Universiteit Leiden

Computer Science

Interactive Visualization and Mining
of Massive Time Series

Name: Alberto Baggio
Student-no: s1209000

Date: 30/01/2014

1st supervisor: Arno Knobbe
2nd supervisor: Ugo Vespier

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgements

I am highly indebted to my academic supervisors Ugo Vespier and Arno Knobbe for their feedbacks, time, effort and constant guidance throughout the work.

Last but not least, I would like to express my gratitude towards my parents, brother Riccardo, girlfriend Ilaria, aunt Mary and friends Andrea, Laura, Francesca.

Abstract

When large amount of data is collected from complex time-evolving system, interactive time-series visualization showed to be really useful to perform exploratory analysis and form an intuition of the system behavior. The literature proposes several tools, but none of these provides high interactivity, centralized data collection and open experiments sharing. VizTool, the novel time series visualization software we propose, bridges this gap allowing for interactive and exploratory data analysis. When visualizing time series, some sort of data approximation is needed since this is plotted in a viewport with a pixel width typically smaller than the number of points in the series. Many approximation algorithms have been proposed but none of these presents a satisfying trade-off between the compression ratio and the ability to preserve perceptual features in the data. Our work tackles the problem by presenting a method to select a data adaptive hybrid approximation obtained by composing diverse techniques.

Finding unknown patterns from time series data, known as motif discovery, resulted to be one of the most relevant topic in the field related to our work. Due to its expensive computational cost, quadratic in the number of points, several algorithms which compute the motifs over data approximation have been proposed. However, such algorithms exclude exact results in any case. Therefore, as additional contribution we propose an anytime motif discovery algorithm which returns a result anytime its termination is invoked, and which progressively converges to the exact result.

Contents

List of Figures

Chapter 1

Introduction

Over the last years, there has been a rapid increase in the amount of data being collected. We witnessed a real data explosion which drastically changed the way we work and do research in a big variety of disciplines, such as biology, medicine, telecommunications, sensor network. Within this context, time series collection and analysis have become one of the most relevant fields of study. When dealing with large amounts of data measured from complex time evolving systems, interactive time series showed to be an effective way to perform exploratory analysis and form an intuition of the system's behavior. The studies and developments within the field have been numerous, but we experienced a lack of easy-to-access tools, supplying centralized data collections and open sharing of experiments.
In order to bridge this gap, our work proposes VizTool, a novel time series visualization tool that offers interactive and exploratory data analysis. It is a platform that fosters community efforts by providing a place to share and analyze datasets and experiments collaboratively over common testbeds. We are convinced that transparency and comparability of experiments are crucial elements of the research improvement process.
The tool adopts a subsampling hierarchy that makes the platform scalable to datasets of any size: this is a solution which exhibits an efficient resource usage by adapting the displayed data resolution to the client capabilities. The final outcome is a portable, ubiquitous and safe tool which implements a web client-server architecture, where the data is stored in a unique, safe and centralized server. Several state-of-the-art techniques able to boost the overall interactivity (such as data prefetching and data buffering) have been applied. Moreover, several components, such as data annotation and data import/export, have been integrated for data management and data sharing.

The main challenge when visualizing large time series is to maintain interactivity while allowing the user to quickly zoom in and retrieve detailed portions of the data. Moreover, since the data is plotted in a viewport with a pixel width which is typically much smaller than the number of points in the time series, some sort of approximation of the original data needs to be performed. The best approximation for this task is the one with the best trade-off between compression ratio and the ability to preserve important perceptual features in the data. The literature contains many examples of approximation algorithms for time series, from frequency-domain methods, such as Discrete Fourier Transform and Discrete Wavelet Transforms, to time-domain methods such as Piecewise Aggregate Approximation, Adaptive Constant Approximation and Gaussian Aggregation. These are however focused on minimizing the Euclidean distance between the original data and the reduced one. There are few algorithms which actually aim at preserving the perceptual features of the original data. Ramer-Douglas-Peucker, Perceptually Important Points and Important Extrema are the most widely cited. However, while at low compression ratios, they model the data pretty well, their approximation error (in terms of Euclidean distance) becomes considerable at high compression ratios. Considering these limitations, we claim that there is not a single existing technique which behaves well under the interactive visualization requirements depicted above. Therefore, we propose a method to select a data-adaptive hybrid approximation obtained by composing various techniques.

Time series motif discovery, i.e. the task of finding unknown patterns from time series data, is one of the most relevant topics in the research field related to our work. Indeed, it is one of the fundamental time series mining components provided by our tool. The obvious solution to such a problem has a worst-case quadratic cost in the number of points in the series, which compromises the overall interactivity. From our experience, in the majority of cases, the domain expert looks just for a brief overview of the motifs, while the exact report is required only in sporadic circumstances. The literature proposes several algorithms that compute the motifs over approximation of the data, but these exclude the exact result in any case. Therefore, as additional contribution, we propose an anytime motif discovery algorithm which progressively improves the obtained results, a solution which adapts the execution time to the user's needs without compromising the software interactivity.
The data collected from the InfraWatch Project, where a big number of sensors were embedded and attached to the Hollandse Brug by Strukton, has been the main industrial dataset we used in our experimentations.

The first chapter of this thesis introduces the work which has been conducted so far: time series data mining tasks and visualization tools will be discussed. Moreover, the principal techniques for time series motif discovery and time series segmentation will be discussed. In Chapter 2, we will outline the software architecture and the main components that it provides. The conceptual choices we made will be proposed as well. The observations and the many steps which brought us to the formulation of the data adaptive hybrid approximation algorithm will be outlined in Chapter

3. Chapter 4 outlines the motivations and the concepts behind the resolution-aware motif discovery algorithm we propose. Finally, in Chapter 5 our conclusions are presented and some proposals for future work are made.

Part of the research in this thesis has been presented at Benelearn 2013, the annual machine learning conference of Belgium and the Netherlands, in Nijmegen. This master thesis is written as a partial fulfillment of the requirements of the degree Master of Science in the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University, and is supervised by Arno Knobbe and Ugo Vespier.

Chapter 2

Related work

Three main research fields are related to the content of this thesis. The main outcome of the work is a novel web-based tool to explore and mine high dimensional time-series. Therefore, our research focussed mainly on interactive time series data mining. Here, we analyze the methods proposed so far to interactively involve the domain experts, and so, make them more efficient and effective while performing data mining tasks on massive time series data. Hence, topics such as ubiquitous and anytime computing will be looked at closely.
The aforementioned research fields involve all the studies which concern data mining tasks for time series. Indeed, we will focus mainly on two relevant problems: data sub-sampling and time series motif discovery. We will analyze pros and cons of several segmentation techniques mentioned in the literature, and we will propose an anytime approach to perform time series motif discovery.
Finally, the analysis of all the open platforms to share and analyze time series and experiments collaboratively is intrinsically related to the content of this thesis.

2.1 Background

The following section introduces some basic notions and terminology we will use throughout the entire thesis. Moreover, we define some assumptions we make and which will be valid henceforth.
A time series is a collection of values collected over time.

Definition. *A time series T is an ordered sequence of n real-valued variables $T = \{t_1, t_2, \ldots, t_n\}$, $t_i \in R$*

In many data mining tasks, as motif discovery, we will often work with subsequences of a given time series.

Definition. *Given a time series $T = \{t_1, t_2, \ldots, t_n\}$ of length n, a subsequence S of T is a series of length $m \leq n$ consisting of contiguous time instant from T, $S = \{t_k, t_{k+1}, \ldots, t_{k+m-1}\}$, with $1 \leq k \leq n - m + 1$.*

Often, time series are too big to be analyzed and data approximation is necessary. The aim of data approximation is to reduce the size of data while retaining the signal fundamental shape and characteristics.

Definition. *Given a time series $T = \{t_1, t_2, \ldots, t_n\}$ of length n, a representation of T is a model $\hat{T}$ of reduced dimensionality $\hat{d} << n$ such that $\hat{T}$ closely approximates T.*

Most of the data mining tasks such as clustering, classification, prediction and motif discovery, involve the concept of similarity. The notion of similarity should be based on perceptual criteria.

Definition. *The similarity measure $D(T, U)$ between time series T and U, is a function taking two time series as inputs and returning the distance between them.*

Every evaluation which involves the concept of similarity in our work, will use the Euclidian Distance. Although we know it is not the best measure to assess the intuitive notion of shape, it has shown to be effective when working with really large datasets.

Definition. *Given two time-series $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_n\}$, the Euclidian Distance between them is defined as:*

$$L_2(P, Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{2.1}$$

In many cases, we have to compare a time series with its approximated representation. In such a case, being the points in the second fewer than the points in the first, we need to up-sample the approximation. We will always perform the up-sampling process by using Linear Interpolation.

Definition. *Given p_i a point in the original series $P = \{p_1, p_2, \ldots, p_n\}$ and p_{t_i} its timestamp, q_i the commensurate point in the reduced signal $Q = \{q_1, q_2, \ldots, q_m\}$ is computed as:*

$$q_i = \begin{cases} q_l + \dfrac{q_r - q_l}{q_{t_r} - q_{t_l}} \cdot (p_{t_i} - q_{t_l}) & \textit{if } q_{t_l} \neq q_{t_r} \\ q_l \equiv q_r & \textit{if } q_{t_l} \equiv q_{t_r} \end{cases} \tag{2.2}$$

$$l = max(q_{t_1}, \ldots, q_{t_m}) \leq p_{t_i},\ r = min(q_{t_1}, \ldots, q_{t_m}) \geq p_{t_i}$$

2.2 Data Mining tasks for time series

This section gives a brief overview of the main data mining tasks related to time series. We will focus on data segmentation and time series motif discovery, as both

are deeply analyzed in our work.
According to [7], the purpose of time series data mining is to try to extract all meaningful knowledge from the data. Therefore, the most prominent problems arise from the high dimensionality of data and the difficulty of defining a form of similarity-measure, based on human perception. We totally agree with this formalization of the problem, hence as Esling does, we organize the time series data mining in three major areas: representation techniques, similarity measurement and indexing method.
The first data mining task for time series we mention is **clustering**, i.e. the process of grouping different time series in such a way that time series inside the same group (cluster) are most similar to each other than the ones into a different group. That is to say that clustering aims to compose groups of time series which are similar to each other and dissimilar to time series in other groups. Formally, clustering works in order to minimize intra-cluster variance and maximize inter-cluster variance. The problem is extended even to non-overlapping subsequences of a given time series. The main approaches taken to solve clustering problems are: *Self Organized Maps* (SOM), *Hidden Markov Models* (HMM) and *Support Vector Machines* (SVM).
Given a set of time series, the **classification** task aims to assign a label to each one of these, using the knowledge gained from a set of time series for which the labels are already known (training set). Formally, given an unlabeled time series T, classification assigns it to one class c_i from a set $C = \{c_1, c_2, \ldots, c_m\}$ of predefined class. In classification, differently from clustering, the set of classes is already known and the model is trained on an example dataset.
The last two data mining tasks we mention are **prediction** and **anomaly detection**. The first aims to build models out of the current data in order to forecast the next incoming data. The second seeks abnormal subsequences in a series. Usually, the standard approach to detecting anomalies in the data is to build a model which describes the series, and successively, mark as anomalies sequences that exhibit different behavior from the one modeled.

2.2.1 Similarity measures

Almost each data mining task involves the concept of distance among signals. Hence, a definition of similarity between time series which takes into account the perceptual criteria is required. Several time series similarity measures can be found in the literature, and they can be classified in four categories. *Shape based* distances compare the overall shape of the series. *Edit based* metrics compute the total number of steps which need to be taken to transform one series into another one. *Feature based* distances extract features from the series and compare them with any kind of distance function. Finally, *structure-based* similarity measures extract higher level structure from the series and compare them each other.

2.2.2 Segmentation

Segmentation seeks for accurate approximations of time series by reducing its dimensionality, while retaining the shape and the characteristics of the data [7]. From a mathematical point of view, the aim of such a task is to minimize the error between the approximated signal and its original version.
The literature presents a plethora of segmentations techniques. According to [12], we can divide them in two categories: *non data-adaptive*, which keep constant the number of points used to reduce every region in the series; *data-adaptive*, which adapt the points used to reduce a given region, according to some features it exhibits.

2.2.2.1 Piecewise Aggregate Approximation

Surely, the *Piecewise Aggregate Approximation* (PAA) is one of the non-data-adaptive most-used dimensionality reduction technique; it is also known as span-based averaging and simply averaging. It approximates the signal by using the mean values of equally sized frames obtained by splitting the series with a constant-size sliding window. Given a signal $P = \{p_1, p_2, p_n\}$ and a sliding window of size m, it approximates the signal as $Q = \{\hat{s_1}, \hat{s_2}, \hat{s_m}\}$, where $s_1 = \{p_1, \ldots, p_{\frac{n}{m}}\}, \ldots, s_m = \{p_{\frac{(m-1)n}{m}+1}, \ldots, p_n\}$. The compression ratio is given by n/m. As we previously analyzed in [1], while it keeps relatively low the Euclidian Distance between approximation and original series even at high compression ratio, the perceptual features of the signal are largely lost. Therefore, the signal results to be progressively smoothed.
A slightly modified version of the PAA can be found, instead of computing the arithmetic average of the aggregation intervals, it computes the weighted average. The weight of each point is its normal distribution g(x) in the interval.

2.2.2.2 Piecewise Linear Approximation

The Piecewise Linear Approximation (PLA) is probably the main undertaken approach over the past years. As the PAA, it approximates the series with a set of straight lines. But, here each one of the segments is obtained by linking the first and the last point in the interval. PLA is an adaptive-method, i.e. the segments are not equally sized. Starting from two points, those are recursively joined until a given euclidean error threshold is reached. The top-down version of the algorithm works the other way around: starting from one segment linking the first and the last point in the series, it recursively splits it in different segments, until each one of these presents an error below the threshold. A sliding window version is available as well. It grows a window left to right until the threshold is exceed.

2.2.2.3 Ramer-Douglas-Peucker

The Ramer-Douglas-Peucker algorithm, also known as end-point fit algorithm or split-and-merge-algorithm, is one of the earliest data-adaptive algorithm. Shortly, it uses more points to approximate regions of the series where the standard deviation is higher. Given a time-series $P = \{p_1, p_2, \ldots, p_n\}$, the method starts marking the first and the last point in the series. Then, it finds the furthest point from the segment merging the two marked points, we call this point p_f. If p_f is closer than epsilon to the segment, all the not marked points are discarded. Otherwise, the algorithm recursively calls itself twice with the subsets $s_l = \{p_{start}, \ldots, p_f\}$ and $s_r = \{p_f, \ldots, p_{end}\}$, respectively as arguments. When the recursion is completed, the approximated series is the one composed of the only marked points. The point-segment distance is computed by using the perpendicular distance. As shown in [1], at reasonable compression ratios RDP has a really high capability of preserving the perceptual features of the signal, better than the algorithms previously mentioned. But, as the compression grows the approximated series result to be an unreal representation of the original one.

2.2.2.4 Maxima Extrema

Fink et al in [8], propose Maxima Extrema a fast lossy compression technique, based on assignment of importance levels to the time series minima and maxima. The authors define four type of extrema: strict, left, right and flat extrema. They suggest how every time series can be compressed by selecting its strict, left and right extrema. Moreover, in order to achieve higher compression ratios, Fink et. al formalize the concept of important extremum. Given a distance function d and a positive value R, the point a_i of the time series $P = \{a_1, a_2, \ldots, a_n\}$ is an important minimum if it exist indexes i_l and i_r, where $i_l < i < i_r$, such that, a_i is a minimum among $a_{il}, \ldots, a_{ir}$, $dist(a_i, a_{il}) \geq R$ and $dist(a_i, a_{ir}) \geq R$. The definition of important maximum is the other way around. Therefore, we can derive the formalization of importance of a given point as the maximal value of R for which the point is still an important extrema. By using this formalization, we can easily reduce a given series by picking the required number of points after having ordered them according their importance value. Maxima Extrema presents characteristics similar to the Ramer-Douglas-Peucker algorithm.

2.2.2.5 Discrete Fourier Transform

The Discrete Fourier Transform is another widely-used technique. It reduces a time series of length n to a set of n sine/cosine waves, which composed together reconstruct the original data. The algorithms converts the series from the time domain to the frequency domain. Each one of the obtained waves has an associated frequency, which is composed of an integer multiple of the fundamental frequency and a complex number named the Fourier coefficient. The FFT costs $O(n^2)$ operation, but by using the Fast Fourier Algorithm, we can compute it in $O(nlog(n))$ time. According to our experiments [1], the FFT approximation unveils the best trade-off between Euclidian Distance and perceptual features preserving. In [1], given a time series of length n, in order to reduce it to $m < n/2$ points, we computed the FFT coefficients and calculated the inverse of the coefficients trimmed to m values. By doing this, we ensured a reduced signal of length m.

2.2.2.6 SAX

In [15], Lin et al present the Symbolic Aggregate Approximation (SAX), a symbolic representation composed of equiprobable symbols, which allows for dimensionality reduction and indexing. As PAA, it splits the series in equally sized regions and compute the mean of the value in each one of these. Then, according to the mean amplitude, it assigns to each one a symbol from the alphabet. The iSAX representation extends SAX by allowing different resolutions for the same word.

