dataset types and considering the overall average across all datasets. Nevertheless, we observe in Figure 5(b) that, across all datasets of the UCR-Archive taken individually, k-Graph outperforms the baselines for NMI and AMI, albeit not significantly. For ARI and RI, SPF is slightly above k-Graph. It is also important to note that some of the traditional baselines demonstrate strong performances (across all measures) although these approaches were specifically proposed for time series. Nevertheless, among the top performing methods (ranked first or second in Figure 5(a)), 3 approaches on 4 have been proposed for time series clustering. Moreover, we observe that k-Graph performance is lower than the baselines for the TRAFFIC category. The latter is composed of one dataset (Chinatown) that contains very short time series (24 data points). As k-Graph relies on extracting meaningful subsequences, such dataset with very short time series is particularly hard to handle. Consequently, we can empirically observe a potential limitation of k-Graph for very short time series.

Furthermore, the comparison illustrated in Figure 6 with k-Shape provides intriguing insights. It unveils scenarios where
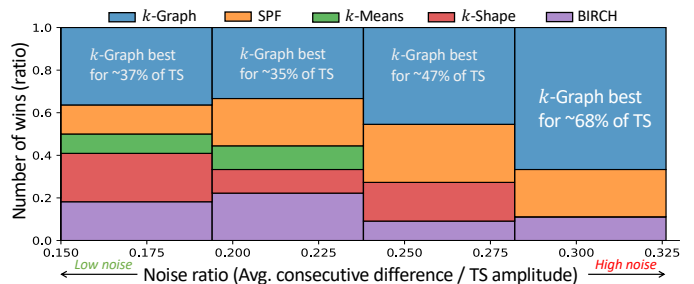

Fig. 7: Number of win (ratio) versus Noise (ratio) in the UCR-Archive

k-Graph outperforms k-Shape, SPF, BIRCH, and vice versa, highlighting each approach's distinctive strengths and weaknesses when applied to diverse datasets. The latter emphasizes two key points: (i) the importance of an adaptable and robust approach like k-Graph that meets the objectives outlined in the initial challenge across various datasets, and (ii) the complementary nature of k-Graph, k-Shape, and BIRCH, suggesting potential synergies for improved accuracy.

We investigate the performance differences between k-Graph and the four best-performing baselines by assessing the impact of noise on accuracy. The noise level of datasets in the UCR-Archive is measured by the average noise ratio, defined as the ratio of the average difference between consecutive points to the maximum amplitude of the time series. We calculate the number of wins (i.e., instances where a method achieves the best ARI score) for datasets within predefined noise ratio intervals. As shown in Figure 7, the proportion of k-Graph wins increases with the noise ratio, reaching 68% for time series with a noise ratio above 0.28. Overall, Figure 7 indicates that k-Graph's efficiency is unaffected by noise and is more robust than the baselines.

