Visualizing and Discovering Non-Trivial Patterns In Large Time Series Databases

Jessica Lin
Eamonn Keogh[1]
Stefano Lonardi

Computer Science & Engineering Department
University of California, Riverside
Riverside, CA 92521

Correspondence:
Eamonn Keogh
Computer Science & Engineering Department
University of California, Riverside
Tel: +1 951 827 2032
Fax: +1 951 827 4643
E-mail: eamonn@cs.ucr.edu

Short running title: Time Series Visualization

[1] Dr. Eamonn Keogh is supported by NSF Career Award IIS-0237918

Visualizing and Discovering Non-Trivial Patterns In Large Time Series Databases

Jessica Lin
Eamonn Keogh
Stefano Lonardi

Computer Science & Engineering Department
University of California, Riverside
Riverside, CA 92521

Correspondence:
Eamonn Keogh
Computer Science & Engineering Department
University of California, Riverside
Tel: +1 951 827 2032
Fax: +1 951 827 4643
E-mail: eamonn@cs.ucr.edu

Abstract

Data visualization techniques are very important for data analysis, since the human eye has been frequently advocated as the ultimate data-mining tool. However, there has been surprisingly little work on visualizing massive time series datasets. To this end, we developed VizTree, a time series pattern discovery and visualization system based on augmenting suffix trees. VizTree visually summarizes both the global and local structures of time series data at the same time. In addition, it provides novel interactive solutions to many pattern discovery problems, including the discovery of frequently occurring patterns (motif discovery), surprising patterns (anomaly detection), and query by content. VizTree works by transforming the time series into a symbolic representation, and encoding the data in a modified suffix tree in which the frequency and other properties of patterns are mapped onto colors and other visual properties. We demonstrate the utility of our system by comparing it with state-of-the-art batch algorithms on several real and synthetic datasets. Based on the tree structure, we further device a coefficient which measures the dissimilarity between any two time series. This coefficient is shown to be competitive with the well-known Euclidean distance.

Keywords: Time Series, Visualization, Motif Discovery, Anomaly Detection, Pattern Discovery

Introduction

The U.S. Department of Defense (DoD) collaborates with numerous contractors who help provide the skills and services required for U.S. defense. Among these contractors, The Aerospace Corp (TAC) has a unique and crucial responsibility. The Aerospace Corp has to provide engineering assessments for the engineering discipline specialists who make the critical *go/no-go* decision moments before the launch of every unmanned space vehicle launched by the DoD (Note that Engineering Discipline Specialist is the correct term, without disrespect we will use the terser term technician throughout this paper). The cost of a false positive, allowing a launch in spite of a fault, or a false negative, stopping a potentially successful launch, can be measured in the tens of millions of dollars, not including the cost in morale and other more intangible detriments to the U.S. defense program.

The launch control rooms at the DoD are similar to the familiar Hollywood movie recreations [28]. There are several rows of work cells, each with a computer display and a headset. Each work cell is devoted to one technician; for example, propulsion, guidance, electrical, etc. Each display presents some common data (say vehicle location and orientation), as well as data specific to that discipline.

The technician making the *go/no-go* decision has access to data from previous launches and must constantly monitor streaming telemetry from the current mission.

Currently, the technicians use electronic strip charts similar to those used to record earthquake shock on paper rolls. However,

while these charts illustrate the recent history of each sensor, they do not provide any useful higher-level information that might be valuable to the technician.

To reduce the possibility of wrong *go/no-go* decisions, TAC is continually investing in research. There are two major directions of research in this area.

- Producing better techniques to mine the archival launch data from previous missions. Finding rules, patterns, and regularities from past data can help us "*know what to expect*" for future missions, and allow more accurate and targeted monitoring, contingency planning, etc [28].
- Producing better techniques to visualize the streaming telemetry data in the hours before launch. This is particularly challenging because one may have to monitor as many as dozens of rapidly changing sensors [28].

Although these two tasks are quite distinct, and are usually tackled separately, the contribution of this work is to introduce a single framework that can address both. Having a single tool for both tasks allows knowledge gleaned in the *mining* stage to be represented in the same visual language of the *monitoring* stage, thus allowing a more natural and intuitive transfer of knowledge.

More concretely, we propose VizTree, a time series pattern discovery and visualization system based on augmenting suffix trees. VizTree simultaneously visually summarizes both the global and local structures of time series data. In addition, it provides novel interactive solutions to many pattern discovery problems, including the discovery of frequently occurring patterns (motif discovery) [8, 31, 41], surprising patterns (anomaly detection) [10, 26, 39], and query by content [12, 17, 23, 37]. The user interactive paradigm allows users to visually explore the time series, and perform real-time hypotheses testing [1, 21].