2.2.3 Time series motif discovery

Time series motif discovery is the task of finding unknown patterns from time series data, that is to say finding every subsequences that appear recurrently in a longer time series [7]. Motif Discovery is a relevant problem to a big variety of disciplines like biology, medicine, telecommunications, sensor network. This relevance comes from the utility motifs have in the generation of insights about the system behavior from a domain expert point a view.
Different formalizations of the problem can be found in the literature, and among others two are the most common-used. The first formalization, aims to find the top-K motifs ranked according to the number of instances, i.e. the number of times a given subsequence appears in the time-series. This formalization aligns the motif discovery problem to the clustering problem. In many of the proposed solutions, two or more time series constitute a motif if their distances is less than a user given parameter R.
The second formalization aims to find the top-K subsequences matches in the time series, ranked according to their similarity. It does not consider the patterns number of occurrences. The latter is the one used in our work.
Given the relevance of the problem many time series motif discovery algorithms have been proposed. We can classify them into two categories, the exact match discovery and the approximated ones. The firsts rely their research on the original data and ensure exact results, while the seconds use series approximations and therefore give approximated result. The high research on approximated algorithms is given to the high computational cost of such a task. Given a time-series P of length n and a motif of length m, the obvious algorithm to perform exact match discovery costs $O(n^2m)$. A task which presents quadratic cost in the number of items, clearly, cannot be used with massive series.
Before to describe the most relevant motif discovery techniques proposed, we have to mention that every algorithm assumes normalization of the time series or subsequences, in order to remove offset and scaling effects. Moreover, each one of them

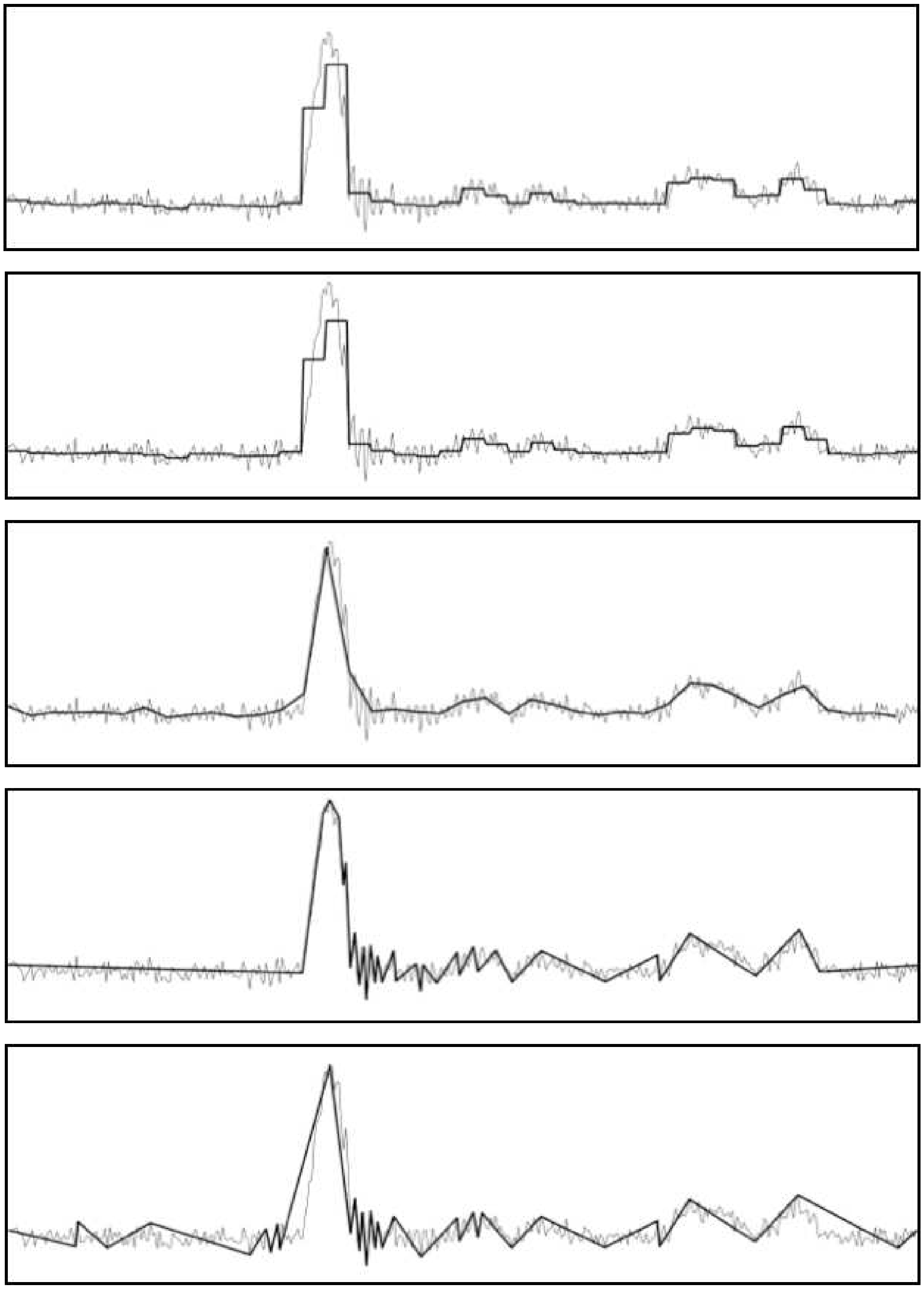
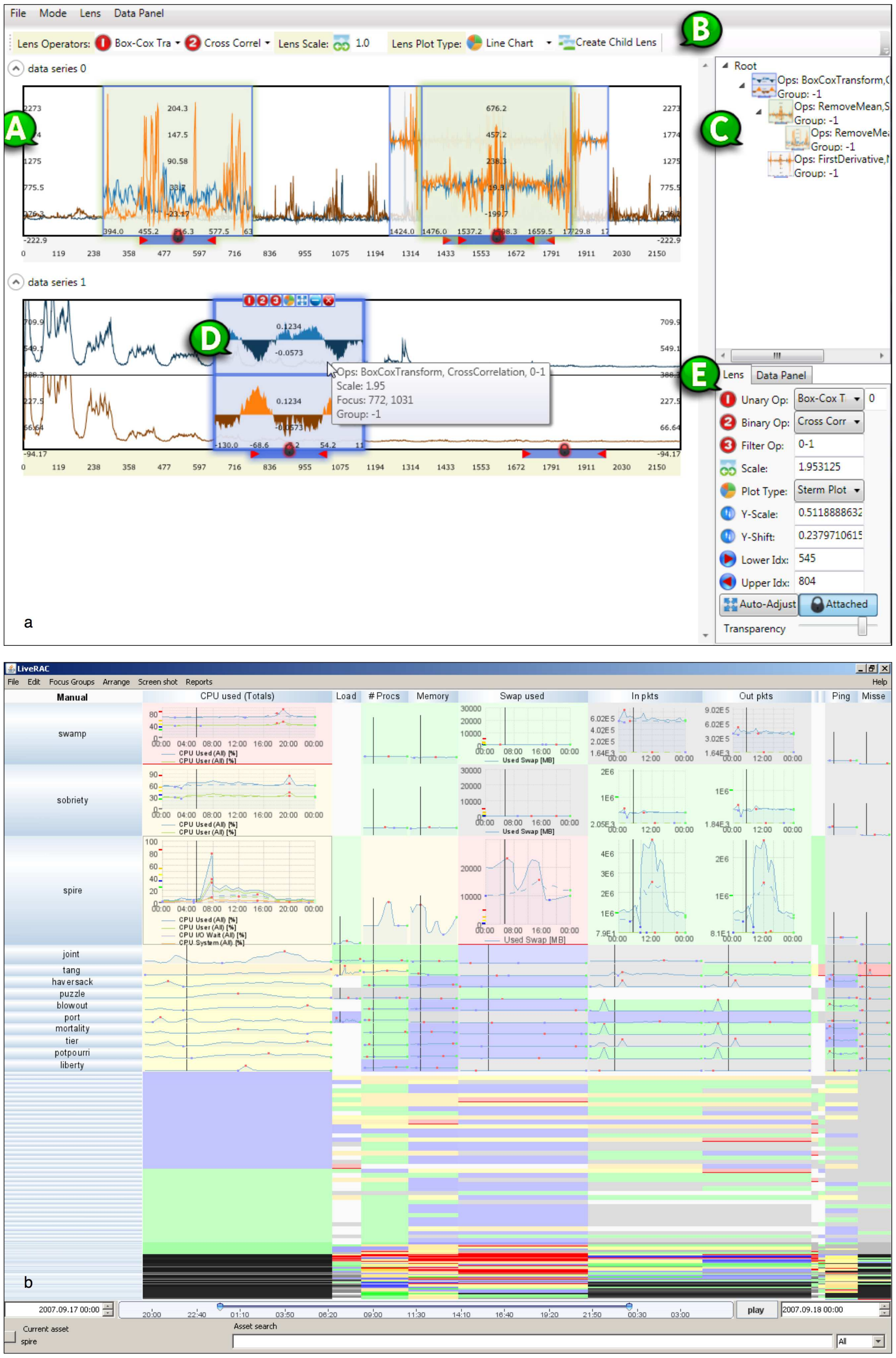


Figure 2.1: **Segmentation techniques example.** Piecewise Aggregate Approximation, Gaussian Approximation, Fast Fourier Transform, Ramer-Douglas-Peucker, Maxima Extrema.



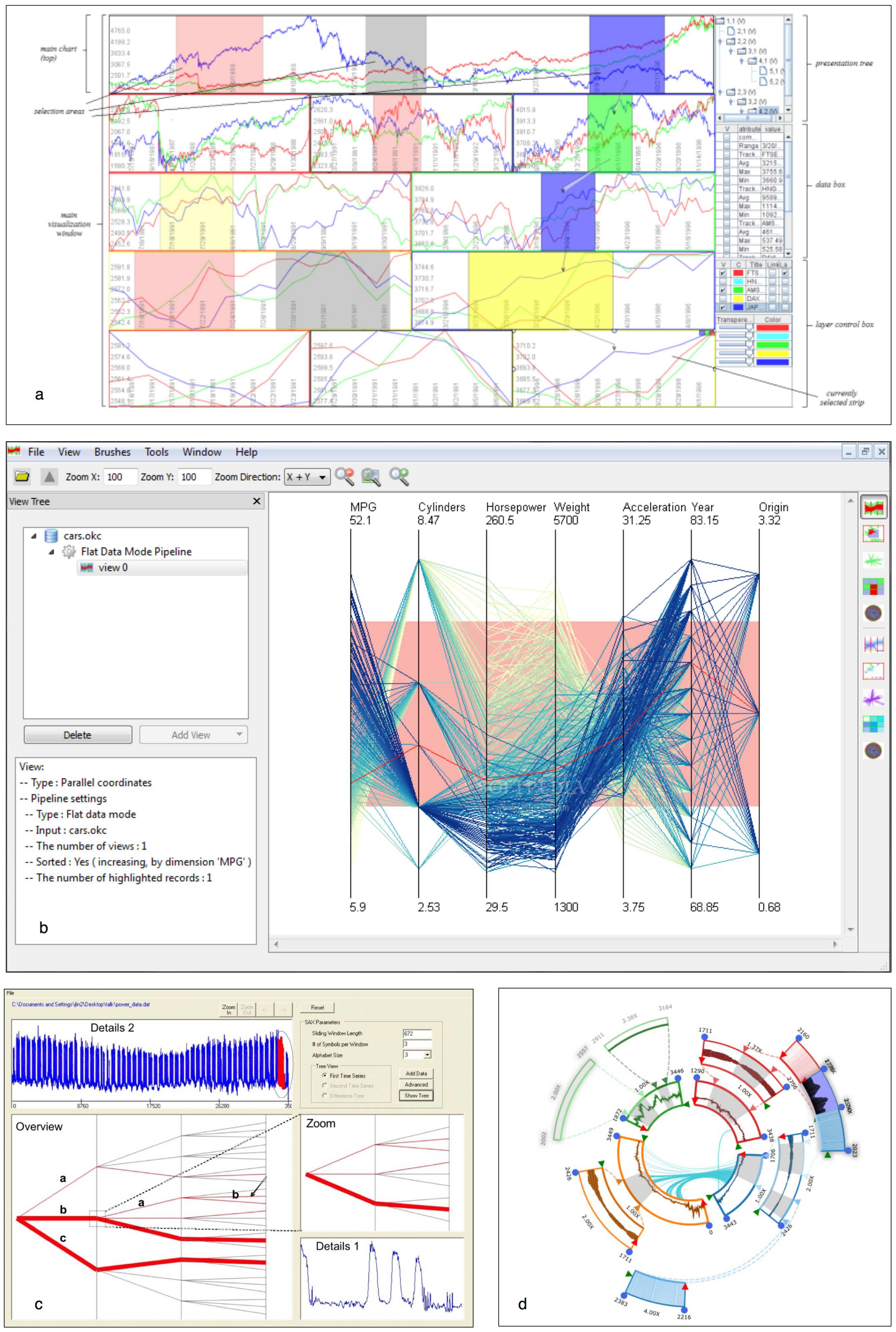
Lens Operators: Box-Cox Tra, Cross Correl; Lens Scale: 1.0; Lens Plot Type: Line Chart

Ops: BoxCoxTransform, CrossCorrelation, 0-1
Scale: 1.95
Focus: 772, 1031
Group: -1

Unary Op:	Box-Cox T
Binary Op:	Cross Corr
Filter Op:	0-1
Scale:	1.953125
Plot Type:	Stem Plot
Y-Scale:	0.5118888632
Y-Shift:	0.2379710615
Lower Idx:	545
Upper Idx:	804

2007.09.17 00:00 — 2007.09.18 00:00

Figure 2.2: **Time series visualization tools. a.** Chronolenses. **b.** LiveRAC.



View:
-- Type : Parallel coordinates
-- Pipeline settings
-- Type : Flat data mode
-- Input : cars.okc
-- The number of views : 1
-- Sorted : Yes (increasing, by dimension 'MPG')
-- The number of highlighted records : 1

Figure 2.3: **Time series visualization tools. a.** Stackzooming. **b.** XmdvTool. **c.** VizTree. **d.** Kronominer.

excludes from the results the trivial matches, i.e. all the motifs composed of overlapping windows.

2.2.3.1 Muen-Keogh exact motif discovery

In [19], Mueen et al. propose MK, probably the most cited exact algorithm in the literature. The main concept behind it is that, in most of the cases, it is possible to discard a-priori matches to be analyzed. Basically, it makes sense to test just the promising matches and skip the other. The authors exploit the triangular inequality in order to define a lower bound and prune the computation. The idea is that, given a subsequence ref as reference, for any pair of windows (a, b) in the signal, we know that $D(ref, a) - D(ref, b) \leq D(a, b)$. So, we can use the distances between the reference sequence and the windows in the signal to define lower bounds for every possible windows match. Now, by knowing the best-so-far match, we can prune the research by a-priori discarding matches with lower bounds greater than what we have so far. In the worst case, i.e. when at each step every lower bound is less than the best so far, the computational cost keeps to be quadratic in the number of elements. In [19], the authors adopt a technique widely-used, the early abandoning: when computing the distance between two time-series or subsequences, it is possible to abandon the computation as soon as the error exceed the best-so-far distance.

2.2.3.2 Mueen, all lengths motifs

In [18], Mueen proposes an algorithm which allows the discovery of exact motif for every possible length of the window. An obvious implementation of such an algorithm would have a computational cost equal to $O(n^3m)$. But, with the use of a lower bound, the author is able to prun the computation by freeing it from discovering the same motif at different lengths. We will not go to much in detail with the description of the constructed bound, since it would need a deep mathematical discussion. However, we mention that the basic idea behind it is that we can build bound for error of longer sequences by using the error between shorter distances. Basically, at each step with windows of length m, the algorithm is able to prun the computation by using the knowledge gained at the step before, with windows of length $m - 1$.

2.2.3.3 Castro mrMotif

In [3], Castro et al exploit iSAX symbolic representation to obtain motifs at different resolutions. The iSAX segmentation converts each window to a symbolic string as described in the previous section. Two windows constitute the same motif if their symbolic representations fit each other. In order to extract motifs at different resolution, each window is converted to multiple iSAX representation. The conversion is performed by adopting different alphabet lengths, which are all power of 2 in the range $[2, 64]$. Differently from MK where the authors look for the best match, Castro et. al seek for the best top-k matches, ordered according the number of instances in the signal. The same authors in [4], make an extension to this work which uses Markov Chain models in order to estimate the expected frequency of a motif. This knowledge is obtained regarding a reference model that reflects the background distribution of the motifs. By using it, the domain experts have an instrument to evaluate the degree of relevance of the found motifs.

2.2.3.4 Top-k motif discovery

Usually, we force the top-k motif discovery algorithms to produce a set of results such that they are not overlapping each other. From this configuration, arises a big memory issue. Every time a new motif enters the top-k list, up to four other motifs could be removed. Therefore, we need to keep into memory the scores of the previously checked matches. They are approximately $O(n^2)$ pairs of windows, a huge number when working with long series. In [14], Lam et al tackle the problem. The work, by doing some assumptions, proposes an online algorithm which consumes $\Omega(w + 2klog(w))$ float number on average (w is the motif length). The proposed online-algorithm implements early abandoning as well.

Finally, we just mention that, a plethora of techniques which compute approximated motifs by using random projection algorithm, as [24], have been proposed. These are usually outperformed from aforementioned algorithms, so, we skip the analysis of them.

2.3 Interactive time series data mining

In the last few years, several time series visualization and mining tools have been proposed, but none of them combines interactive and intuitive approaches with the problem of handling large datasets. We can identify two typologies of products. The first is composed of those tools which supply an interactive time series visualization and in few cases allow on-the-fly data transformation. The second is about those platform which allow algorithms and filters composition (workflow engine), without providing any data visualization interface. This section describes the most relevant tools proposed, and in the majority of the cases developed (there are few white papers).

2.3.1 A visual analytics for peak preserving prediction of seasonal time series

In [9], Hao et al. present a time series approximation and prediction algorithm, ables to preserve the most relevant peaks and motifs in the signal. The algorithm is a modified version of the Ramer-Douglas-Peucker approximation technique, where a simplified distance metric is used in order to narrow down the computational cost. The proposed algorithms assigns a different weight to each one of the points in the signal, in relation to its age. This is due to the authors assumptions that more recent points should always affect more the prediction. In [9], Hao et al. stress the effectiveness of the algorithm, but as we observed in [1], time series approximations techniques which reduce the signal by extracting most relevant peaks from it, achieve satisfying results only at low compression ratios. At relatively high reduction, the euclidian distance between original data and approximation explode. So, it cannot be considered a right approach when dealing with large data.
The aforementioned technique is used into a visualization tool which uses a high-resolution cell-based visualization, in order to concisely depict the time series. The predictions are outlined into a detached window by using line charts; on the left side are visible historical data, on the right side predictions embedded into a certainty band. The band draws indicators which according to the their color, point out the prediction accuracy. The data zoom-out is constrained within fixed limit by the platform; this is because if exceed a reasonable threshold, the on-the-fly data aggregation would become too expensive, affecting the overall interactivity. It is straightforward, but necessary, to observe that the zoom-out limitation end up with considerably narrow down the user exploratory possibilities.

2.3.2 LiveRAC

In [17], McLachlan et al. present LiveRac a visualization tool implemented in Java which supports big time series analysis and exploration. As many other tools, the system uses matrix layout and semantic zooming, i.e. visualization techniques which displays charts by using an order-able matrix, and where the displayed objects adapt their graphic representation according to the screen available space. The authors suggest how the user interface design and architecture has been driven by a set of principles. The first of them is the observation that simultaneous visualization of different levels-of-detail provides deep and useful context information. Secondly, the user, when possible, should be fed with graphical representations he is used to. Therefore, the tools depicts the data by using standard line and bar chart. The spatial positioning is the strongest perceptive impression, so the user is supported by a side-by-side comparison rather than remember the previously seen information. Moreover, the overplotting results to be misleading. The simultaneous visualization of several views becomes more effective when these are connected each other, i.e. the zooming of one time-series is propagated to all the other. Finally, McLachLan

et al. stress the importance of presenting a data overview before allowing it to be zoomed and filtered.
What we are mainly concern about, is the fact that although the tool copes with massive dataset, it does not make any data pre-elaboration. That is to say that the data aggregation (they are implemented 4 different typologies: min, max, mean and cardinality) is performed on the fly, and so it is time consuming. As stated by authors, when the user changes the time windows or first expands a cell from a block chart, the server query to obtain more data may takes seconds. The tool faces this and keeps interactivity by disjoining server querying and graphical rendering into two different threads. But, we are still concerning about the effectiveness of these with really massive data. The tools does not allow data composition or the use of on-the-fly filters/algorithms. Figure 2.2.a depicts a screenshot of the tool.

2.3.3 ChronoLenses

In [26], Zhao et al. present one of the most interesting tools we analyzed: ChronoLenses, a technique for interactive time series visualization and exploratory analysis, which enables time series derivation and composition and allows interactive parameters and functions experimentation. The tool introduces the concept of lenses, used to select and zoom portions of data which can be filtered and transformed. All this takes place on-the-fly and gives immediate feedback to the user, which is enabled to progressively and iteratively build pipelines. The tool supplies mainly two functionalities: the time-series transformation by using operators; and the time-series computation as composition of other time series. The tool implements all the functionalities the authors list as necessary: the dynamic selection of interesting region as input; the dynamic transformation of used parameters; immediate visual feedback of the build pipeline; the possibility of reusing intermediate results and going back to earlier steps in the pipeline. We did not test the behavior of the tool when dealing with massive data sets, so we do not know the effective scalability of it. In Figure 2.2.b, is visible a screenshot of the tool.