B. Execution Time Evaluation

Presentation We compare our method's execution time against other approaches, all configured for a multi-core CPU setup. To ensure consistency, we focus on methods with pure Python implementations, including Time2Feat, k-Shape, and DTC, but also include k-Means (implemented in C) for a comprehensive evaluation. Figure 8 shows that k-Graph, k-Shape, and DTC
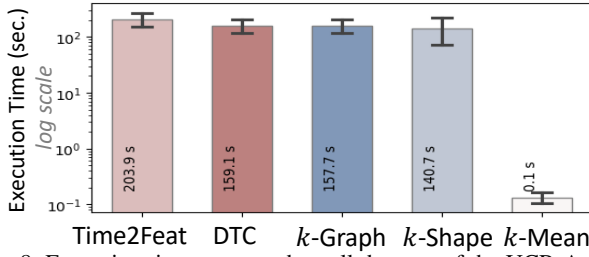
Fig. 8: Execution time computed on all datasets of the UCR-Archive.
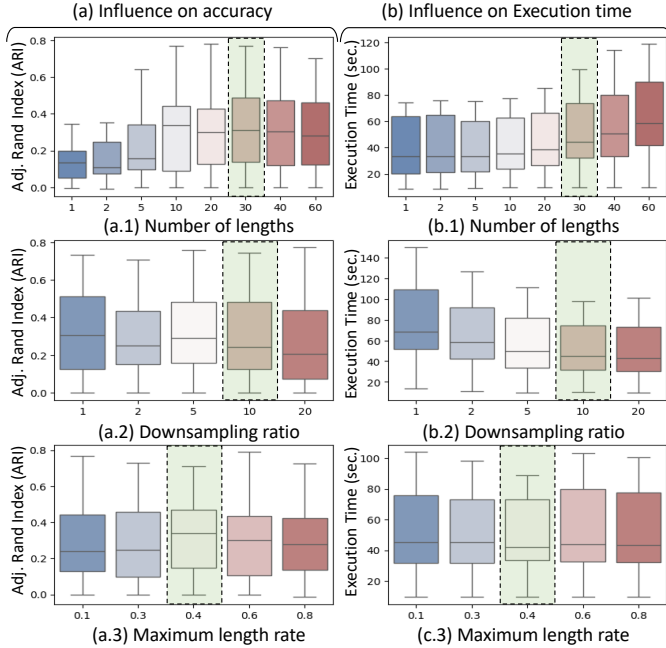

Fig. 9: Influence of k-Graph parameters on (a) accuracy and (b) execution time (green hover the default values of these parameters).

Fig. 10: k-Graph interpretability for the Trace dataset from the UCR-Archive. The highlighted nodes are the most representative and exclusive subsequences identified by k-Graph for each cluster.

have similar execution times, differing by about 10 seconds. Time2Feat is the slowest, taking 50 seconds longer, while k-Means is the fastest, likely due to its C implementation. *Discussion* The observed results indicate that extracting numerous graphs does not compromise the algorithm's efficiency, as it maintains competitive performance compared to widely adopted methods, as required by the first challenge. However, it is noteworthy that k-Means, employing a straightforward solution, outperforms other approaches regarding speed.

C. Parameter Influence

Presentation As introduced in Section III, k-Graph has three main parameters: Number of Lengths (M) for randomly selected subsequence lengths, Sample ($smpl$) to reduce subsequences in PCA training, and Rate Maximum Length (rml) for the upper limit of subsequence length selection. Figure 9 demonstrates parameter impacts across UCR Datasets, showing effectiveness (ARI) in column (a) and efficiency (seconds) in column (b). The rows show variations in M (eight values), $smpl$ (five values), and rml (five values) parameters respectively. For consistency, default values of $M = 30$, $smpl = 10$, and $rml = 0.4$ are maintained when not studied. *Discussion* The experimentation with the parameter M reveals that increasing the number of lengths positively impacts accuracy until it reaches a plateau at around $M = 30$ lengths.
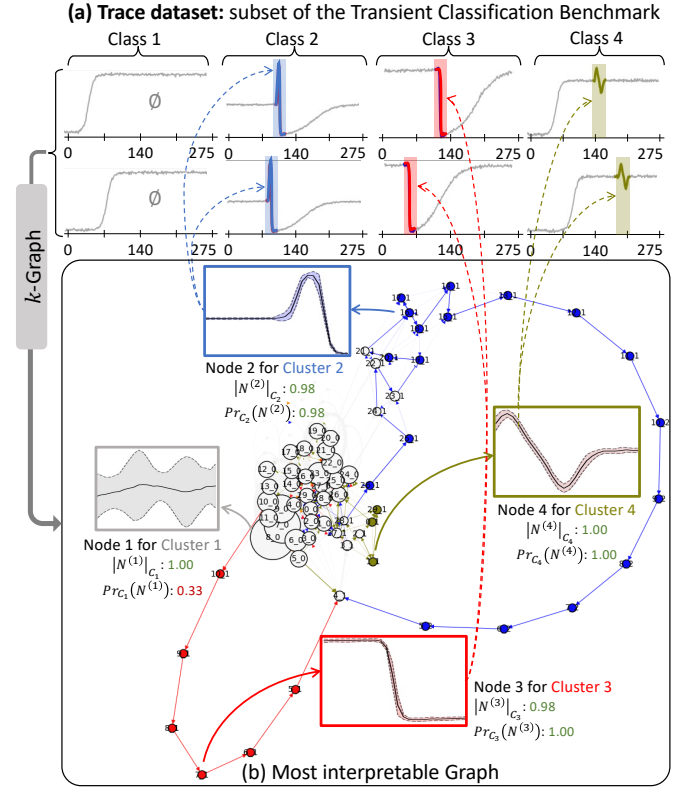
Beyond this point, the accuracy shows little improvement. Correspondingly, as the lengths increase, the time required by the algorithm also increases, demonstrating a trade-off between accuracy and computational efficiency. A favorable balance between execution time and performance is achieved with approximately 30 different lengths. Examining the $smpl$ parameter indicates that reducing the number of subsequences minimally impacts performance until around 10. However, it significantly affects the algorithm's execution time. As we decrease the number of subsequences, the time performance improves. Achieving a good balance involves reducing the number of subsequences by 10 times. Finally, the analysis of the rml parameter shows that adopting $40\%(0.4)$ of the length of the shortest series in the dataset as the maximum subsequence length is the optimal solution. This choice provides similar execution times while demonstrating superior performance in terms of effectiveness.