Perhaps the best-known and most referenced researcher in information visualization and user interfaces is Dr. Ben Shneiderman of the University of Maryland. He has championed a set of principles for designing usable and informative scientific data visualization systems [40]. The shortest summary of these principles is known as the Visual Information Seeking Mantra: "*Overview, zoom & filter, details-on-demand*". As we will show in this paper, our work fits neatly into these principles. We give an *overview* of the global structure of an arbitrarily long time series in constant space, while we allow the user to *zoom* in on particular local structures and patterns, and provide *details on demand* for patterns and regularities that the user has tentatively identified.

While there are several systems for visualizing time series in the literature, our approach is unique in several respects. First, almost all other approaches assume highly periodic time series [43, 45], whereas ours makes no such assumption. Other methods typically require space (both memory space, and pixel space) that grows at least linearly with the length of the time series, making them untenable for mining massive datasets. Finally, our approach allows us to visualize a much richer sets of features, including global summaries of the differences between two time series, locally repeated patterns, anomalies, etc.

While the evaluation of visualization systems is often subjective, we will evaluate our system with objective experiments by comparing our system with state-of-the-art batch algorithms on several real and synthetic datasets.

Background and related work

We begin this section by briefly reviewing the most important time series data mining tasks. We will then consider current visualization techniques and explain why they are unsuited to the problem at hand.

Time series data mining tasks

For lack of space, this brief introduction to the important time series data mining tasks is necessarily subjective and somewhat domain driven. Nevertheless, these three tasks cover the majority of time series data mining research [7, 8, 10, 12, 17, 20, 24, 26, 31, 33, 34, 41].

Subsequence matching

Sequence matching is perhaps the most widely studied area of time series data mining [12, 17]. The task has long been divided into two categories: whole matching and subsequence matching [12, 23].

- **Whole Matching**: a query time series is matched against a database of individual time series to identify the ones similar to the query.
- **Subsequence Matching**: a short query subsequence time series is matched against longer time series by sliding it along the longer sequence, looking for the best matching location.

While there are literally hundreds of methods proposed for whole sequence matching (see, e.g., [24] and references therein), in practice, its application is limited to cases where some information about the data is known *a priori*.

Subsequence matching can be generalized to whole matching by dividing sequences into non-overlapping sections. For example, we may wish to take a long electrocardiogram and extract the individual heartbeats. This informal idea has been used by many researchers and is also an optional feature of VizTree. We will therefore formally name this transformation *chunking,* and define it below.

Definition 1 *Chunking*: the process where a time series is broken into individual time series by either a specific period or, more arbitrarily, by its shape.

The former usually applies to periodic data, for example consider power usage data provided by a Dutch research facility (this dataset is used as a running example in this work, see Figure 3 and Figure 17): the data can be chunked by days, weeks, etc. The latter applies to data having regular structure or repetitive shape, but not necessarily having the same length for each occurrence. Electrocardiogram data are such an example, and they can be separated into individual heartbeats.

There is increasing awareness that for many data mining and information retrieval tasks, very fast approximate search is preferable to slower exact search [6]. This is particularly true for exploratory purposes and hypotheses testing. Consider the stock market data. While it makes sense to look for approximate patterns, for example, *"a pattern that rapidly decreases after a long plateau*," it seems pedantic to insist on *exact* matches. As we will demonstrate later, our application allows rapid approximate subsequence matching.

Anomaly detection

In time series data mining and monitoring, the problem of detecting anomalous/surprising/novel patterns has attracted much attention [10, 33, 39]. In contrast to subsequence matching,

anomaly detection is identification of previously *unknown* patterns. The problem is particularly difficult because what constitutes an anomaly can greatly differ depending on the task at hand. In a general sense, an anomalous behavior is one that deviates from "normal" behavior. While there have been numerous definitions given for anomalous or surprising behaviors, the one given by Keogh et. al. [26] is unique in that it requires no explicit formulation of what is anomalous. Instead, they simply defined an anomalous pattern as one "*whose frequency of occurrences differs substantially from that expected, given previously seen data*". Their definition was implemented in an algorithm (called "Tarzan") that was singled out by NASA as an algorithm that has "*great promise in the long term*" [19]. As it will become clearer later, a subset of the system that we propose here includes what may be considered a visual encoding of Tarzan.

Time Series Motif Discovery

In bioinformatics, it is well documented that overrepresented DNA sequences often have biological significance [3, 11, 38]. Other applications that rely heavily on overrepresented (and underrepresented) pattern discovery include intrusion detection, fraud detection, web usage prediction, financial analysis, etc.