2.3.4 Kronominer

In [25], Zhao et al. present KronoMiner a multi-aim tool for the visual exploration of multivariate time series, mainly focused on non-intrinsically-periodic linear sequence of points. The tool has been developed earlier than ChronoLenses [26], by the same authors. It presents many common points with the afore-mentioned tool, but it differs in the visualization-technique used. KronoMiner displays the data by using a radial display, where circular and linear layout are combined together. The tool claims to be an instrument ables to deal with multivariate data sets, where different views facilitate the user exploration. It makes this possible, by proposing different perspectives of the data, such as: large scale overview; temporal multi-foci hierarchies, side-by-side comparison, etc. Besides data scrolling, zooming and panning, the user exploration is supported by data static analysis. The work introduces the concept of lenses, later on introduced even in ChronoLenses. Here, the authors introduce MagicAnalytics Lens, an interactive visualization technique which allows for on-the-fly computation and visualization of functions involving two time series. As many other tools, it does not clarify the effective scalability when dealing with massive datasets. We really concern about the effective responsiveness of it, given the on-the-fly computation of many operations as zoom, pan or the use of the MagicAnalytics Lens. Moreover, we strongly believe that the standard line-graph visualization is still more intuitive and readable. Conversely, the radial visualization helps patterns identification, but loses the immediate temporal feeling given by the first. Figure 2.3.d depicts the graphical-layout used in the time series visualization.

2.3.5 XmdvTool, prefetching for visual data exploration

In [6], Doshi et al. tackle the scalability problems encountered when the tools have to deal with massive datasets. They show how interactivity involves the necessity of real time responses. To solve the aforementioned problem, the paper proposes semantic caching and several prefetching techniques, which are embedded into a freely publicly available tool: XmdvTool. The semantic caching is defined as the dynamic caching of objects which logically belong to a same entity, and so it can be described by a unique semantic descriptor.
The necessity for semantic caching is justified by the fact that client queries are not causality driven, but follow a clear pattern. Moreover, given the exploratory structure of the tool, these tend to be contiguous each other rather than ad-hoc. Shortly, the authors motivate semantic caching and prefetching with several reasons: database queries are usually contiguous each other; usually, the exploration tends to be localized into a region for long periods; the tool structure facilitate the users behavior prediction; user performed operations occurs within substantial temporal intervals, so there is temporal-room to load data.
The several prefetching strategies, one of the key focus for the work, show an evolutive behavior which progressively, achieves better performances. The system achieves this mainly by analyzing the user most relevant scroll direction and by guessing the region of interest in the data (statistical analysis). From these assumptions, 5 different prefetching strategies are formalized: random, the next direction and so the data to be loaded is randomly choose; same direction, the next direction is equal to the previous one, this according to the observation that the user usually scroll in the same direction for a while; focus strategy, the data belonging to the most interesting region is loaded, where the most interesting regions are established by analyzing the chronological browsing data collected from other users; vector strategy, the next direction is computed as a three-dimensional vector, where the dimensions respectively refer to movement start, movement end and level-of-detail. Two different techniques are used to predict the next user direction in the last strategy. The first technique average the historical user movement, the second performs

a weighted average, giving progressively higher weights to the recent movements. The authors mention that the benefits gained by using a prefetching strategy together with semantic caching, are really higher than the ones obtained by just using semantic caching. Figure 2.3.b depicts the tool.

2.3.6 Cypress, managing time series streams with multi-scale compressed trickless

In [21], Reves et al. present Cypress a framework to archive and query massive time series streams. [21] does not present any visualization tool, but we mention it because it proposes a technique which results to be interesting to the aim of maintaining interactive and responsive any hypothetical tool. Cypress tackles the problem of efficiently storing the data and quickly perform statistical queries and data mining tasks on it. In particular, the authors suggest that many statistical queries can be solved over compressed data, and so propose three different data compression methodologies.
The work proposes a multi-scale technique to decompose the data and obtain sparse representations of these in several domains (time domain, frequency domain). Each incoming stream is separated into three sub-streams, named trickles, focalized on different time scales.
The first sub-stream (LoF trickles) is composed of the significant frequencies in the signal. It is obtained by applying a low-pass filter to the original signal with cut frequency equal to $f_s/2M$, where f_s is the sampling frequency and M a parameter which needs to be tuned according to the data. A further step is taken, the LoF trickles is subsampled to M values by using another low-pass filter. The second sub-stream (Spike trickles) extracts the peaks from the signal obtained by subtracting the LoF trickles to the original signal, where the noise has been removed. Finally, the third sub-stream (HiF trickles) is the signal residual, obtained by performing random projection of the signal obtained by subtracting LoF and Spike sub-streams to the original series. The random projection is the compressive random projection by Johnson-Lindenstrauus.
The three sub-streams, focused on different scales (base frequencies, peaks and residual), decompose the original data in a sparse and compressed representation. The authors show the solutions of many statistical queries over the compressed data, so optimizing time and space consumption.

2.3.7 Multi-resolution techniques for visual exploration of large time series data

In [10], Hao et al use not linear time distortion techniques in order to generate aggregate visual layout, named multi-resolution grid layouts. So, the not-linear scaling of the temporal axis is the main innovation introduced by their work. The authors formalize the concept of DOI, Degree Of Interest, and use it in the procedure

of drawing multi-resolution charts, i.e. higher is the interest of a data region, higher is the resolution used to display it. Precisely, the DOI of any element x in the data, is defined as its a-priori degree of interest (API[x]) minus the distance it has from the current point of focus. [7] outlines two different formalization of DOI: time-dependent, which it is not affected by the data features but just by time; and data-dependent, the other way around. In order to compute the data-dependent Degree Of Interest, the data is split between different bins and, by using a specific function (the standard deviation could be one), a DOI is assigned to each one of these. The final graphical outcome is a matrix layout which uses chromatic scales to outline the amplitude of data inside each cell. Moreover, the bigger is the Degree Of Interest of a given region, the bigger it will be the associated cell width and will be the data resolution.
Despite the space efficiency, we do not consider this an effective solution, because with a not-linear scaling of the temporal axis, the human capability of abstraction, gained by data visualization, results to be partially lost. That is to say that the charts become much less readable and intuitive.

2.3.8 Visual interactive preprocessing of data

In [2], Bernard et al propose a framework which allows explorative and visual analysis of preprocessing pipelines, integrating both data diagnosis and data transformation functionalities. As stressed by the authors, before the explorative analysis phase, the data always needs some preprocessing. But, despite the importance of this, the majority of tools handle the data only after a preprocessing phase carried out in a black-box approach by the domain experts. So, the works aims to tackle the problem by supplying a framework, where data is interactively preprocessed by arbitrarily composing the provided components, and where the results of each step are graphically displayed. At each step, the user is driven and facilitated by the tool.
The framework supplies to the user five components. The first is a preprocessing toolbox which provides instruments to clean, reduce, normalize, segment, measure similarity and build descriptors of the data. Analytical operations like time series composition are not supplied, because the authors consider this aside from data preprocessing. The workflow-view displays and allows navigation of the composed pipeline. The detail-view displays the intermediate results of the preprocessing. The raw-data-selection-view displays the original data. Finally, the statistics-view presents a set of statics over the data and their change in relation to the taken steps.

2.3.9 Stackzooming

In [11], Javed and Elmqvist propose Stackzooming a technique which allows for multi-foci interaction, i.e. the simultaneous high-resolution visualization of several and different regions of the same time series, while the context and temporal

awareness are preserved. The tool works as follow: the user interactively builds a hierarchical stack of time series strips. Strips at different stack levels represent different zooming levels (increase the top to the bottom) and same level strips (siblings) represent branch of the visual exploration. The work tackles the same problem presented in LiveRac, the multi-foci interaction, but instead of using a unique view combined with data visual distortion, it exploits the mono-dimensional nature of the data in order to show hierarchical strips. The presented technique is implemented and tested in Trackxplorer tool.
We consider the usefulness of such a technique constrained, since according to our experience, it is not a key issue the simultaneous visualization of several views of the same time series. Conversely, we consider much more useful the simultaneous comparison of different time series, feature implemented by stackzooming with the use of the un-readable and overwhelming overplotting. In Figure 2.3.a, is visible an screenshot of the tool.

2.3.10 VizTree

In [16], Lee et al present VizTree a time series pattern discovery and visualization system, based on augmenting suffix trees. The data are approximated through a symbolic representation and codified into a modified suffix tree, where patterns frequencies and other properties are mapped with the use of different colors and other visual features. The used symbolic representation is SAX, Symbolic Aggregate Approximation. It transforms the time series into a sequence of equiprobable symbols. This is achieved by splitting the signal in different regions of same size to be transformed. The signal is split with the use of a sliding window. The strings obtained are inserted into a suffix tree which has k leafs per node, where each one of these k-leafs value is one of the symbol from the k element chosen alphabet. Every time a new string is inserted, the frequency of the related path is updated by increasing its thickness. By using this technique, it is possible to identify motifs just by locating the thick paths in the suffix tree. The same methodology allows the location of anomalies by identifying the thin paths.
The authors propose even an extension to the suffix-tree in the diff-tree, which allows different time series comparison. It is obtained by computing the difference between the branches thickness of the two time series suffix trees. The thick paths inside the suffix-tree represent anomalies.
To conclude, the works propose an interesting technique to quickly perceptually locate patterns and anomalies in the data, able to scale when dealing with massive datasets. A screenshot of the tool is visible in Figure 2.3.c.

2.3.11 ATLAS, maintaining interactivity while exploring massive time series

In [5], Chan et al present ATLAS, the most complete system we encountered so far. It combines high performance databases with predictive caching and LOD (Level Of Detail) handling, both solution coming from computer graphics and database systems. The authors outlines three necessary features into a visualization tool: the possibility of using filters, aggregations and trending over the data; the load balancing when dealing with massive datasets; the responsiveness even when big data fetching is requested. Several techniques from database systems are proposed, like the use of column-oriented database, in order to boost operations like data aggregation, and load balancing. From computer graphics, is taken the principle of Level-of-Details; according to it, different levels of details are loaded in relation to the current zoom and the space available in the viewport. By doing this, the workload is moved from the client to the server.
ATLAS supports scrolling, zooming, panning, filtering, ordering and grouping of time series, which are displayed through line or bar-chart. Many of these operations are constrained, so the interactivity it is not affected. An example of this is the panning, it is constrained to a maximum speed in order to keep constant the tool responsiveness. Moreover, the tool uses many expedients to boost the performances, as the prefetching (same concept as in XmdvTool, i.e. if the user at the time t is scrolling from left to right, probably even a the time t+1 it will be scrolling from left to right) or the loading of data which are actually visible in the client window. To keep at the best the responsiveness, the system keeps active the interactivity with the user even when data is been loaded, marking with apposite labels the sections which are waiting for data.
ATLAS seems to us the most effective tool under the responsiveness point of view. It does not allow time series composition and components usage as Chronolenses, but it combines optimal solutions on both client and server side, which make it probably the most scalable tool we looked at. There is a big aspect we are concerning about: the data aggregations necessary to the LOD it is performed on-the-fly by the server, it might compromise the responsiveness when using really big time series.

2.4 Open platforms for sharing and analyzing dataset and experiments collaboratively

In this section, we briefly review the online platforms which allows algorithms and experiments sharing. Given the extent and the heterogeneity of the considered environment, we narrow it to the only platforms which collect and organize data related to machine learning and data mining tasks.
The UCI Machine Learning Repository[1] is one of the several repositories which col-

[1] http://archive.ics.uci.edu

lect databases, domain theories and data generator shared by the machine learning community. MLOSS[2], the Machine Learning Open Source Software is a Java library which implements several machine learning tasks.
The MLComp[3] platform, which collects algorithm and dataset, exhibits characteristics similar to what we propose. By supplying several datasets, it allows the comparison of user submitted algorithms with the ones already uploaded into the platform. The user target is the complete machine learning community, so it does not supply any time series focused tool, which allows for explorative and iterative analysis.
OpenML [23], combines the functionalities supplied by MLComp with new ones, but it does not allow user-submitted algorithm execution. OpenML[4] proposes a standard format to describe experiments configuration, algorithms and results. This format relies on Expose, an anthology which gives a formal model to describe different types of experiments. Within the platform a set of APIs is provided, by allowing automatic data upload. Finally, the tool allows direct SQL interaction with the underlying database. Many other web-platforms can be found, but the majority of these collects result achieved with the standard algorithm configurations, which constrains reusability and usefulness of them.
Differently from all the aforementioned platforms, our work supplies a repository where experimental results and algorithm are collect, but at the same time provides an effective interface to explore and navigate the data.

[2]http://mlos.org
[3]http://mlcomp.org
[4]http://exdp.cs.kuleuven.be

Chapter 3

TimeViz

This chapter analyzes all the technical aspects related to our software implementation and outlines the motivations behind every choice we made. A briefly description of the functionalities provided, their implementation and their usefulness will be given. We will mention even some functionalities and strategies we did not adopt, explaining the reasons of such a choice.
Furthermore, we will discuss some experiments and studies we performed over several segmentation techniques available in the literature. As result of these analysis, a model selection technique is proposed. It performs model selection in order to approximate different regions of a given series using different segmentation strategies, according to some each one of these regions exhibits.

3.1 Overall architecture

Here, we describe the overall architecture of the platform, mentioning solution and technologies adopted. The last part of the section describes the main functionalities and components provided by the tool. The description of the data mining components is left to the next chapter, where we propose an implementation of an anytime motif discovery algorithm.

3.1.1 Client-server web-based solution

The time series visualization has been proven to be an effective way to exploit the human capability of abstraction, so it has become a really-used approach in a lot of areas as different as medicine, web, telecommunications, financial market, sensor-network. In many of these scenarios the displayed and analyzed data is really sensitive (medicine is just an example), so a safe and centralized location where safely store the data is, in our opinion, a prerogative. Moreover, on daily basis the size of this data is exponentially growing and would be really inconvenient, often impossible, to move them across different laptops/desktops. These are the main

motivations which strongly enforced us to adopt a client-server solution, where the former serves only the necessary data to the clients, allowing safety, portability, ubiquity and scalability. We have to notice that really few of the tools previously analyzed in chapter 1, adopt a solution similar to our. We are really concerned about the scalability of those solutions with series coming from projects like Infrawatch [13], where about $5GB$ of data is daily collected. Just one year of data from the aforementioned project is about $1.5TB$, which would not fit in the drive of the majority of the standard laptops. In our opinion, the adopted client-server solution provides a relevant set of convenient characteristics: *safety*, all the sensitive data is stored in a safe and centralized server, which performs authentication and privileges checking; *portability*, the client runs on web-browser and uses all standard web components; *ubiquity*, the same reasons which makes it portable, makes it ubiquitous as well; *scalability*, the platform copes with any data size and any client viewport; *usability*, it does not need any installation procedure and presents a simple and clear user interface.
The client accesses the data by using a set of RESTful APIs provided by the web-server. Every call returns JSON packages. We will not examine in depth the APIs implementation being this just a technical aspect which is aside from our dissertation. But, we mention that every call is performed by using AJAX, and polling techniques are adopted every time computationally expensive tasks are submitted.

3.1.2 Technologies and overall design

The platform so far presents many functionalities and provides a good overall stability, but we consider the work still at a prototyping phase. Due to this, we mainly-used the programming language Python in the back-end development. We know that Python it is not one of the languages with fastest execution time, but it allows for easier and faster prototyping than the majority of existing ones. Specifically, the server relies on Django, a high-level Python web framework which strongly drives the user to the use of clean and pragmatic design, based on the Model View Controller pattern. The final outcome is a not blocking server, which keeps running and serving the data even when exceptions are generated. The software bottlenecks, as the motif discovery algorithm, have been implemented using C/C++.
The server implements a wrapper over the data storage, easily expandable in order to allow the communication with any database management system or data format. The data used in our experiments, is stored by using HDF5 data model, supplied by the pyTables Python library. Several motivations drove us to such a choice: HDF5 is a standard format; it is really suitable when working with numeric data, like time series; it allows a good data compression; it is organized in a filesystem like structure, which makes the data easy to be accessed and managed.
The client runs over the standard web browser, therefore in the development of it we used standard web libraries, languages and protocols such as: HTML, CSS, AJAX, Javascript, JSON. To give a well-defined structure we relied on AngularJS, an open-

source JavaScript framework, maintained by Google. It assists us with running our single-page applications, forcing us to adopt the same Model View Controller (MVC) pattern adopted with the server. All the data visualizations and interaction relies on the pure JavaScript library Highcharts. We made several modifications to it, especially, to add buffering and prefetching features.

3.1.3 Functionalities

From a tool, which allows the interactive and explorative data visualization we would expect data import/export functionalities and experiments sharing capabilities. Here, we describe how we achieved these requirements.

3.1.3.1 Data export and import

What we noticed from our personal experience is that the domain experts are used to perform simple tasks, like data export, in naive ways. Often we saw them manually copy-pasting portions of data from files to other. Given that, we believe data-export a really necessary feature into a visualization tool. We implemented a solution which does not only allow data export for any given time-range, but which allows the resolution selection as well. The generation of the CSV export files is performed server-side and it could require several time. So, to keep the user awareness and interactivity the client adopts polling techniques which constantly check the operation status. We set some constraints regarding the maximal size of the exporting file. In the cases, where the time-range and resolution combination exceeds a server-set threshold, we force the user to decrease the export resolution. Every generated file is temporary kept on the server until the requesting-user download it. Our platform aims to be a place where users can share and analyze experiments collaboratively. With such an aim driving our mind-set, in order to improve user experience and to boost the tool utility, we implemented data import functionalities. Any user can create and share collections of data-sets and import time-series into them. The tool graphical interface allow the user to set several data import configurations. Figure 3.1.a and Figure 3.1.b depict the graphical interfaces supplied to export and import datasets.

3.1.3.2 Sharing functionalities and data annotation

Every time a user submits a task computation to the server, the results are permanently stored and become explorable by the all users. This is an effective functionality when working and analyzing collaboratively the same data. Moreover, a unique location where all the experiments results are stored and collected, is a productive way of organizing and tracing the work done.
What we propose is an effective tool to explore massive time series. In such a context, would be useful to be able to store interesting views we found while analyzing

the data. Would be even better to be able to share these with other user. Our tool satisfies this by implementing a permanent links generator, which generates static URLs linking to views representing data the users found interesting. Each one of these URLs is composed of a unique hash. We adopted such a solution because an URL is easy to be shared and stored, it allows the users to refer and share interesting data portions just by using string. Each permanent link has associated some explicit and implicit information: time series and time-interval of the displayed data; date and user of creation; some notes related to it, like the motivations of interest. In order to trace of all the findings, all the generated links are stored and can be consulted by all the users.

3.1.3.3 Ubiquitous, scalable and portable solution

The solution we propose is a client-server web-based solution, where the client uses all standard technologies and libraries. Moreover, as we will explain later on, the points loaded into the client are always related to its viewport and so only trivial aggregation need to be performed. Mobile and other devices with weak hardware performances will always have to deal with constrained data, renderings and computations. All this characteristic make our work usable with any devices and from everywhere, de facto making it ubiquitous, portable and scalable to any dataset.