D. k-Graph Interpretability

Presentation In this section, we explore how k-Graph enhances the interpretability of clustering results compared to k-Shape. While k-Shape and k-Graph differ in interpretability—k-Graph offers a graph-based, subsequence-based interpretation, whereas k-Shape provides centroids of the same size as the dataset's time series—we still present k-Shape's centroids to highlight the interpretability advantage of k-Graph. Figure 10 provides visual insights using the Trace dataset from the UCR-Archive, which contains simulated time series representing various instrument failures in nuclear power plants (four
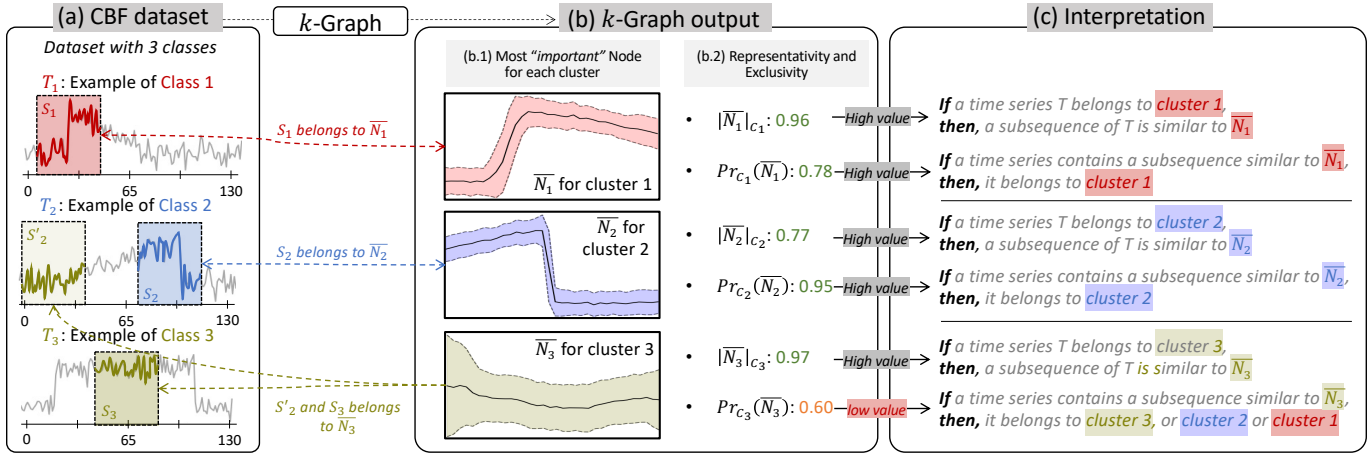
Fig. 11: Interpretability of k-Graph clustering applied to the CBF dataset.

classes). In this figure, we observe the graph $\mathcal{G}_{\bar{\ell}}$ for the optimal length $\bar{\ell} = 36$, along with the most representative and exclusive nodes for each cluster ($\bar{N}_{C_1} = N^{(1)}$, $\bar{N}_{C_2} = N^{(2)}$, $\bar{N}_{C_3} = N^{(3)}$, $\bar{N}_{C_4} = N^{(4)}$). The colored nodes in Figure 10 represent those with an exclusivity above 0.9 (i.e., the γ-graphoids with $\gamma = 0.9$).