3.1.3.4 Data buffering

The charting library we use, fully reloads the new data from the server, acting a busy wait, every time the user performs a zoom or scroll operation on the plot. In order to avoid this and ensure a more fluid user experience, we implemented a client-side data buffering. The basic concept we adopt comes from video-games environment: the client has always loaded several versions of the currently displayed data, which defer each other in the data resolution and temporal-range. We call this parallel versions of the displayed data, buffers. The total number of buffers is a customizable parameter. Let's say b is the buffer containing the currently displayed data, the client always has two buffers list, respectively *back* and $front$. The first buffer in *back* contains data in the same temporal-range of b but at double the resolution. Each successive buffer in *back* contains data in the same temporal-range but at double the resolution of the previous one. Instead, the first buffer in $front$ contains data in a temporal range three times larger than b and centered in b. The resolution of this data is half the one in b. Each successive buffer in $front$ contains data with half the resolution of the previous one and into a temporal-range three time larger. By adopting this solution, every time a zoom-in, zoom-out or scroll operation is performed, there are high probability that the data is fetched from one of the buffer, avoiding so a data-loading busy wait. Clearly, the $front$ and *back* lists need to be updated every time one of the mentioned operations is performed. More buffer we have, higher is the probability of already having the data, but higher is the client

memory consumption as well.
After a deep testing, using data from totally different contexts, we can confirm that the advantages achieved by using such a buffering technique are totally smoothed by the rapidity our tool has in loading the data. So, the default configuration of our software has the data buffering disabled. We advise to enable it only in context where the infrastructure bandwidth is limited.

3.1.3.5 Data range bands

A direct consequence of displaying massive dataset in constrained viewports is the loss of many local information. Let's say we are displaying into a screen of a thousand pixels width, a time-series originally composed of a million of points. All the sporadic peaks in the series will be lost and omitted from the visualization. We could trace these peaks by using segmentation techniques which focus the approximation on the peak-preserving, as the Ramer-Douglas-Peucker, but as side effect we would have an overwhelming representation of the signal. To give the user, the awareness of the range inside which the original series is distributed, we store maximum and minimum values inside each aggregate interval and display it as max and min bands containing the approximated signal. The next section contains a deeper analysis of the problem.

3.1.3.6 Data Mining tasks

The tool implements a set of data mining tasks such as motif discovery, correlation matrix computation and similarity computations. All of them exploit a hierarchy-based visualization in order to prune the computation. Specifically, we propose an anytime motif discovery algorithm which return at any moment the best top-k matches found so far. The next chapter describes with more details the results we had and experimentations we did during the implementation of this algorithm.

3.2 Visualization

When working with high dimensionality datasets, all the explorative tools have to tackle a big issue: how is it possible to narrow down the size of the visualized data and keeping their meaning? How is it possible to do this in smart and effective way? The majority of the tools tackle the problem by using a naive approach: given a client viewport and a time-interval to be displayed, all the points in the range are aggregated on-the-fly according to the number of points in the viewport. Clearly this approach totally compromises the scalability of the tool. Let's say we have the same scenario presented before: a viewport of a thousands of pixels and a time series of one million points. It is pretty obvious that the approximation of such a series to a thousands of points is extremely time consuming, and cannot be done in a period of time which does not affect the tool interactivity. In VizTool, we tackle

the problem by making preprocessing of the data. Precisely, we built a hierarchy of the data at different resolutions.
This section introduces the hierarchy-based visualization solution we adopted. The second part presents a model selection technique to efficiently approximate large time series.

3.2.1 Hierarchy based visualization

The solution we propose adopts a logic similar to the one adopted to Google Maps. What we experience when browsing through a map, is that at any instant the displayed image resolution is directly related to the zoom level, i.e. more we zoom, higher is the ground resolution. Our solution does exactly the same: lower is the displayed temporal-interval, higher is the resolution of the loaded data. This a technique widely-used in the video games field, where closer is a rendered object, higher are the polygons which compose it.
So, on-the-fly aggregation of large time-series is too expensive and totally compromise the interactivity and responsivity of the software. We tackle the problem by adopting an approach we did not find anywhere else in the literature. Every time new data are imported into the platform, a sub-sampling hierarchy is built. Each hierarchy level contains the data approximated at a different resolution. Given a time-series composed of n sampled values, the structure contains $log_2(n)$ segmentations. Each one of these segmentations is composed of $n/2^i$ points, where $i \in [0, log_2(n)]$ is the level number.

Hierarchy summary	
Computational cost	$O(n)$
Total levels	$log_2(n)$
$\|level_0\|$	original data
$\|level_i\|$	$\|level_{i-1}\|/2$
$\|level_{log_2(n)}\|$	1

Specifically, given any i_{th} hierarchy level, each point in it is the aggregation of 2^i points in the original sampling. Such aggregations are performed by using arithmetic average of the points in the intervals. That is to say that every level is approximated at half the point of the previous one, by using Pairwise Aggregate Approximation. So, any point x in the i_{th} level is the aggregation of the points in the interval $[x \cdot 2^i, (x+1) \cdot 2^i]$, where $x \in [0, n/2^i]$. In order to avoid the propagation of the computational error, every approximation is performed using the original data, even though it can be more time consuming.
Any given time series will always be displayed in a viewport composed of a-defined number of pixels. Such a straightforward observation completely motivates the reliability of our solution. The basic concept is that it does not make any sense to load more points than the pixels in the viewport, otherwise data aggregation is needed. The best scenario is the one where we have approximately one point

per pixel. In our solution, we can assure this by fetching the data from the level containing approximately a number of points, in the selected time-range, equals to the viewport ones. We assure that by fetching the data from the level:

$$R = log_2(\frac{expected_{values}}{w}) \tag{3.1}$$

Where w is the number of pixels in the viewport and $expected_{values}$ are the expected values in the selected temporal-range, computed as the temporal-interval times the series sampling rate. The expected value are an approximations, due to the fact that the sampling rate is not always constant and there could be missing values.

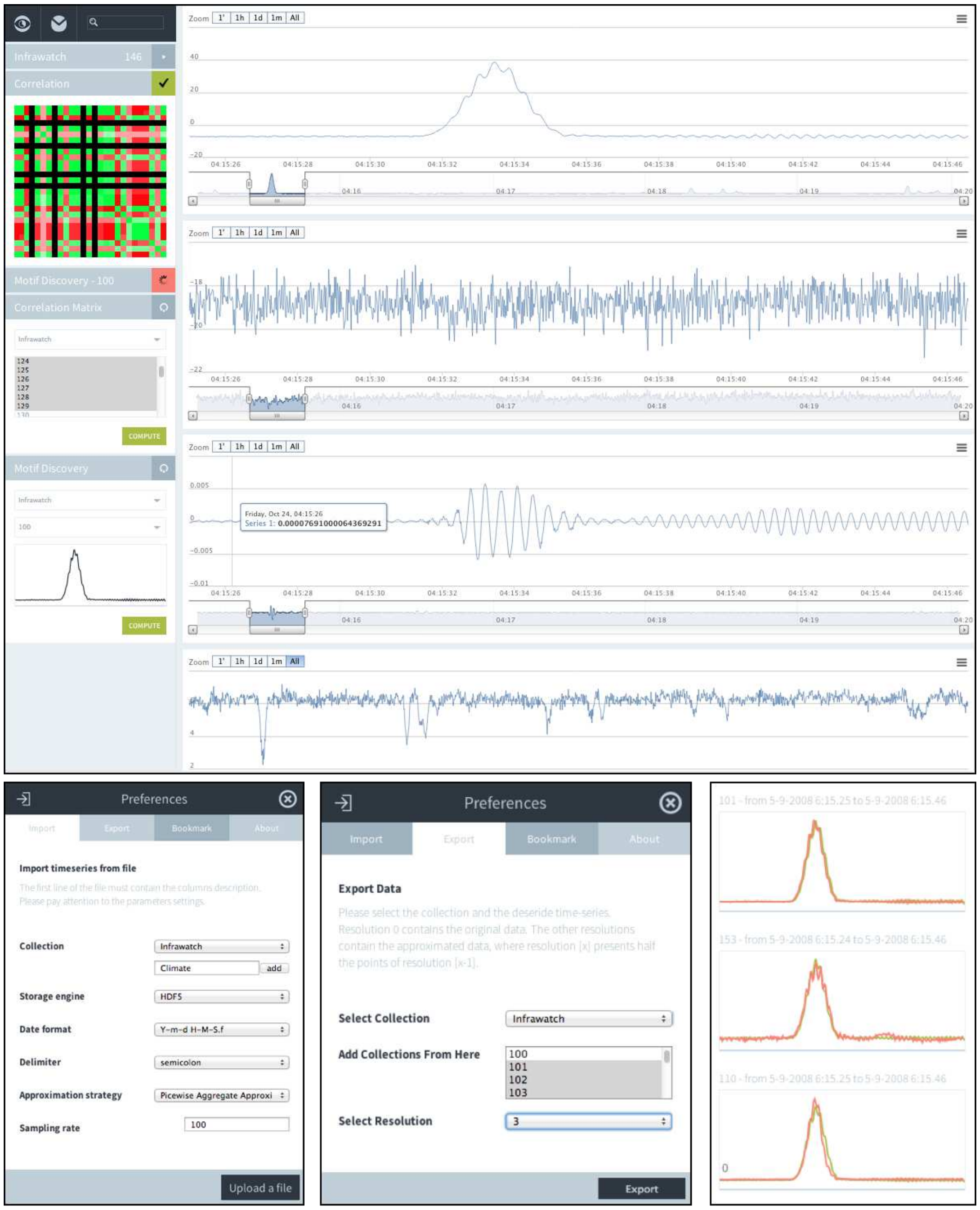
Algorithm 1: Subsampling Hierarchy.

```
Data: T, time series of length $n$
Result: H, sub sampling hierarchy composed of $log_2(n)$ levels
$i \leftarrow 0$;
$sum \leftarrow 0$;
while $i \leq log_2(n)$ do
    for $j$=0 to $n$ do
        $sum \leftarrow sum + T[j]$;
        if $j \mod 2^i = 0$ then
            $H[i][j/2^i] \leftarrow sum/2^i$;
            $sum \leftarrow 0$;
        end
    end
    $i = i + 1$;
end
```

What we propose and adopt is a fully scalable solution, which copes with any data size and device viewport. Our platform allows for interactive exploration of the data by decreasing resolution and size of massive time-series according to the current zooming level.

3.2.2 Locating subsequences

The segmentations of the original series are obviously stored according to their chronological order. Moreover, each one of these segmentations has an index. Every time we fetch the data related to a given temporal-range, from one of the hierarchy level, we need to look for the position of this interval in the approximation. Basically, we need to find the index of the first value in the range. The inconsistency of the sampling-rate, prevents us from computing that index as the sampling rate times the gap, we have between the interval starting date and the approximation starting date. It would have constant time $O(1)$. We need to compute the index in an



Import timeseries from file

The first line of the file must contain the columns description.
Please pay attention to the parameters settings.

Collection	Infrawatch
	Climate add
Storage engine	HDF5
Date format	Y-m-d H-M-S.f
Delimiter	semicolon
Approximation strategy	Picewise Aggregate Approxi
Sampling rate	100

Export Data

Please select the collection and the deseride time-series.
Resolution 0 contains the original data. The other resolutions contain the approximated data, where resolution [x] presents half the points of resolution [x-1].

Select Collection	Infrawatch
Add Collections From Here	100 101 102 103
Select Resolution	3

Figure 3.1: **VizTool screenshot. a.** The right side depicts the massive time series visualization structure. The left menu allows the usage of several data-mining components to the displayed time series. **b.** Data import form detail. **c.** Data export form detail. **d.** Detail of motifs discovered by using the tool.

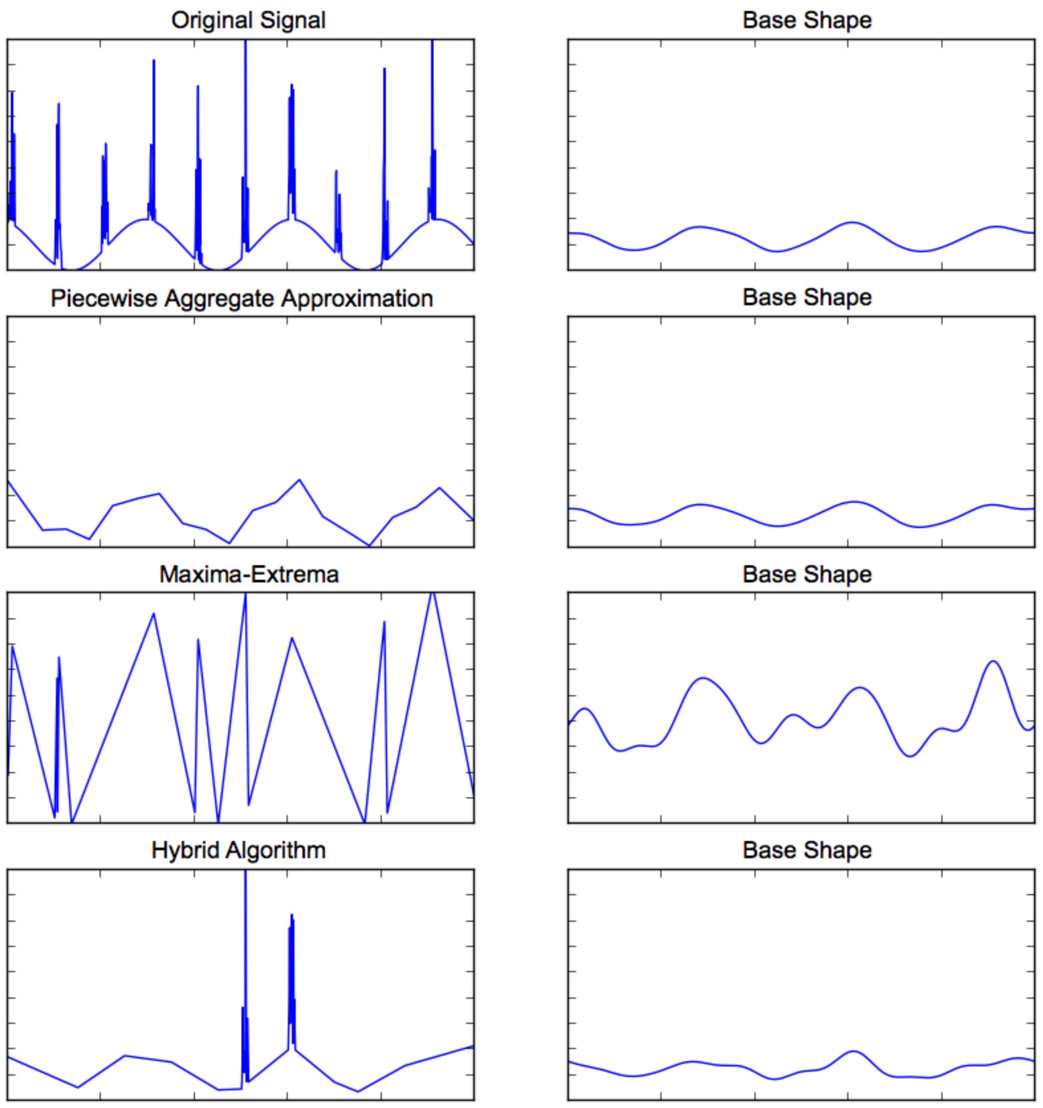


Figure 3.2: **Dominant frequencies loss.** The approximations obtained by using peak preserving algorithms miss the dominant frequencies of the original signal. The right side of the chart plots the frequencies retained by applying a low-pass filter to the approximated series.

alternative way. Scrolling through all the data and checking each point timestamp seems to be so expensive $O(n)$. We have to remember that low levels keep to be massive and so, an $O(n)$ operation compromises the interactivity. Therefore, due to its logarithm cost $O(log_2(n))$ and the easy implementation, we perform such a research by using the Binary Search. It needs to be pointed out that with just 30 iteration we are able to scroll a massive series of 1 billion points.

3.3 Data-adaptive hybrid approximation

In the process of building the sub-sampling hierarchy, we use the Piecewise Aggregate Approximation segmentation algorithm. Our experiments show how in comparison to many other algorithms, such an algorithm keeps low the error between the approximated signal and the original one even at really high compression ratios. However, when the compression ratio increases, it gradually smooths the signal with the consequent loss of the perceptual features in it.
It has been widely proved how these perceptual features are a fundamental component to exploit the abstraction capabilities humans have. We tackle the problem by proposing a method to select a data-adaptive hybrid approximation obtained by composing diverse techniques. This algorithm produces approximations which preserve the perceptual features of the signal while keeping low the approximation error.

3.3.1 Peak preserving side-effect

The literature presents several algorithms which focus time series approximation on preserving the perceptually important points of the series. As we already mentioned, such techniques lose their effectiveness with the increasing of the compression ratio. It happens because the peak-preserving algorithms do not fulfill a basic concept: higher is the compression ratio, lower must be the local information preserved in the reduced representation. While at high compression ratios algorithms as the Pairwise Aggregate Approximation focus the reduction on the main shape of the signal, the peak preserving ones keep focusing on the peaks in the signal. Consequently, the main shape of the signal is progressively lost.
A simple and effective method to extract the main shape of any given signal is to apply a low-pass filter to it. To do so, we compute and trim to a fixed frequency the Fast Fourier Transform inverse of our signal. By extracting the dominant frequencies in the series, we evaluate the capability any given method has to preserve the main of the signal. Figure 3.2 outlines the experiments we performed with a synthetic time series of sinusoidal form, where random peaks have been added. For each one approximation techniques, the signal is compressed 100 times. As expected, non-data-adaptive algorithms as PAA completely loss the perceptual features of the signal but keep intact their main shape. Data-adaptive algorithms like Maxima

Extrema preserve the peaks, but come up with an approximation which is totally not representative of the original series.
As we will see, the algorithm we propose at high compression ratios mainly select not-data adaptive algorithms, which totally preserve the signal fundamental shape.

3.3.2 Some observations over common segmentation algorithms

Here, we outline some observations we made by using some standard machine learning techniques. We used them in the design of a data-adaptive model selection algorithm. We, briefly, review the experiments we performed and results we had. For a deeper understanding and a more complete dissertation, refer to [1].
The first step we took was the analysis of the performances in terms of approximations error, for several of the segmentation algorithms available in the literature. Clearly, given the structure of our software, we were interested in the only algorithms which produce as outcome a numeric representation of the signal. We did not consider methods like SAX, which produces symbolic representations. What we observe from such an experiment is that the not-data adaptive approximations show a really similar behavior each other.
A standard technique to improve the performances of a set of models is the ensemble of them in a unique and comprehensive model. Such a model cannot be used to approximate data: it requires the computationally expensive computation of all the ensembled approximations. But, it can be used as mathematical proof of the observation we just mentioned. The analysis of the weight each algorithm assumes, at different compression ratios, gives us a better understanding and a proof of that. Therefore, we performed a linear blending of the approximated series at diverse compression ratios.

$$x_i = w_0 + \sum_{k \in K} w_k \cdot r_{k_i} \tag{3.2}$$

Linear Blending where r_{k_i} is the i_{th} value of the reduced signal up-sampled to the original one, and $k \in K$ is one of the dimensionality reduction techniques. Within the blending operation, the following algorithms have been used: Piecewise Aggregate Approximation, Gaussian Weighted Average, Ramer-Douglas-Peucker, Fast-Fourier-Transform and Maxima-Extrema. To learn the involved weights, we minimized the regularized square error using the linear gradient descent. Figure 3.3 depicts the achieved results. It proves the insight we had by comparing the Euclidean Distance error of the several representations. Specifically, we measured how PAA, Gauss and FFT produce really similar results. As consequence of this, and of the weak results Maxima Extrema achieves at every compression ratio, we removed from the analyzed algorithm list all the redundant and weak ones, retaining only the Piecewise Aggregate Approximation and the Ramer-Douglas-Peucker algorithms.
Briefly, by using the linear blending technique, we removed from our lists al the redundant and inefficient algorithms. At this point, by plotting the distribution

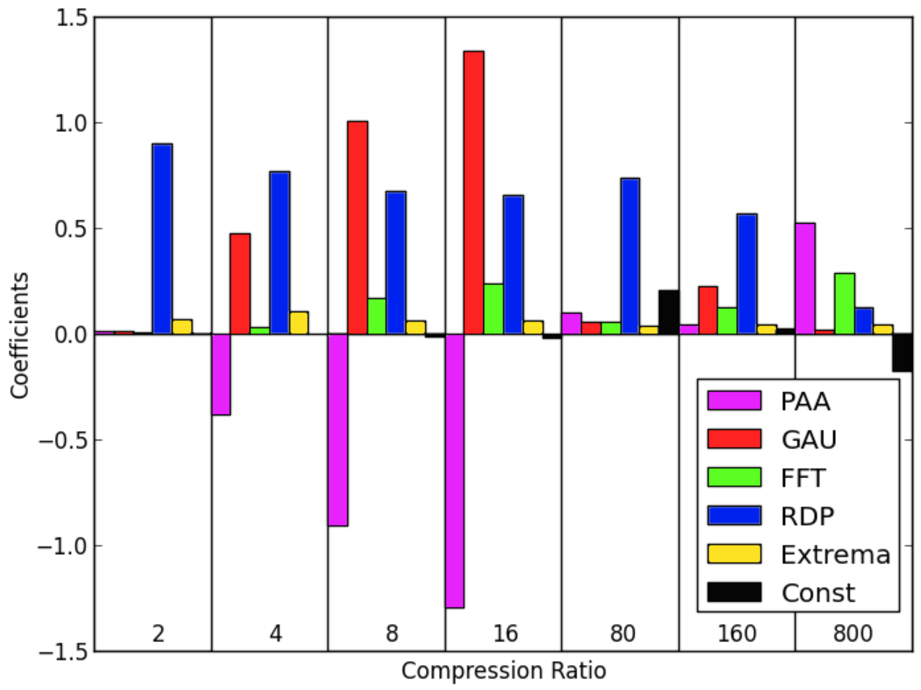


Figure 3.3: **Linear Blending.** The data-adaptive algorithms show better performances at low compression ratios. When the compression is high it is true the other way around. Moreover, non data-adaptive algorithms show similar behavior.

of several dataset, we made a basic observation: in the series often happens to have aggregation windows with low data standard deviation and other where the standard deviation is really high. It suggested us a basic idea: instead of correlating the performance each technique has to the compression ratios, we should correlate them with the features the data exhibits. According to the data variability (we have) in a given window, we should be able to identifying an algorithm more suitable to approximate the data in it.
This assumption has been tested and experimented by using binned linear regression. The idea arise from the Feature-Weighted Linear Stacking proposed by Sil et al. [8] during the Netflix competition: “FWLS combines model predictions linearly using coefficients that are themselves linear functions of meta-features” [22]. In our case, the meta-feature is the standard deviation inside the sliding windows. For any compression ratio, instead of computing a single weight per method, we computed w different weights. Basically, the aggregations intervals were split across w bins, each one containing aggregation windows with data standard deviation inside a

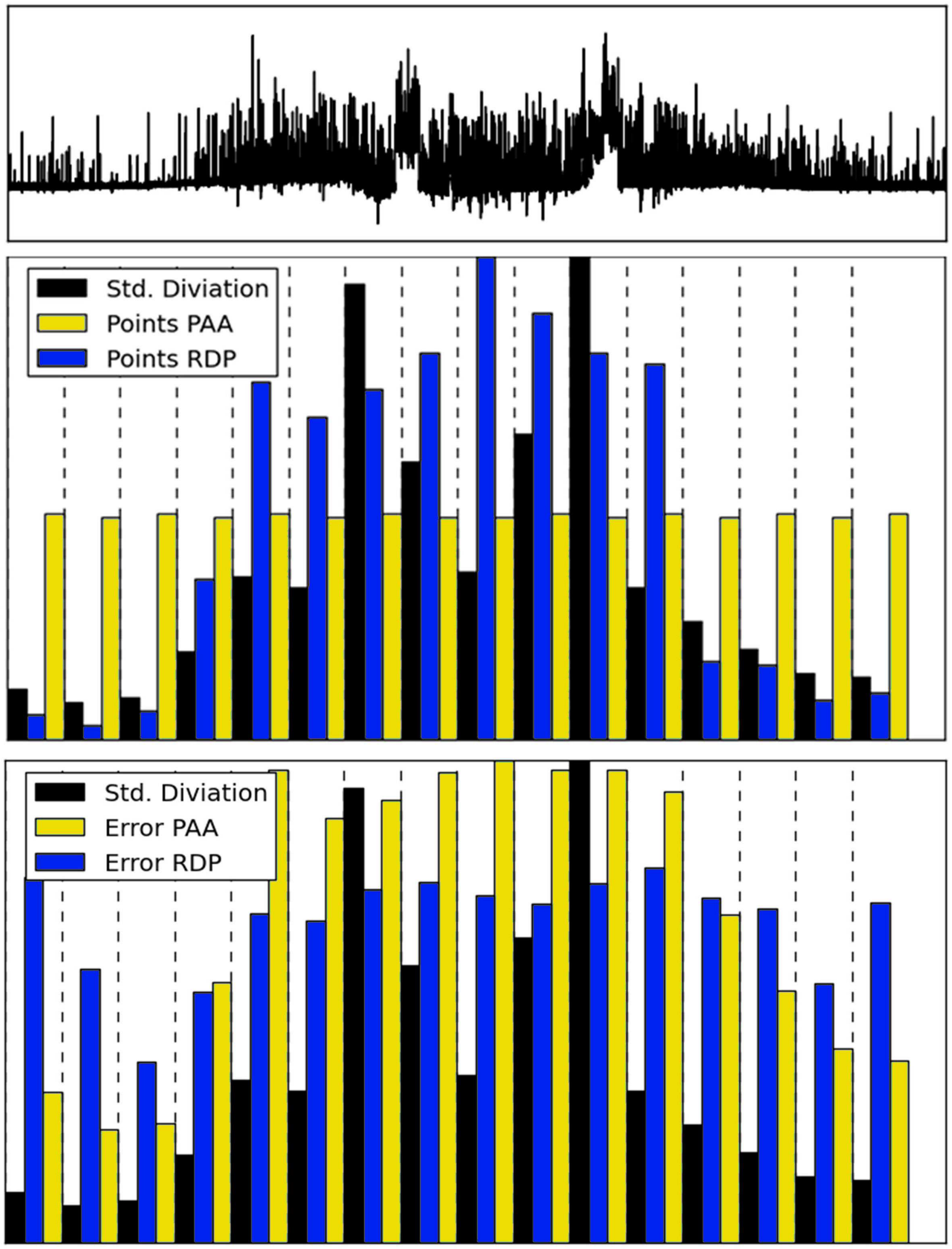
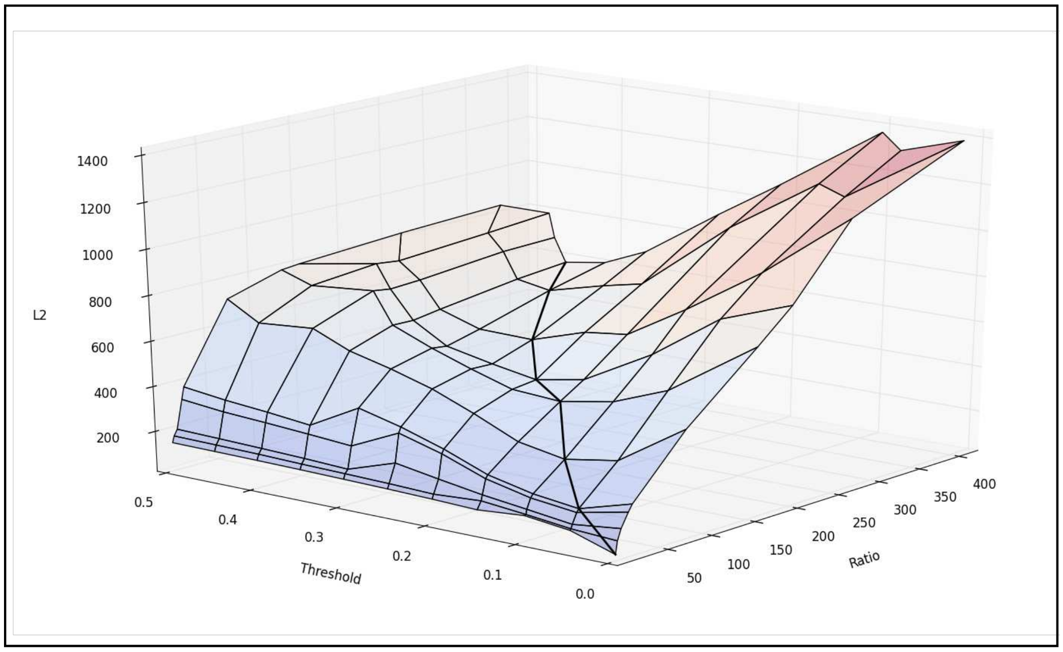


Figure 3.4: **Observations.** The first chart outlines the signal shape. The second outlines the points used respectively by PAA and RDP, according to the standard deviation in each window. The third depicts the error introduced by RDP and PAA in each aggregation window.



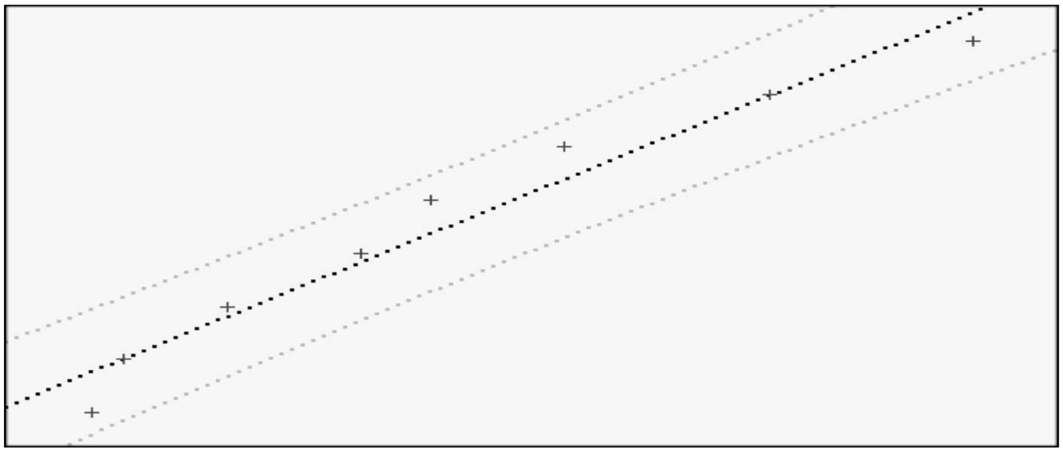


Figure 3.5: **Optimization.** The chart outlines the hybrid approximation error obtained by using different values for the *threshold* parameter, at different compression ratios. The chart below outlines the optimal parameter value, achieved by performing linear regression over the optimal path points. Computed the variance inside each aggregation window, the algorithm uses the *threshold* to select the segmentation algorithm used to approximate the region.

certain range, and later, linear blending of them was performed.

$$x_i = w_{0_{b(i)}} + \sum_{k \in K} w_{k_{b(i)}} \cdot r_{k_i} \tag{3.3}$$

Weighted Least Squares Regression where: $b(i)$ represents the bin containing the window enclosing x_i; $k \in \{0, 1\}$ represents the dimensionality reduction technique $(P_{AA} \wedge R_{DP})$; r_{k_i} is the $i_t h$ points of the k-technique reduction up-sampled to the original signal time-shape.
From the results, we observed that aggregation windows with high standard deviation are better approximated by the Ramer-Douglas-Peucker algorithm and, on the other hand, windows with low data variance are better reduced by PAA. Moreover, this performance are highly influenced by the compression ratio, at low compression ratio RDP performs better, at high is true the other way around.
This can be explained by observing that the data-adaptive approximation technique uses more points in portions of data which present high fluctuation in those points, as depicted in Figure 3.4. We can even note that the rapid change of the standard deviation between adjacent windows, it is not instantaneously propagated to the number of points used by RDP, which exhibits a slow adaptation.
We need to highlight that to observe the aforementioned phenomena in a clear and pronounced means, it is essential to select the aggregation-window size properly.

3.3.3 Hybrid algorithm

In the previous section, we outlined some observation we made. It can be summarized as follows: the data adaptive algorithm uses more points where the standard deviation is higher, and by focusing the approximation on the signal peaks, it has a higher capability of preserving the perceptual important points; at low compression ratios RDP achieves better result, while at high compression ratios PAA is better; finally, RDP slowly adapts the number of points used when the standard deviation rapidly changes. These phenomena lead us to ensemble in a single method the data-adaptive Ramer-Douglas-Peucker algorithm and the non-data-adaptive Piecewise Aggregate Approximation.
Now we know that we should exploit RDP where the standard deviation is high and PAA in the other cases, but we still have to cope with a some uncertainty. Given a series and a compression ratio, which size should have the window used to check the standard deviation ($window/ratio$)? Computed the standard deviation, how is selected the segmentation algorithm to be used ($threshold$)? How many points have to use the two techniques per window approximation $\|RDP\|/\|PAA\|$?
Several experiments we performed [1] suggest that the threshold linearly decreases with the resolution. Moreover, for any compression ratio the optimal ratio between the points used respectively by RDP and PAA is always included among 10 and 20; i.e. RDP uses from 10 to 20 times the points used by PAA. We found more complex

to establish the optimal ratio between the size of the window used to compute the standard deviation and the compression ratio. Intuitively, we noticed that at low compression ratios the optimal window seemed to have big size, while it was reducing with the increasing of the ratio.
Figure 3.5 depicts the optimal path of the *threshold* parameter.

$$Q = \{Reduce(s_1), Reduce(s_2), \ldots, Reduce(s_t)\} \tag{3.4}$$

$$Reduce(s_i) = \begin{cases} P_{AA}(s_i) & \text{if } \sigma(s_i) < Tr \\ R_{DP}(s_i) & \text{if } \sigma(s_i) \geq Tr \end{cases}$$

$$\text{where} \quad \begin{aligned} &Tr = min(V) + [max(V) - min(V)] * threshold \\ &V = \{\sigma(s_1), \ldots, \sigma(s_t)\} \\ &t \equiv \tfrac{n}{windows} \geq \tfrac{n}{ratio} \end{aligned}$$

The tests we conducted outlined better performance of our Hybrid Algorithm in almost every cases. It was partially true, because at really low compression ratios this was not happening. That was due to the fact that at low compression ratios our algorithm was splitting the series in small windows. But, we observed as RDP works better with a unique big aggregation window, due to the fact that it is ables to catch the correlation between the bounding data of the adjacent windows. Therefore, at this point we added a further step to our algorithm: after having selected the segmentation technique for every aggregation window, we merged all the adjacent windows, to be approximated with RDP.
The last step we conducted was the optimization of the aforementioned parameters, as visible in Figure 3.4 and Figure 3.5 we used linear regression in order to optimize the threshold in relation to the compression ratio. We stress that their correlation coefficient is approximately equal to 96%, so, it really makes sense to linearly map the ratio-threshold correlation. Other experiments suggest how the ratio between the point used by RDP and PAA slightly affects the final results. So we suggest to use an optimal ratio of 20.

3.3.3.1 Pseudo-Code

Here we present the pseudo-code of the data adaptive hybrid model selection algorithm we propose.

Algorithm 2: Data-adaptive hybrid algorithm.

```
Data: T, time series of length $n$. $n$, the length of the returned signal.
$w$, sliding window inside which the standard deviation is checked.
$rdpPoints$, ratio between the point used per window by PAA and RDP.
$threshold$, parameter used to perform the model selection.
Result: A, approximated series
hybRatio ← len(T)/(n/w);
V ← [];
count ← 0;
H ← [];
rdp ← 0;
paa ← 0;
for i=0 to len(T)/w do
    V[count] ← $\sigma$(T[i*hybRatio:(i+1)*hybRatio]);
    count ← count + 1 ;
end
tr ← min(V) + ((max(V) - min(V))*threshold);
for i=0 to len(V) do
    if V[i] $\geq$ threshold then
        regionRDP ← rdp + 1;
    else
        regionPAA ← paa + 1;
    end
end
unit ← len(T) / (paa + rdp · rdpPoints);
paa ← round(unit);
rdp ← round(unit*rdpPoints);
count ← 0;
while count ¡ len(T)/w do
    a ← count;
    while (count + 1) < len(T)/w and V[count+1] $\geq$ threshold do
        count ← count + 1;
    end
    if V[a] $\geq$ threshold then
        H ← H + RDP(V[a·hybRatio:(count+1)·hybRatio],rdp*(count-a+1) );
    else
        H ← H + PAA(V[count·hybRatio:(count+1)·hybRatio], paa);
    end
end
return H
```

It accepts as input: a time series T; a sliding window w inside which the standard

deviation is checked; a ratio between the point used respectively by PAA and RDP segmentation techniques; a threshold used to perform the model selection. The final outcome is a representation of T, approximated by n points.
The first step the algorithm performs, is the computation of the data standard deviation for each one of the windows of length m, composing the signal. According to this, the number of windows to be segmented, respectively by using RDP and PAA approximation techniques, is calculated. By using it as reference, the number of points to be used per unit are established. At this point, the adjacent windows to be split with RDP are merged together and the signal is approximated. PAA uses a number of points per window equal to the unit, while RDP uses a number of points equal to the unit times the RDP/PAA ratio parameter.

Chapter 4

Resolution-aware motif discovery

Time series motif discovery is the task of finding unknown patterns from time-series data. Specifically, we define a motif every subsequence that appears recurrently in a longer time-series [7]. Figure 4.7.a illustrates an example of motifs discovered in an industrial dataset, where data from a wire winding process are collected [20]. Since is formalization, motif discovery has been used by a lot of researchers in a big variety of domains like biology, medicine, telecommunications, sensor network. Such a relevance comes from the utility the technique has in the process of generating insights about the system behavior, from a domain expert point of view.
The problem is usually tackled into two different scenarios: one copes with a time-series dataset and looks for recurring time-series in it; the other looks for recurring subsequences in a unique time-series. We will always refer to the second scenario, which is more general and more challenging, since it deals with more problematics aspects, as the trivial matches or the not-overlapping matches condition. As previously mentioned in the related work, two are the mostly-used formalizations of the motif discovery problem. The first looks for the top-k motifs ranked according their number of instances, i.e. the number of times the motif appears in the time-series. Basically, it researches subsequences or time-series which score, an each other similarity, contained inside a given range (the literature usually defines it as R). This formalization aligns the motif discovery problem to the clustering one. The second formalization looks for the top-K subsequences pairs in the time series, ranked according to their similarity. Therefore, it considers only pairs of windows and does not take into account the motif number of occurrences. We formalize the problem we want to solve as:

Top-k pairs query. Given a time-series $T = \{t_1, t_2, .., t_n\}$, a top-k pairs query $Q(k, w)$ takes two parameters k and w, and considers P the set of all possible not trivial matches of length w in T. The query $Q(k, w)$ returns an answer set from P

that consists of k pairs, such that for every p in the answer set and for any other pair $p' \in P$, $p.score \leq p'.score$ and p is not overlapping with p'. The score are computed by using the Euclidian Distance similarity measure after each subsequence is normalized, in order to remove offset and scaling effects.