Discussion In Figure 10, most nodes represent subsequences from all clusters (i.e., grey nodes), but some, like Node $N^{(2)}$, show high representativity and exclusivity. This figure highlights k-Graph's interpretability capabilities, showing that its clustering partition aligns with the labels of the Trace dataset. The graphoids effectively identify the discriminant features of clusters 2, 3, and 4. Notably, Node $N^{(4)}$ has $|N^{(4)}|_{C_4} = 1$ and $Pr_{C_4}(N^{(4)}) = 1$, indicating it is crossed only by time series from cluster 1.

For cluster 1, the distinctive characteristic is the absence of discriminative patterns, as indicated by Node $N^{(1)}$ with $|N^{(1)}|_{C_1} = 1$ and $Pr_{C_1}(N^{(1)}) = 0.33$. While this node is highly representative, its low exclusivity suggests that cluster 1 lacks specific patterns, which aligns with the unique nature of class 1 in the Trace dataset. This ability to detect clusters without discriminant subsequences highlights k-Graph's robustness in handling diverse data structures. The representativity and exclusivity measures are crucial for users to evaluate the validity of identified nodes, providing a mechanism to filter out potential misinterpretations and ensuring that the clustering results are both meaningful and actionable.

E. Interpretability in Practice

Even though the graph can be important for the user to interact and explore the clustering result, the graph is not necessary for interpreting the clustering partition. In practice, we only need to return the most "important" nodes (i.e., with the highest representativity and exclusivity).

Figure 11 depicts the k-Graph output on the CBF datasets (illustrated in Figure 11(a)). k-Graph can automatically select the most representative and exclusive node for each cluster (illustrated in Figure 11(b)). Each node is associated with its corresponding exclusivity ($Pr_{C_i}(N)$) and representativity ($|N|_{C_i}$) values. These values give a direct interpretation of the clustering. For example, the high representativity ($|\bar{N}_1|_{C_1} = 0.96$) and high exclusivity ($Pr_{C_1}(\bar{N}_1) = 0.78$) of $\bar{N}_1$ imply
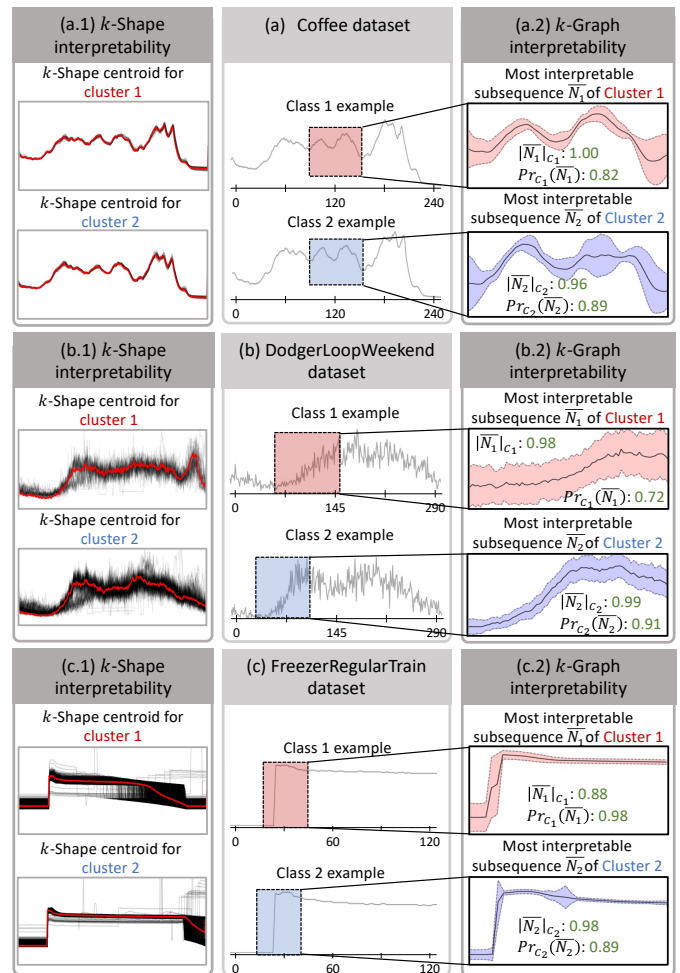
Fig. 12: Comparisons of k-Graph and k-Shape interpretability for each cluster obtained on three UCR-Archive datasets. (*.1) centroids (in red) computed using k-Shape. (*.2) most representative and exclusive node $\bar{\mathcal{N}}$.