The problem has been tackled by adopting mainly two approaches: the exact motif discovery and the approximated motif discovery. The first relies the research on the original data and mathematically guarantees the return of the best matches. Instead, the second relies the computation on an approximation of the data, and so guarantees the return of a list of matches, but cannot guarantee that they are the best even in the original representation of the series. The deep research in the approximated motif discovery field is motivated by the quadratic cost of such a task. The obvious exact match discovery, given a time series T of length n and a motif of length m, costs $O(n^2m)$ operations. Several algorithms have been proposed, which use bounds on the similarity function with the aim of pruning the computation. The idea is that it is possible to use the knowledge gained from the previous matches computation, in order to stop a current match computation, as soon as it results to be not promising. The problem of such algorithms is that they keep having quadratic cost in the worst case, and many of these do not deal with the condition that the top-k matches list has to contain only not overlapping motifs.
From our experience, we can observe that in the majority of the cases the domain expert looks for a briefly overview of the motifs in the series, while in just few of these he looks for the exact motifs list. Therefore, we do not consider a reliable solution the implementation of an exact motif discovery algorithm, into a tool which allows interactive and explorative time series analysis, since the user would always have to wait a big amount of time to get the results. However, not even an approximated motif discovery is the right choice in our opinion, since such a solution does not allow for exact results in any case.
Both exact motif discovery and approximated discovery provide the answer after a fixed amount of time, and either run to completion or they not provide any result. However, frequently, it may would be useful to terminate the algorithm before completion. It is possible to do such an operation and have significant results by using anytime algorithms, which return results whose quality depends on the amount of time they have been able to run. Given this, we experimented and consequently implemented in our software a solution which combines both the approximated and exact algorithms: an *anytime* motif discovery algorithm which step by step refines the results converging to the exact ones. A solution which adapts the execution time to the user needs.

Anytime algorithm. An algorithm that even if interrupted at any moment before the end, returns a valid solution to the problem. Moreover, it exhibits an evolutive behavior, which progressively improves the returned result.

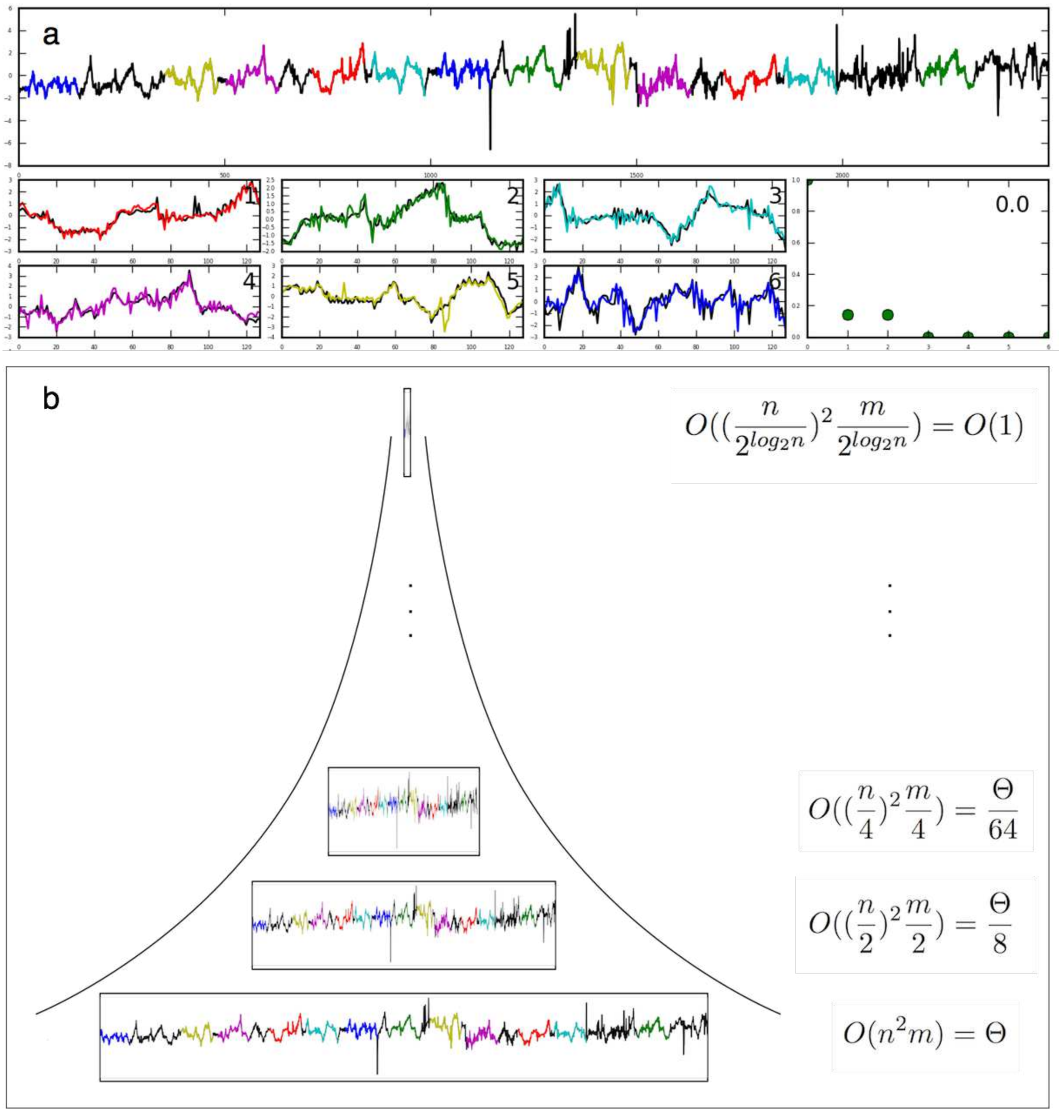


Figure 4.1: **a. Motif discovery example.** Motifs discovered in an industrial dataset, where data from a wire winding process are collected. The motifs have been discovered by using the exact match discovery algorithm. The fifth discovered motif (yellow) depicts the importance of subsequences normalization, in order to remove offset and scaling effects.
b. Anytime resolution-aware motif discovery. The algorithms step-by-step discovers the motifs over all the hierarchy levels. It starts from the last level in the hierarchy and progressively climbs it. After the first level has been performed, the algorithm returns result anytime is terminated. Every step takes $O(8)$ times the operations performed in the previous one.

4.1 Anytime resolution-aware motif discovery

We propose an anytime algorithm which solves the top-k motifs discovery problem. Each time is terminated, it returns the best-so-far top-k discovered matches. It exploits the sub-samplings hierarchy adopted by our tool, where the first level contains the original series and any other level contains the original series subsampled at half the points of the previous level.
The concept behind the algorithm is that we can solve the top-k motif discovery problem by finding out the motifs over approximations of the original signal, which step-by-step are increased in their resolution with the consequent results improving. The algorithm starts the motif discovery from the second to last level in the hierarchy, composed exactly by two points. Such points are the best match at the time $t = 1$. The second step goes up a hierarchy level and computes the motifs in it. The third and any successive level do the same. Therefore, step-by-step the algorithms climbs the hierarchy till the last level containing the original data is reached. At this level, the exact matches discovery is performed. Given any step s_i which looks for the motifs of length w, the previous step s_{i-1} looks for motifs with half the length, since it presents the signal approximated with half the points of the ones in s_i. Basically, at each step the motif length is doubled.
The algorithm is anytime, because after the first step has been done, it can be terminated anytime and still produces results. We want to stress that our approach step-by-step converges to the exact match solution, so it is an exact algorithm when it runs till completion.
If we analyze the complexity, we notice that the last step has a worst computational cost equal to $O(n^2m)$. The previous one, since works over an approximation composed of half the points, presents a worst case computational cost equal to $O((n/2)^2(m/2))$. The same consideration is valid for every step taken by the algorithm. Therefore, every step requires $O(8)$ times the computations performed in the previous one, approximately one order of magnitude.
We have to observe that in many cases the starting point is not the last level in the hierarchy, but the highest level i_{th} such that, given a sliding window of length w in the original data, we have that $\frac{w}{2^i} > 1$; i.e. the motifs lengths at each step must be greater or equal to that one.
Given a time series T of length n and a sliding window of length w, used to discover motifs, our algorithm computes the motif discovery in a decreasing order over all the hierarchy levels such that:

$$\frac{w}{2^i} > 1, \quad i \in \{0, 1, .., log_2(n)\} \tag{4.1}$$

Given a hierarchy level i and a time series of length n, every step has a computational cost equal to:

$$O(\frac{n}{2^i}^2 \frac{m}{2^i}) \tag{4.2}$$

4.1.1 Induced pairs analysis

An anytime task can be terminated at any moment, and it should always returns the best results it has been able to discover during the execution time. In our case, if the algorithm termination is invoked while this is computing the motif discovery over one of the hierarchy levels, for such a level we would have just partial results. An obvious idea would be to merge the results obtained from the partial level analysis with the one gained at the previous step, where the full discovery process over an approximation with half the resolution was conducted. That it is not possible, since we cannot compare matches obtained from different hierarchy levels, being their Euclidian Distance differently scaled.
But, we can observe that there is a high probability that the top-k matches obtained at the step s_i are similar to the ones obtained at the step s_{i+1}. So we can use the ranking achieved at the step s_i to induce the order used to check the matches at the step s_{i+1}. By doing this, in all the cases where the interrupted step computation has checked at least k matches, we can use this more accurate partial results, which are based on the previous step ranking, but reordering it according to the computation performed over a more accurate signal segmentations.
We can clarify the idea with a simple example. Suppose a scenario where we are looking for the top-4 motifs in a massive time-series. Our anytime algorithm has been terminated at the step s_i, after he has been able to check the first 10 windows pairs, following the order induced by s_{i-1}. At this point, our solution returns the top-4 element obtained by re-ordering, according to more precise calculations, the top-10 matches list achieved at s_{i-1}.
That is not enough. The total pairs of windows to be compared at any step s_i are related to the motifs length and approximated data length. So, they are equal to:

$$|P_{s_i}| = \frac{(n-m)(n-m-1)}{2}$$

These are the induced matches at any step s_{i+1} which presents doubled motifs and approximations length in comparison to s_i, and so a number of total matches equal to:

$$\left|P_{s_{i+1}}\right| = \frac{(2n-2m)(2n-2m-1)}{2}$$

$$\frac{\left|P_{s_{i+1}}\right|}{|P_{s_i}|} = \frac{4(n-m)(n-m-\frac{1}{2})}{(n-m)(n-m-1)} \approx 4$$

Every step presents approximately O(4) times the matches induced by the previous one. Therefore, after having compared all the induced ranking, the algorithm for each step compares linearly, from left to right, the remaining matches.

Induced matches comparison order. The order used to compare pairs of windows at each step, is induced by the ranking achieved at the previous step. After the induced ranking computation is completed, the remaining matches are computed following the linear order from left to right. The first step in the algorithm compares all the possible matches, by using a sliding window which moves from left to right.

4.2 Pseudo-code

Here, we present the pseudo code of the anytime motif discovery algorithm we propose. The two functions used to rank the subsequences matches are presented as well. We stress that in order to maintain compact the code, the outlined algorithm does not perform the additional step which checks and removes the overlapping motifs, in the rankings returned by LinearPairRanking and InducedPairRanking. Such operation can be done by scoring the rankings from left to right and remove each pairs overlapping with one or more of the ones previously encountered.
The function *norm* normalizes each subsequences passed as parameter. Such operation is necessary in order to avoid offset and scaling effects. The function L_2 computes the Euclidian Distance between the two passed subsequences. In this version of the algorithm, it adopts *early abandoning* in order to prune the computation.
We do not report the code which makes the algorithm implementation a thread which can be terminated anytime. But, we just use the variable *interrupt* assuming it can be set to *true* from a daemon checking for user interruption. Therefore, the variable *interrupt* notifies the user interruptions to the anytime motif discovery algorithm.

4.2.1 Linear Pair Ranking

The LinearPairsRanking function ranks all the possible matches in the time-series T, by using a sliding window which moves from left to right. The function is called just once, when the ranking of the matches in the smallest hierarchy level is performed. Therefore, it has to be executed till completion and it does not present any synchronization point which checks for the user interruption. The *sort* function orders the tuples in P according to the last value in them: the similarity score between two windows pointed by the first two index in the tuples.

Algorithm 3: Linear Pairs Ranking (T, w).

```
Data: $T$, time series of length $n$. $w$, motifs length
Result: P, ranking of all the possible pairs in T
count ← 0;
P ← [];
for $a$ = 0 to len(T) - $w$ + 1 do
    for $b$ = $a$ + $w$ to len(T) - $w$ + 1 do
        P[count] ← {a, b, $L_2$(norm(T[a:a+w]), norm(T[b:b+w])) };
        count ← count + 1 ;
    end
end
return sort(P)
```

4.2.2 Induced pairs ranking

Algorithm 4: InducedPairsRanking(T, ranking, w, k).

```
Data: $T$, time series approximation of length $n$. $w$, motifs length. ranking,
      ranking induced by the previous step.
Result: P, ranking of all the possible pairs in T
count ← 0;
P ← [];
for $a = 0$ to len(ranking) do
    i1 ← ranking[a][0];
    i2 ← ranking[a][1];
    P[count] ← (i1, i2, $L_2$(norm(T[i1:i1+w]), norm(T[i2:i2+w])) );
    if interrupt then
        if $a \geq k$ then
            return sort(P);
        else
            return ranking;
        end
    end
end
for $a$ = 0 to len(T) - $w$ + 1 do
    for $b$ = $a$ + $w$ to len(T) - $w$ + 1 do
        if (a,b) $\notin$ ranking then
            P[count] ← {a, b, $L_2$(norm(T[a:a+w]), norm(T[b:b+w])) };
            count ← count + 1;
        end
        if interrupt then
            return sort(P);
        end
    end
end
return sort(P)
```

The InducedPairsRanking function is composed of two loops. The first computes the matches ranking following the orders induced from the algorithm-previous-step ranking. The second step computes the ranking of all the remaining matches linearly from left to right, and merges this with the one obtained in the first loop. Both the loops, at the conclusion of each iteration, check if the user asked for the algorithm interruption. If that is the case, if the algorithms has been able to check more than k matches, then the new partial ranking is returned. Otherwise the previous step ranking is returned.

4.2.3 Anytime motif discovery

Algorithm 5: AnytimeMotifDiscovery(H, w, k).

```
Data: $H$, time series approximations hierarchy. $w$, motifs length. *ranking*,
      ranking induced by the previous step.
Result: top-k ranking
M ← [];
t ← m;
count ← 0;
complete = false;
while $t > 1$ do
    M[count] ← t;
    t ← t/2;
    count ← count + 1;
end
count ← count - 1;
topk ← LinearPairsRanking(H[count-1], w);
while *!(interrupt) and !(count = 0)* do
    count ← count -1;
    topk ← InducedPairsRanking(H[count], map(topk), w, k);
end
return *topk[0:k]*
```

The AnyTimeMotifDiscovery algorithm accepts as input a time series sub-samplings hierarchy H, a motif length w, a parameter k and returns the top-k motifs within the time series. After having computed the motifs ranking, for the highest level i in the hierarchy, such that $m/2^i > 1$, it starts climbing the hierarchy. At each step of the climb, the level matches-ranking is computed by inducing the matches order obtained at the step before. Since each step, presents a signal approximation which has twice the points of the one at the previous step, the *map* function maps the indexes of the matches to be induced to the higher resolution approximation. It does this by multiply each index by two. The algorithm ends when the first hierarchy level ranking operation has been complete. At any moment the algorithm termination can be invoked, by setting to true the interrupt variable.

4.2.4 Some remarks on the pseudo code

The proposed dissertation proves that it makes sense to use our subsampling hierarchy in order to provide a scalable anytime motif discovery algorithm. In order to keep the explanation simple, we did not care about the algorithm performance optimization, therefore the presented pseudo-code makes several assumptions which do not hold when working with massive datasets. Specifically, it assumes that both the time-series representation and the matches ranking can fit in the main memory. We took such an approach because we wanted to keep clear, simple and compact the presented code.
According to the design of our platform, it would be pretty easy to refactor the presented code to a version able to read the data directly form the disk. We stress that we used the HDF5 data format due to its fast data access and filesystem like data structure, which allows us to access the data as if it are stored in the main memory. De facto, the presented code would remain pretty similar.
As previously outlined, at each algorithm step the total number of analyzed matches is quadratic with the number of points in the series approximation, so each step induces an $O(n^2)$ ranking list to the next one. By using a lower bound over the similarity measures, it is possible to discard all the not-promising matches and so retain on average just O(w + 2klog(w)) matches [14].

4.3 Experimental Analysis

In this section, we outline some experiments we conducted. They prove that after few iterations our algorithm achieves results which are really satisfying. Therefore, the use of the a sub-samplings hierarchy, in an anytime approach, is an effective way of supplying high user interactivity while retaining high the quality of the provided results.

Experiments Datasets					
Dataset	*Source*	*Series length*	*Used points*	*Motifs Length*	*k*
Poem	Rakthant	1 351	1 351	32	7
ECG	Rakthant	1 000	1 000	32	7
Winding	Rakthant	2 500	2 500	128	7
EEG (LFSS)	Mueen-Keogh	182 124	40 000	256	20
EOG	Mueen-Keogh	8 000 000	40 000	256	20
ECG (ECGT)	Mueen-Keogh	60 000	40 000	256	20
Insect (B)	Mueen-Keogh	78 254	40 000	256	20
Random Walk	Mueen-Keogh	40 000	40 000	256	20
Strain sensor	Infrawatch	28 000	28 000	256	20

We implemented the anytime motif discovery algorithm in C. We run all the programs in a 2.0 GHz quad core Mac OSx platform with 4GB of RAM. The datasets we use in the experiments include real time series and synthetic ones. From [19], we use the provided code which generates synthetic random walks datasets. As in [19] and many other works, we use a time series of length 180 124 which is a sampled EEG data, a series sampling insect behavior and a series sampling EOG data. From [20], we use: a sampled poem; an industrial dataset where data from a wire winding process are collected; a heart beat time series (ECG) from PhysioBank ATM. Finally, we use the data coming the Infrawatch project [13], where a total of 145 sensors were embedded and attached to the Holand Brug by Strukton. The table above summarizes the experiments datasets, configurations and parameters.

4.3.1 Precision and recall

For each one of the analyzed datasets, we recorded the top-k patterns obtained by using the exact motif discovery algorithm over the original data, and used these as the ground truth for our study. Then, by using the information retrieval metrics *precision* and *recall*, we assessed the quality of the top-k rankings returned at each step of the algorithm execution. These measures were computed by using the number of true positive (TP), false positive (FP) and false negatives (FN): the TP are the number of motifs present in the results obtained by using the original signal and its approximation; the FP are the motifs which are incorrectly in the approximated version; and the FN are the motifs which are in the ground truth but are not in the approximated ranking. Given this, precision and recall are defined as:

$$precision = \frac{TP}{TP + FN} \qquad recall = \frac{TP}{TP + FN} \tag{4.3}$$

We recall that each motif is composed of two windows, respectively pointed by the indexes a and b. In order to be compared with the ground truth, the indexes of any motif computed over a series approximation need to be mapped to the original series scale, since the last presents more points. So, given an approximated motif in the series-segmentation retrieved from the l_i hierarchy level, its indexes (a, b) are mapped to the original space by multiplying them by two at the power of i. Recalling that the level $l_i = 0$ contains the original sampled series.

$$map(a, b) = (a \cdot 2^i, b \cdot 2^i) \tag{4.4}$$

4.3.2 Matching rule

Given one motif g_i in the ground truth and one motif t_i in any series approximation, *precision* and *recall* are highly affected by the rule used to asses if g_i and t_i can be considered the same motif. The obvious rule would check the equality of the motifs indexes, after having mapped the indexes of t_i to the original sampling scale; i.e. the motifs are equal if $(a_{g_i} = a_{t_i})$ and $(b_{g_i} = b_{t_i})$. Clearly such a rule is too strict, since each point in the series approximation is the aggregation of a window in the original sampling, and so any motif starting from any arbitrary point in the middle of the aggregation window, would never match any approximated motif. A trivial example are all the ground truth motifs having an odd index, these will never match indexes mapped from an approximated motif, since it will be even for sure. Therefore, we adopt a constant α which introduces some flexibility in the rule. It defines the number of units, expressed as percentage of the motif length, that the two indexes can defer each other to still be consider equal. So, given a motif g_i in the ground truth and a motif t_i in the series approximation, they are considered the same motif if: $\text{mod } (a_{g_i} - a_{t_i}) < \alpha \cdot w$ and $\text{mod } (b_{g_i} - b_{t_i}) < \alpha \cdot w$.
Given our formalization of the top-k motif discovery problem, we do not count the number of times each motif appears. From this, in the top-k ranking achieved with our approach there could be the same motif appearing many times. That happens when the motif appears more than two times in the series. Hence, a motif in the approximation could be the same as a motif in the ground truth but do not share the indexes. In order to partially consider this case and to introduce more flexibility, we adjust the matching rule by changing the logic *and* in the rule with a logic *or*. The final matching rule follows.

Precision and recall motifs matching rule. Given a motif g_i of length w in the ground truth and a motif t_i in any series approximation, they are considered the same motif if:

$$\text{mod } (a_{g_i} - a_{t_i}) \leq \alpha \cdot w \quad \vee \quad \text{mod } (b_{g_i} - b_{t_i}) \leq \alpha \cdot w \tag{4.5}$$

where a and b point respectively at the two windows in the motif and the indexes (a, b) in t_i are mapped to the original series scale.

4.3.3 Results

Precision, *recall* and execution times were recorded for each one of the experiments. Figure 4.2 compares the precision of the results, obtained at each algorithm step, with the execution time needed to achieve them. The outcome is really promising: for any dataset the algorithm achieves result with precision equal or greater than 0.75, in a time which is at least 10 times faster the one needed by the exact match discovery to run to completion. We consider it a really effective solution in all the cases where the user asks for high interactivity.

Electrocephalography sampling (EEG, Figure 4.3) and Elegtrooculogram sampling (EOG, figure 4.4) are two of the most-used datasets in the motif discovery literature, since it is well-known the existence of well-defined patterns in them. So unsurprisingly, the results achieved with such time series are even better than the ones presented before. As visible, precision and recall greater than 0.9 are obtained in up to 64 times less the time required by the exact match discovery to run to completion.
Figure 4.8 depicts the motifs discovered in the Infrawatch sensor 101, where data coming from a strain sensor, embedded into the Hollandse bridge by Strukton, are collected. The algorithm has been left running till completion. The charts outlines the results achieved at the last five executed steps. These are surprising, precision and recall are better at the level using an approximation compressed by 2^5 times than the ones obtained at the levels where the signal has been reduced by 2^4 and 2^3. That is the only dataset which presents such a behavior.
Finally, the results obtained with the synthetic random walks dataset (4.6) are more than surprising. But, as previously mentioned, this is a dataset generated ad-hoc to test the behavior of motif discovery algorithms. So, the outcomes achieved from such a dataset can not be considered a valid representation of real datasets.

The anytime motif algorithm we propose is an effective way to keep high interactivity in the data mining tool. It provides a solution which does not exclude exact results but which is able to return a solution to the problem at any moment it is terminated.

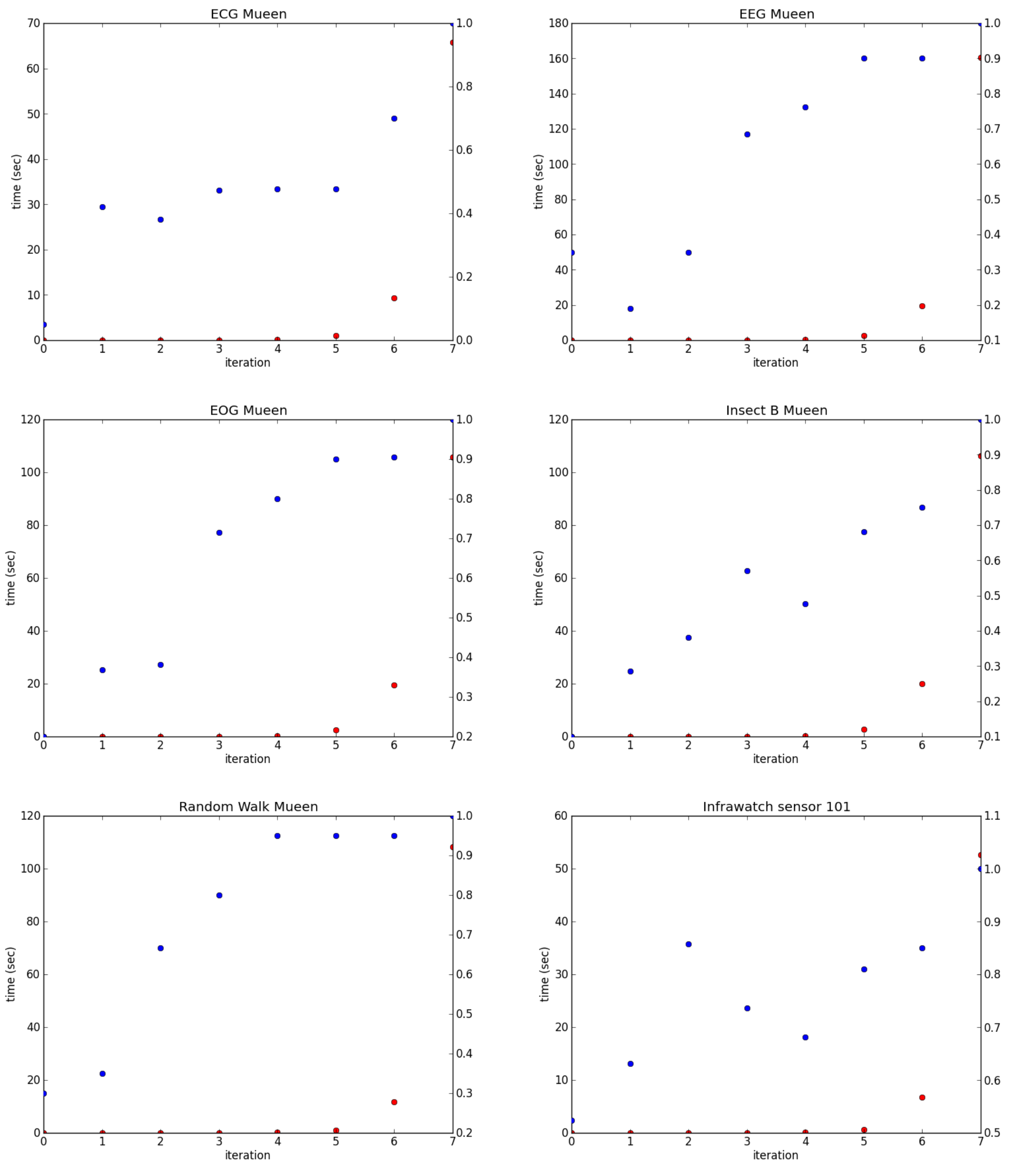


Figure 4.2: **Anytime motif discovery precision vs execution time.** Precision (blue) vs execution time (red) for EEG, EOG, Randow Walk and Insect behavior datasets. The parameter α used in the *precision* and *recall* computation is set to 0.4. The results are really promising, for each dataset the algorithm obtains a precision equal or greater then 0.8 in at least 10 times less the time required by the exact motif discovery algorithm.

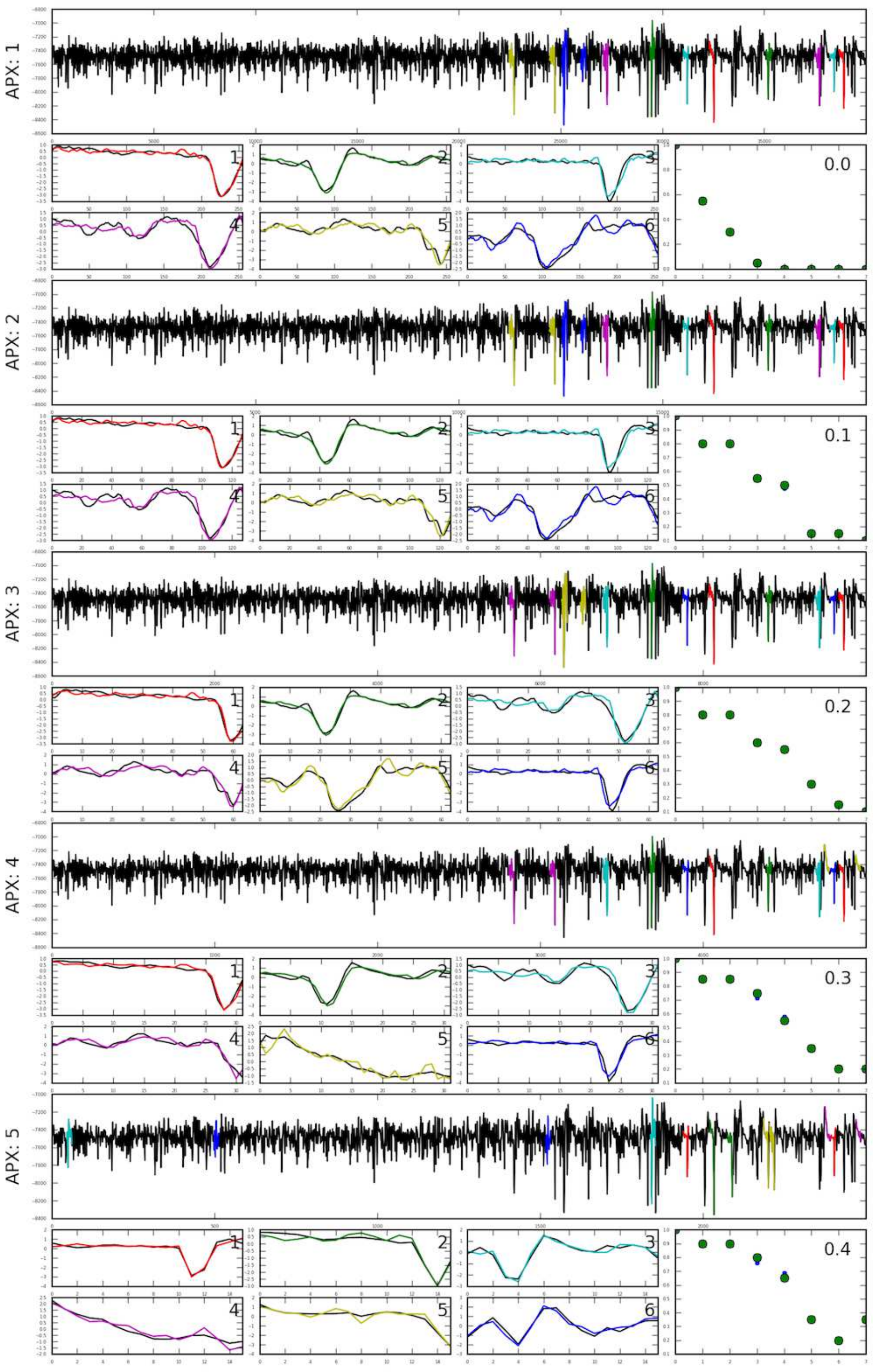


Figure 4.3: **Mueen EEG.**Motif discovered at different resolutions in a dataset where an electroencephalography sampling is collected.

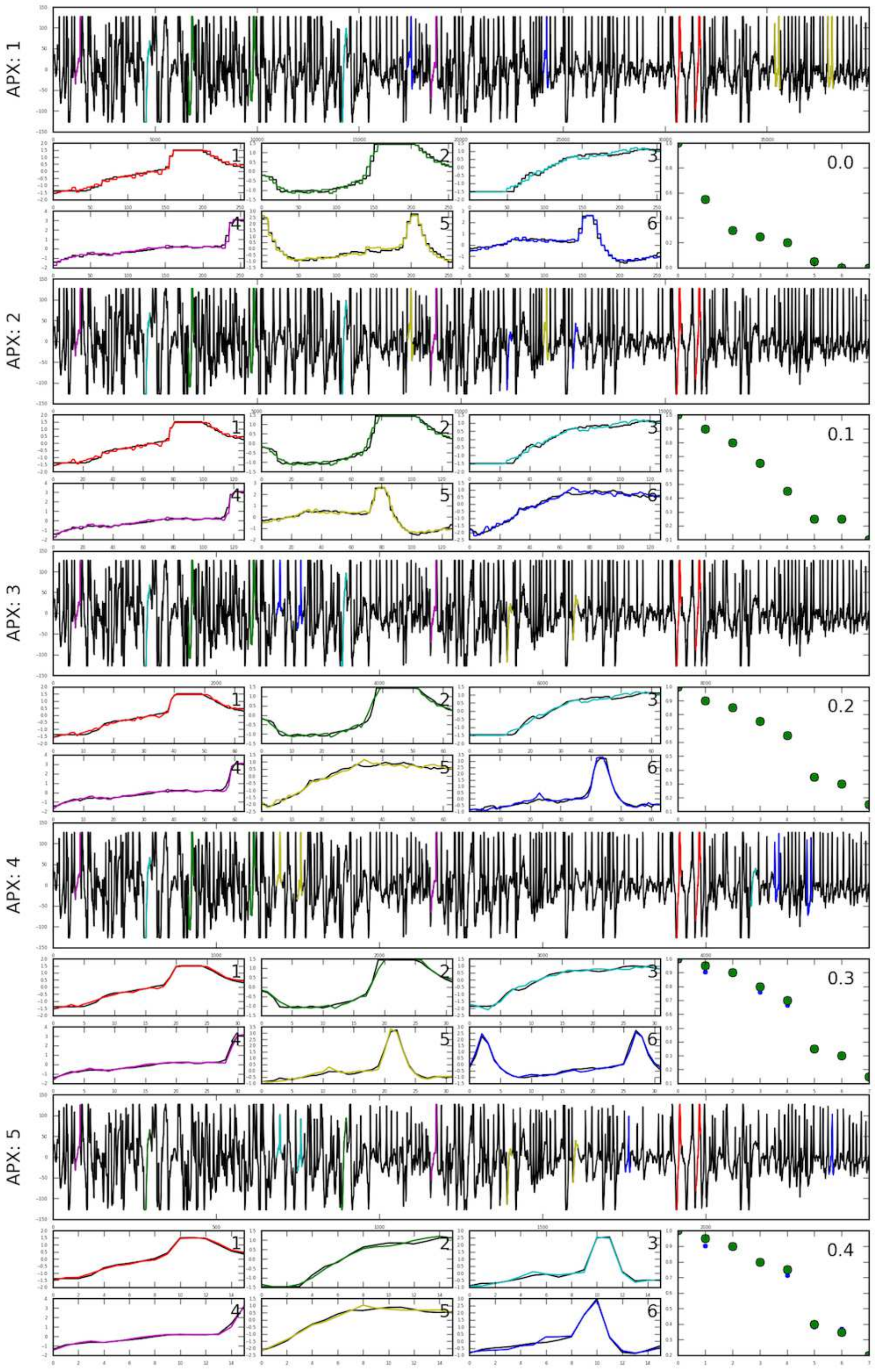


Figure 4.4: **Mueen EOG.**Motif discovered at different resolutions in a dataset where an electrooculogram sampling is collected.

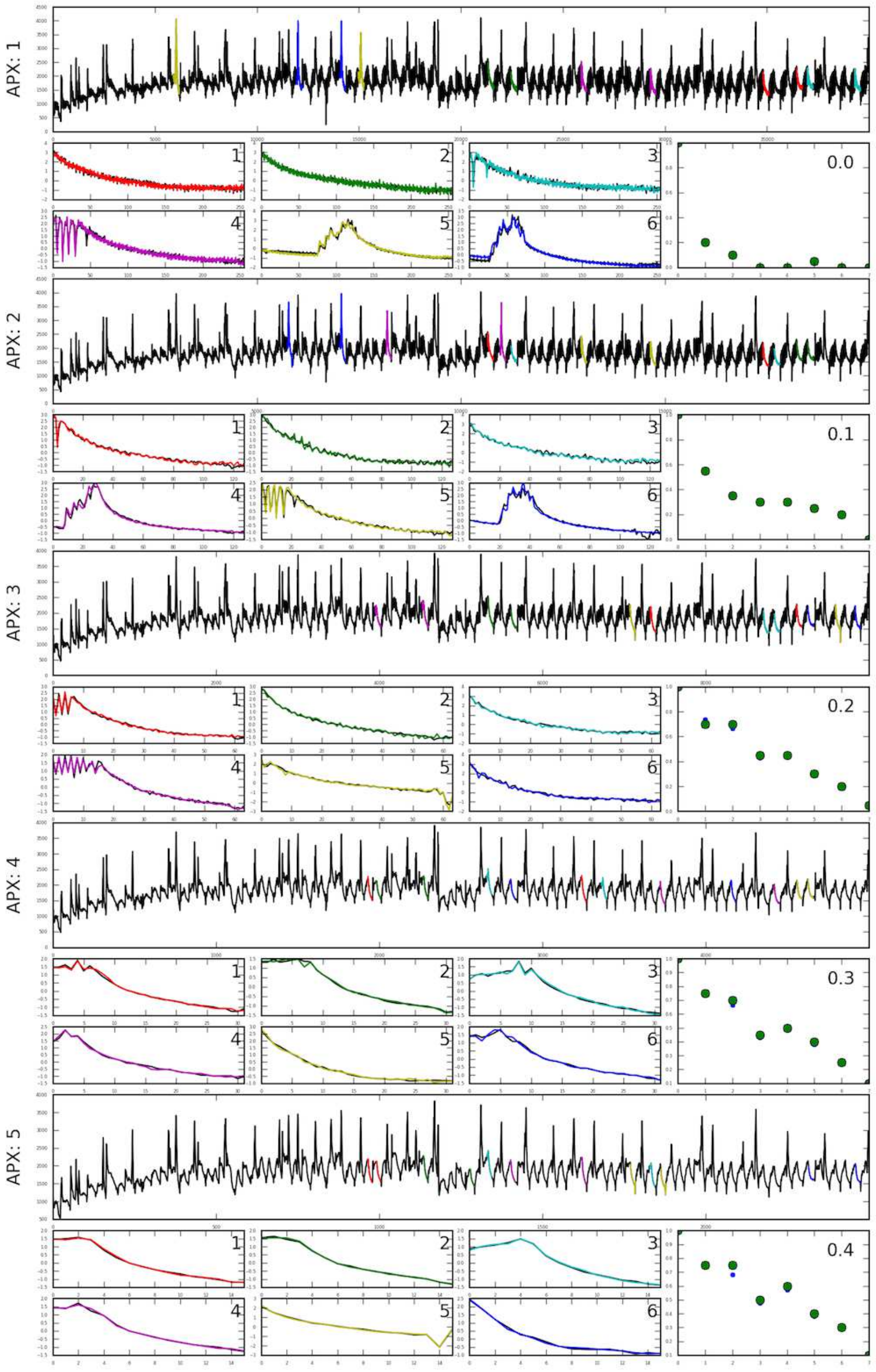


Figure 4.5: **Mueen Insect B.** Motif discovered at different resolutions in a series sampling insect behaviors.