the following interpretation: *"A time series belongs to cluster 1 if and only if a subsequence of this time series is similar to $\bar{N}_1$"*. We can see indeed in Figure 11(a) that a subsequence of T_1 (called S_1) is similar to $\bar{N}_1$.

On the contrary, the high representativity ($|\bar{N}_3|_{C_3} = 0.96$) and medium exclusivity ($Pr_{C_3}(\bar{N}_3) = 0.60$) of $\bar{N}_3$ imply the

following interpretation: *"A time series belongs to cluster 3 if a subsequence of this time series is similar to $\bar{N}_3$. However, a time series T containing a subsequence similar to similar to $\bar{N}_3$ does not necessarily mean that T belongs to cluster 3"*. Figure 11(a) confirms that a subsequence of T_2 and T_3 (called S_2' and S_3 respectively) are similar to $\bar{N}_3$. However, T_2 belongs to cluster 2, and T_3 belongs to cluster 3.

In general, k-Graph helps us to understand that the CBF dataset is composed of (i) time series with a rapid increase followed by a slow decrease (class 1 matched by cluster 1), (ii) time series with a slow increase followed by a rapid decrease (class 2 matched by cluster 2), and (iii) time series that do not contains rapid increase or decrease, and that contain a flat subsequence (class 3 matched by cluster 3).

F. Interpretability Examples

Presentation In Figure 12, we provide comparisons between k-Shape's and k-Graph's interpretability on three additional datasets of the UCR-Archive: (a) Coffee dataset with two classes, (b) DodgerLoopWeekend datasets with two classes, (c) FreezerRegularTrain datasets with two classes. For k-Shape, interpretability is driven by inspecting the centroids of each cluster. For k-Graph, we provide the most representative and exclusive nodes (due to lack of space, we omit plotting the corresponding graphs) for each cluster obtained with k-Graph.

Discussion Figure 12 highlights the most representative nodes for each cluster in four UCR-Archive datasets. In all cases, k-Graph provides clustering consistent with the labels. For DodgerLoopWeekend (Figure 12(b.2)),the nodes clearly correspond to class-specific patterns.

In contrast, the Coffee and FreezerRegularTrain datasets (Figure 12(a) and (c)) present more subtle patterns. In the Coffee dataset, the nodes reveal that class differences lie in the relative heights of two central peaks.

For FreezerRegularTrain, despite similar time series, k-Graph identifies nodes that highlight subtle differences in the initial increase, with cluster 2 showing a more abrupt rise and longer plateau. These nodes are highly representative and exclusive for their clusters ($|\bar{N}_1|_{C_1} = 0.88$, $|\bar{N}_2|_{C_2} = 0.98$ and $Pr_{C_1}(\bar{N}_1) = 0.98$, $Pr_{C_2}(\bar{N}_2) = 0.89$), aligning with discriminative features noted in the UCR-Archive [23]. These examples demonstrate the effectiveness of k-Graph in identifying relevant patterns for complex tasks.