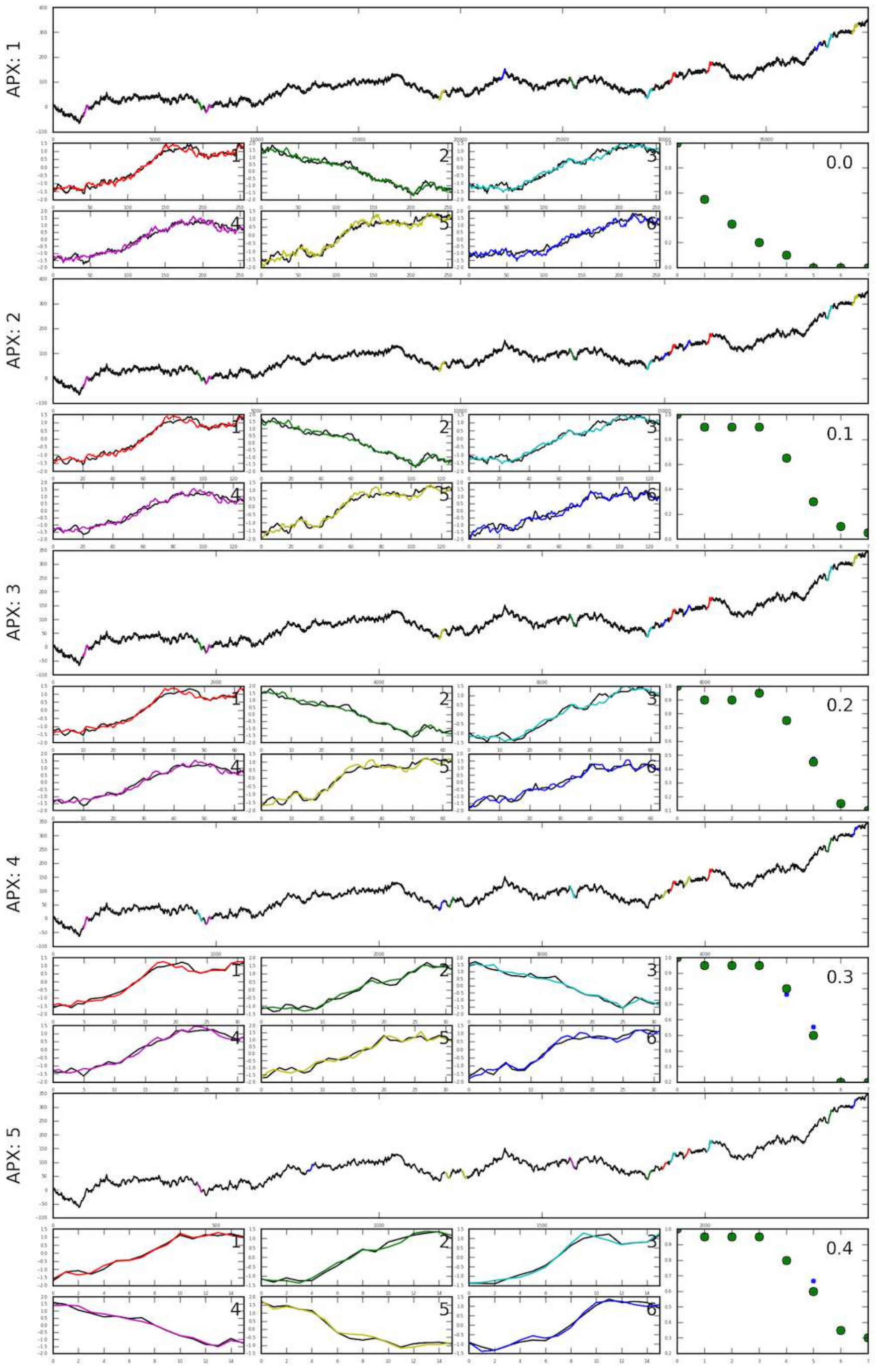


Figure 4.6: **Mueen random walks.** Motif discovered at different resolutions in an synthetic dataset simulating random walks.

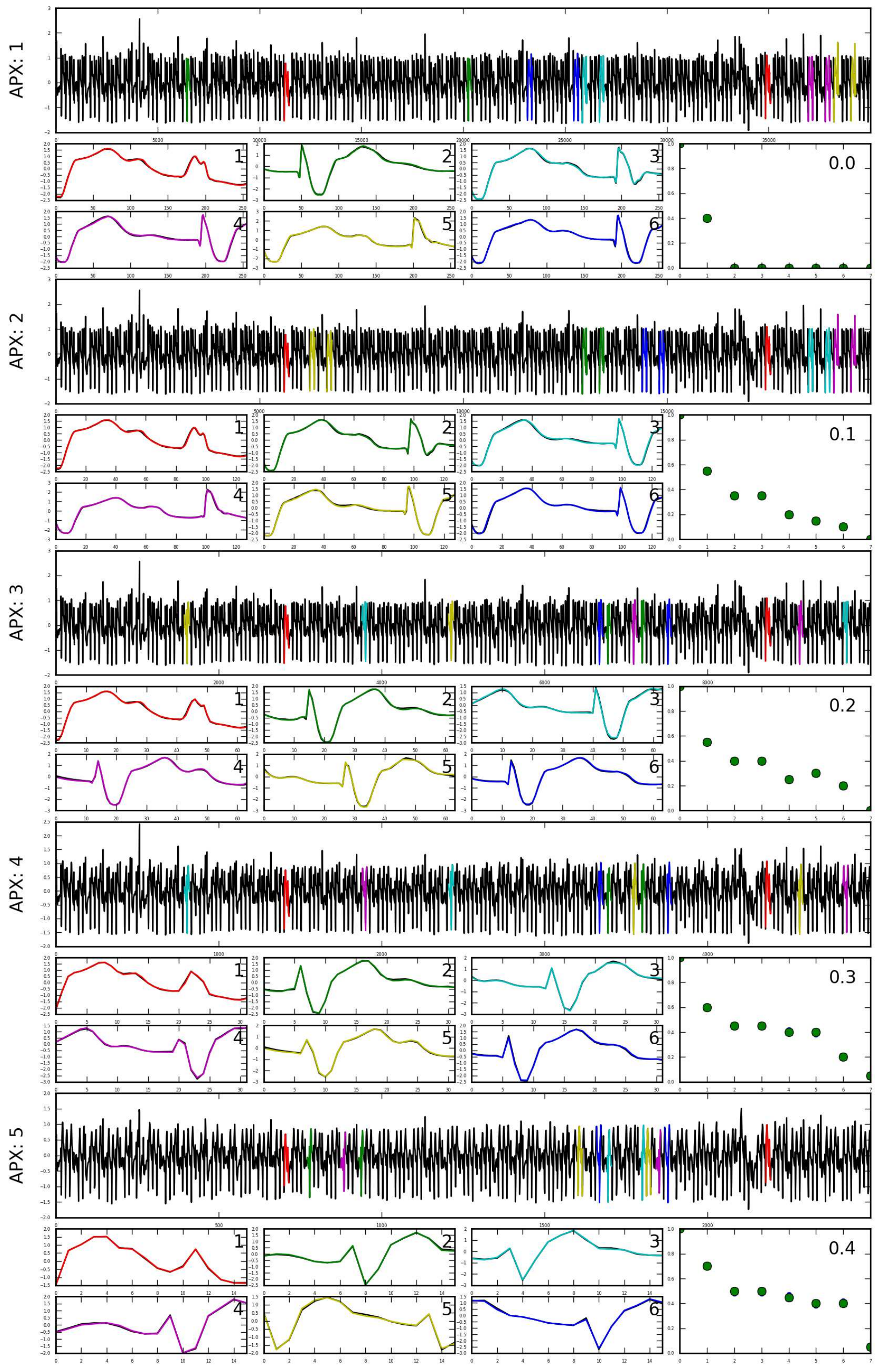


Figure 4.7: **Mueen ECG.** Motif discovered at different resolutions in a series sampling ECG data.

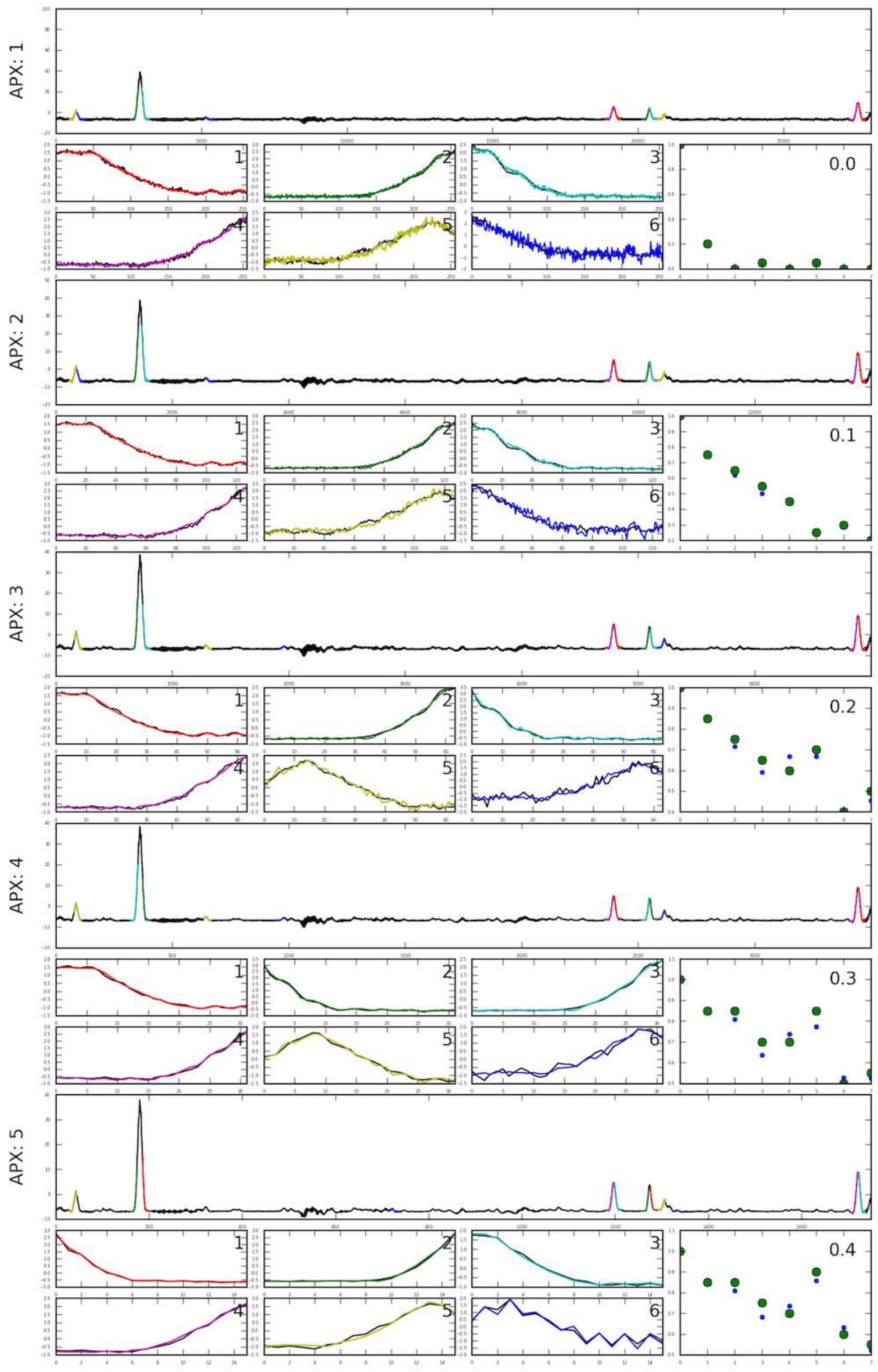


Figure 4.8: **Infrawatch strain sensor (101).** Motifs discovered at different series resolutions in an industrial dataset, where data coming from a strain sensor embedded into a bridge are collected.

Chapter 5

Conclusion and future work

In this work we proposed VizTool, a time series visualization tool which allows for interactive and explorative data analysis. By adopting a subsampling hierarchy, we implemented a scalable and efficient platform, which copes with any dataset size and offers efficient resources usage, adapting the displayed data-resolution according to the client capabilities.
What we propose adopts a web client-server solution where the data is stored in a unique, safe and centralized server, being therefore portable and ubiquitous. Several state of the art techniques, able to boost the overall interactivity, have been applied and integrated in the tool; data pre-fetching and data-buffering are some of these. We increased the tool effectiveness by integrating several data management and annotation components, such as data import-export and sharing functionalities.

Being aware of the importance the signal's perceptual features have in the process of forming an intuition about the system behavior, in the second part of the work we studied and experimented several segmentation techniques. From such an analysis, we observed how reduction techniques that focus their approximation on preserving the peaks in the signal, do not achieve satisfactory representation at high compression ratios. This is due to the approximation of the original series missing the fundamental frequencies. Conversely, as the compression ratio increases, non-data-adaptive algorithms progressively smooth the series and lose the perceptually important points in it. Due to this, we suggested a method to select a data-adaptive hybrid approximation, obtained by composing diverse techniques.

Finally, the last part of the work presents an anytime motif discovery algorithm, able to return results whenever it is interupted. It is an algorithm that progressively refines and improves the achieved results. By using information retrieval metrics such as precision and recall, we tested the algorithm over several datasets highly used in the literature. The experiments prove the effectiveness that the proposed method has in increasing the tool's interactivity, while retaining a high quality of the results.

VizTool provides several functionalities, but a lot of work can still be done. Here we outline some improvements and features which could be implemented. The current software version builds the subsampling-hierarchy by using the Piecewise Aggregate Approximation technique; the proposed hybrid method should be used as well. To do so, it would be necessary to develop a module which automatically optimizes the algorithm parameters over a small subset (training-set) of the dataset which needs to be reduced.

VizTool allows the exploration of already collected data, but it lacks a set of streaming-data APIs which provide realtime-collected data exploration. The adoption of such a solution requires real-time data subsampling, therefore several adjustments to our hybrid algorithm are needed since it is still not an online-algorithm. This solution would allow the triggering of alarms over real-time monitored data, notifying when user-set thresholds are exceeded. These can be inserted as future work.

Several data mining and data management components have been integrated in VizTool. However a component allowing the composition of series and mathematical operators is still missing. The current version implies that the massive data can fit in the storage supplied by one single node server: since this is not always the case, the integration of a map-reduce layer would be a useful feature.

Our work proposed a data-adaptive algorithm able to preserve the perceptual features in the series, as well. An effective metric, which allows us to asses the capability of each algorithm to preserve these important points needs to be proposed. Our assessment is therefore still based on perceptual evaluation and on the Euclidean similarity measures.

Currently, the hybrid algorithm splits the aggregation windows into two different groups, which are respectively reduced with Piecewise Aggregate Approximation and Ramer-Douglaus-Peucker algorithms. A different number of points per window is used by each one of the two segmentation techniques. As we did with the Weighted Least Square Regression, it would be useful to split the windows over more than two bins, and associate each bin with a number of points to be used. Basically, we would like to increase the resolution of the model selection.

Our tool is very effective to explore and analyze massive time series, since it can remain highly interactive by adopting novel and state of the art techniques, such as prefetching, buffering and anytime data mining tasks.

Bibliography

[1] Alberto Baggio, Ugo Vespier, and Arno Knobbe. Automated selection of data-adaptive approximations for large time-series visualization. Benelearn, 2013.

[2] Jurgen Bernard, Tobias Ruppert, Oliver Goroll, Thorsten May, and Jorn Kohlhammer. Visual-interactive preprocessing of time series data. In Andreas Kerren and Stefan Seipel, editors, *SIGRAD*, volume 81 of *Linkoping Electronic Conference Proceedings*, pages 39–48. Linkoping University Electronic Press, 2012.

[3] Nuno Castro and Paulo Azevedo. Multiresolution motif discovery in time series.

[4] Nuno Castro and Paulo J. Azevedo. Time series motifs statistical significance.

[5] Sye-Min Chan, Ling Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *Visual Analytics Science and Technology, 2008. VAST '08. IEEE Symposium on*, pages 59–66, 2008.

[6] Punit R. Doshi, Elke A. Rundensteiner, and Matthew O. Ward. Prefetching for visual data exploration.

[7] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, December 2012.

[8] Eugene Fink and Harith Suman Gandhi. Compression of time series by extracting major extrema. *Journal of Experimental and Theoretical Artificial Intelligence*, 23(2):255–270, 2011.

[9] M. C. Hao, H. Janetzko, S. Mittelstädt, W. Hill, U. Dayal, D. A. Keim, M. Marwah, and R. K. Sharma. A visual analytics approach for peak-preserving prediction of large seasonal time series. In *Proceedings of the 13th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'11, pages 691–700, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[10] Ming Hao, Umeshwar Dayal, Daniel Keim, and Tobias Schreck. Multi-resolution techniques for visual exploration of large time-series data. In *Proceedings of the*

9th Joint Eurographics / IEEE VGTC Conference on Visualization, EUROVIS'07, pages 27–34, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[11] W. Javed and N. Elmqvist. Stack zooming for multi-focus interaction in time-series data visualization. In *Pacific Visualization Symposium (PacificVis), 2010 IEEE*, pages 33–40, 2010.

[12] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 206–215, New York, NY, USA, 2004. ACM.

[13] Arno Knobbe, Hendrik Blockeel, Arne Koopman, Toon Calders, Bas Obladen, Carlos Bosma, Hessel Galenkamp, Eddy Koenders, and Joost Kok. Infrawatch: Data management of large systems for monitoring infrastructural performance. 6065:91–102, 2010.

[14] Hoang Thanh Lam, Toon Calders, and Ninh Pham. Online discovery of top-k similar motifs in time series data. In *SDM*, pages 1004–1015. SIAM / Omnipress, 2011.

[15] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, pages 2–11, New York, NY, USA, 2003. ACM.

[16] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Jeffrey P. Lankford, and Daonna M. Nystrom. Viztree: a tool for visually mining and monitoring massive time series databases. In *In Proceedings of International Conference on Very Large Data Bases*, pages 1269–1272, 2004.

[17] Peter McLachlan, Tamara Munzner, Eleftherios Koutsofios, and Stephen North. Liverac: Interactive visual exploration of system management time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1483–1492, New York, NY, USA, 2008. ACM.

[18] Abdullah Mueen. Enumeration of time series motifs of all lengths.

[19] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, and Sydney Cash. Exact discovery of time series motifs. In *SDM*, 2009.

[20] Thanawin Rakthanmanon, Eamonn J. Keogh, Stefano Lonardi, and Scott Evans. Time series epenthesis: Clustering time series streams requires ignoring some data. In Diane J. Cook, Jian Pei, Wei Wang 0010, Osmar R. Zaiane, and Xindong Wu, editors, *ICDM*, pages 547–556. IEEE, 2011.

[21] Galen Reeves, Jie Liu, Suman Nath, and Feng Zhao. Managing massive time series streams with multi-scale compressed trickles. *Proc. VLDB Endow.*, 2(1):97–108, August 2009.

[22] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-weighted linear stacking. *CoRR*, abs/0911.0460, 2009.

[23] Joaquin Vanschoren, Hendrik Blockeel, Bernhard Pfahringer, and Geoffrey Holmes. Experiment databases. *Machine Learning*, 87(2):127–158, 2012.

[24] Dragomir Yankov, Eamonn Keogh, Jose Medina, Bill Chiu, and Victor Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 844–853, New York, NY, USA, 2007. ACM.

[25] Jian Zhao, Fanny Chevalier, and Ravin Balakrishnan. Kronominer: Using multi-foci navigation for the visual exploration of time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1737–1746, New York, NY, USA, 2011. ACM.

[26] Jian Zhao, Fanny Chevalier, Emmanuel Pietriga, and Ravin Balakrishnan. Exploratory analysis of time-series with chronolenses.