Unlike k-Graph, the centroids computed with k-Shape often lack sufficient information to distinguish between classes. In examples, such as Coffee, DodgerLoopWeekend, and FreezerRegularTrain, distinguishing between classes is challenging. Additional comparisons on UCR-Archive datasets are available in our GitHub repository. This analysis suggests that k-Graph offers improved interpretability over previous solutions.

V. CONCLUSIONS

Despite significant interest in time series clustering, many existing methods lack interpretability. We introduce k-Graph, a novel graph-based approach that clusters time series datasets while providing interpretable clustering partitions. Our experiments show that k-Graph effectively addresses the challenges outlined in Section I. It combines the strengths of existing methods by preserving temporal dependencies and achieving performance on par with raw and deep learning approaches. Additionally, it enhances interpretability without sacrificing granularity, a common issue with feature-based methods.

In future work, we aim to explore the use of Graph Neural Networks in our clustering process and extend our approach to multivariate time series. We note that applying k-Graph to multivariate time series is not straight-forward. The main challenge in doing so is due to the inter-dependencies among dimensions, which should be considered in the graph embedding step. This would lead to a potential explosion of the graph size. We consider this problem non-trivial, and a very interesting future research direction.

VI. ACKNOWLEDGEMENTS

REFERENCES

[1] T. Palpanas and V. Beckmann, "Report on the first and second interdisciplinary time series analysis workshop (itisa)," *SIGMOD Rec.*, vol. 48, no. 3, p. 36–40, Dec. 2019.

[2] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: a survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[3] A. Bagnall, R. L. Cole, T. Palpanas, and K. Zoumpatianos, "Data Series Management (Dagstuhl Seminar 19282)," *Dagstuhl Reports*, vol. 9, no. 7, pp. 24–39, 2019.

[4] T. Palpanas, "Data series management: The road to big sequence analytics," *SIGMOD Rec.*, vol. 44, no. 2, p. 47–52, 2015.

[5] K. Zoumpatianos and T. Palpanas, "Data series management: Fulfilling the need for big sequence analytics," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, pp. 1677–1678.

[6] T. Palpanas, "Evolution of a data series index," in *ISIP*, 2019.

[7] B. Peng, P. Fatourou, and T. Palpanas, "Paris+: Data series indexing on multi-core architectures," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 5, pp. 2151–2164, 2021.

[8] Q. Wang and T. Palpanas, "Deep Learning Embeddings for Data Series Similarity Search," in *SIGKDD*, 2021.

[9] L. Ye and E. J. Keogh, "Time series shapelets: a novel technique that allows accurate, interpretable and fast classification," *Data Min. Knowl. Discov.*, vol. 22, no. 1-2, pp. 149–182, 2011.

[10] P. Schäfer and U. Leser, "Fast and accurate time series classification with WEASEL," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, 2017, pp. 637–646.

[11] C. W. Tan, F. Petitjean, and G. I. Webb, "Fastee: Fast ensembles of elastic distances for time series classification," *Data Min. Knowl. Discov.*, vol. 34, no. 1, pp. 231–272, 2020.

[12] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. Schmidt, J. Weber, G. Webb, L. Idoumghar, P. Muller, and F. Petitjean, "Inceptiontime: finding alexnet for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, p. 1936–1962, 2020.

[13] L. Ulanova, N. Begum, and E. J. Keogh, "Scalable clustering of time series with u-shapelets," in *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*, S. Venkatasubramanian and J. Ye, Eds., 2015, pp. 900–908.

[14] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," *SIGMOD Rec.*, vol. 45, no. 1, pp. 69–76, 2016.

[15] X. Li, J. Lin, and L. Zhao, "Linear time complexity time series clustering with symbolic pattern forest," in *IJCAI 2019, Macao, China, August 10-16, 2019*, 2019, pp. 2930–2936.

[16] J. Paparrizos, Y. Kang, P. Boniol, R. S. Tsay, T. Palpanas, and M. J. Franklin, "TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection," *PVLDB*, 2022.

[17] P. Boniol and T. Palpanas, "Series2graph: Graph-based subsequence anomaly detection for time series," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 1821–1834, 2020.

[18] Y. Gao, J. Lin, and C. Brif, "Ensemble grammar induction for detecting anomalies in time series," in *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, 2020, pp. 85–96.

[19] P. Boniol, M. Linardi, F. Roncallo, T. Palpanas, M. Meftah, and E. Remy, “Unsupervised and Scalable Subsequence Anomaly Detectionin Large Data Series,” *VLDBJ*, 2021.

[20] P. Boniol, J. Paparrizos, T. Palpanas, and M. J. Franklin, “SAND: streaming subsequence anomaly detection,” *Proc. VLDB Endow.*, vol. 14, no. 10, pp. 1717–1729, 2021.

[21] Y. Zhu, A. Mueen, and E. J. Keogh, “Matrix profile IX: admissible time series motif discovery with missing data,” *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 6, pp. 2616–2626, 2021.

[22] A. Bonifati, F. Del Buono, F. Guerra, and D. Tiano, “Time2feat: learning interpretable representations for multivariate time series clustering,” *Proceedings of the VLDB Endowment (PVLDB)*, vol. 16, no. 2, pp. 193–201, 2022.

[23] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. J. Keogh, “The ucr time series archive,” *IEEE/CAA Journal of Automatica Sinica*, vol. 6, pp. 1293–1305, 2018.

[24] T. W. Liao, “Clustering of time series data—a survey,” *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[25] J. Paparrizos and L. Gravano, “K-shape: Efficient and accurate clustering of time series,” *SIGMOD Rec.*, vol. 45, no. 1, p. 69–76, 2016.

[26] J. Paparrizos and S. P. T. Reddy, “Odyssey: An engine enabling the time-series clustering journey,” *Proc. VLDB Endow.*, vol. 16, no. 12, p. 4066–4069, Aug. 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611622

[27] N. Begum, L. Ulanova, J. Wang, and E. Keogh, “Accelerating dynamic time warping clustering with a novel admissible pruning strategy,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 49–58. [Online]. Available: https://doi.org/10.1145/2783258.2783286

[28] A. Javed, D. M. Rizzo, B. S. Lee, and R. Gramling, “Somtimes: self organizing maps for time series clustering and its application to serious illness conversations,” *Data Min. Knowl. Discov.*, vol. 38, no. 3, p. 813–839, Oct. 2023. [Online]. Available: https://doi.org/10.1007/s10618-023-00979-9

[29] X. Wang, K. Smith, and R. Hyndman, “Characteristic-based clustering for time series data,” *Data mining and knowledge Discovery*, vol. 13, pp. 335–364, 2006.

[30] D. Guijo-Rubio, A. M. Durán-Rosal, P. A. Gutiérrez, A. Troncoso, and C. Hervás-Martínez, “Time-series clustering based on the characterization of segment typologies,” *IEEE Transactions on Cybernetics*, vol. 51, no. 11, pp. 5409–5422, 2021.

[31] D. Tiano, A. Bonifati, and R. Ng, “Featts: Feature-based time series clustering,” in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2784–2788.

[32] A. Alqahtani, M. Ali, X. Xie, and M. W. Jones, “Deep time-series clustering: A review,” *Electronics*, vol. 10, no. 23, p. 3001, 2021.

[33] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, “Learning deep representations for graph clustering,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 28, no. 1, 2014.

[34] N. S. Madiraju, S. M. Sadat, D. Fisher, and H. Karimabadi, “Deep temporal clustering : Fully unsupervised learning of time-domain features,” *CoRR*, vol. abs/1802.01059, 2018.

[35] J. Schneider, P. Wenig, and T. Papenbrock, “Distributed detection of sequential anomalies in univariate time series,” *The VLDB Journal*, vol. 30, no. 4, pp. 579–602, 2021.

[36] Z. Cheng, Y. Yang, S. Jiang, W. Hu, Z. Ying, Z. Chai, and C. Wang, “Time2graph+: Bridging time series and graph representation learning via multiple attentions,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.

[37] D. A. Soltysik, D. M. Thomasson, S. Rajan, and N. M. Biassou, “Improving the use of principal component analysis to reduce physiological noise and motion artifacts to increase the sensitivity of task-based fmri,” *Journal of Neuroscience Methods*, vol. 241, pp. 18–29, 2015.

[38] J. Lei and V. Q. Vu, “Sparsistency and agnostic inference in sparse pca,” *Annals of Statistics*, vol. 43, pp. 299–322, 2014.

[39] A. Strehl and J. Ghosh, “Cluster ensembles — a knowledge reuse framework for combining multiple partitions,” *J. Mach. Learn. Res.*, vol. 3, pp. 583–617, 2003.

[40] A. L. N. Fred and A. K. Jain, “Combining multiple clusterings using evidence accumulation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 835–850, 2005.

[41] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in python,” *J. Mach. Learn. Res.*, vol. 12, p. 2825–2830, 2011.

[43] T. Zhang, R. Ramakrishnan, and M. Livny, “Birch: an efficient data clustering method for very large databases,” in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996, p. 103–114.

[44] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: ordering points to identify the clustering structure,” in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, 1999, p. 49–60.

[45] R. J. G. B. Campello, D. Moulavi, and J. Sander, “Density-based clustering based on hierarchical density estimates,” in *Advances in Knowledge Discovery and Data Mining*, 2013, pp. 160–172.

[46] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *KDD’96*, 1996, p. 226–231.

[47] X. Li, J. Lin, and L. Zhao, “Linear time complexity time series clustering with symbolic pattern forest,” in *IJCAI-19*, 2019, pp. 2930–2936.

[48] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, “Reservoir computing approaches for representation and classification of multivariate time series,” *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[49] D. Steinley, “Properties of the hubert-arable adjusted rand index.” *Psychological methods*, vol. 9, no. 3, p. 386, 2004.

Paul Boniol is a Researcher at Inria, in the VALDA project-team. Previously, he worked at ENS Paris-Saclay, Université Paris Cité, EDF Research, and Ecole Polytechnique. His research interests lie between data management, machine learning, and time-series analysis. His Ph.D. focused on subsequence anomaly detection and time-series classification, and won several PhD awards, including the Paul Caseau Prize, supported by the Academy of Sciences of France. His work has been published in the top data management and data mining venues.
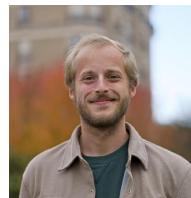
Donato Tiano is a Postdoctoral Researcher at the University of Modena and Reggio Emilia, part of the SoftLab group. He has worked at the University of British Columbia and the CNRS LIRIS lab at Lyon 1 University. His research focuses on interpretable solutions in time series analysis and text mining. Donato’s work is published in leading data management and mining venues, where he also serves on the program committee.

Angela Bonifati is a Senior member of the French IUF and a Distinguished Professor in Computer Science at Lyon 1 University (France), affiliated with the CNRS Liris research lab. She is the Head of the Database group in the same lab. Since 2020, she is also an Adjunct Professor at the University of Waterloo (Canada) in the Data Systems Group. She has co-authored more than 200 publications in data management, including five Best Paper Awards, two books and an invited ACM Sigmod Record paper. She is the recipient of the prestigious IEEE TCDE Impact Award 2023 and a co-recipient of an ACM Research Highlights Award 2023. She is the Program Chair of ICDE 2025, the General Chair of VLDB 2026 and an Associate Editor for the Proceedings of VLDB and for several other journals (VLDB Journal, IEEE TKDE, ACM TODS, etc.).

Themis Palpanas is an elected Senior Member of the French University Insitute (IUF), and Distinguished Professor of computer science at Universite Paris Cite (France). He has authored 14 patents, received 3 best paper awards and the IBM SUR award, has been Program Chair for VLDB 2025 and IEEE BigData 2023, General Chair for VLDB 2013, and has served Editor in Chief for BDR. He has been working in the fields of Data Series Management and Analytics for more than 15 years, and has developed several of the state of the art techniques.