A substantial body of literature has been devoted to techniques to discover such overrepresented patterns in time series; however, each work considered a different definition of *pattern* [4, 35]. In previous work, we unified and formalized the problem by defining the concept of "time series motif" [31]. Time series motifs are close analogues of their discrete cousins, although the definitions must be augmented to prevent certain degenerating solutions. This definition is gaining acceptance, and now being used in animation [5], mining human motion data [41], and several other applications. The naïve algorithm to discover motifs is quadratic in the length of the time series. In [31], we demonstrated a simple technique to mitigate the quadratic complexity by a large constant factor; nevertheless this time complexity is clearly untenable for most real datasets. As we shall demonstrate, VizTree allows users to visually discover approximate motifs in real time.

Visualizing Time Series

Time series is perhaps the most common data type encountered in data mining, touching as it does, almost every aspect of human life, including medicine (ECG, EEG data), finance (stock market data, credit card usage data), aerospace (launch telemetry, satellite sensor data), entertainment (music, movies) [5], etc. Because time series datasets are often massive (in gigabytes or even terabytes), time- and space-complexity is of paramount importance.

Surprisingly, although the human eye is often advocated as the ultimate data-mining tool [21, 40, 42], there has been relatively little work on visualizing massive time series datasets. Below, we will briefly review the three most referenced approaches in the literature and explain why they are not suited to the task at hand.

TimeSearcher

TimeSearcher [15] is a time series exploratory and visualization tool that allows users to retrieve time series by creating queries. This is achieved by use of "TimeBoxes", which are rectangular query locators that specify the region(s) in which the users are interested within any given time series. In Figure 1, three TimeBoxes have been drawn to specify time series that start low, increase, then fall once more. The authors further extended TimeSearcher to provide additional expressitivity which include support to queries with variability in the time interval, and angular queries, which search for ranges of differentials rather than absolute values [16].

The main advantage of this tool is its flexibility. In particular, unlike conventional query-by-content similarity search algorithms, TimeSearcher allows users to specify different regions of interest from a query time series, rather than feeding the entire query for matching. This is useful when users are interested in finding time series that exhibit similar behavior as the query time series in only specific regions.

While TimeSearcher and VizTree proposed here both serve as visualization and exploratory tools for time series, their functionalities are fundamentally different. For example, TimeSearcher is a query-by-example tool for multiple time series data. Even with its flexibility, users still need to specify the query regions in order to find similar patterns. In other words, some knowledge about the datasets may be needed in advance and users need to have a general idea of what is interesting. On the other hand, VizTree serves as a true pattern discovery tool for a long time series that tentatively identifies and isolates interesting patterns and invites further inspection by the technician.

The functionality of TimeSearcher for similarity search is implicit in the design of our system: similar patterns are automatically grouped together. Furthermore, TimeSearcher suffers from its limited scalability, which restricts its utility to smaller datasets, and is impractical for the task at hand.

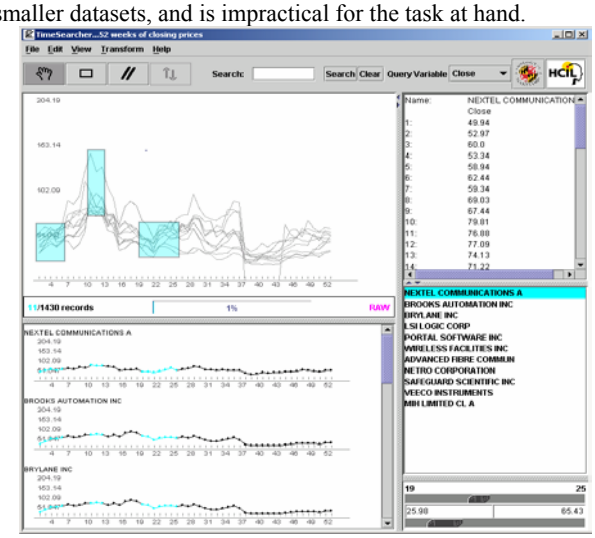

Figure 1: The TimeSearcher visual query interface. A user can filter away sequences that are not interesting by insisting that all sequences have at least one data point within the query boxes.

Cluster and calendar-based visualization

Another time series visualization system is cluster and calendar-based, developed by [43]. The time series data are chunked into sequences of day patterns, and these day patterns are in turn clustered using a bottom-up clustering algorithm. This visualization system displays patterns represented by cluster averages, as well as a calendar with each day color-coded by the cluster that it belongs to. Figure 2 shows just one view of this visualization scheme.

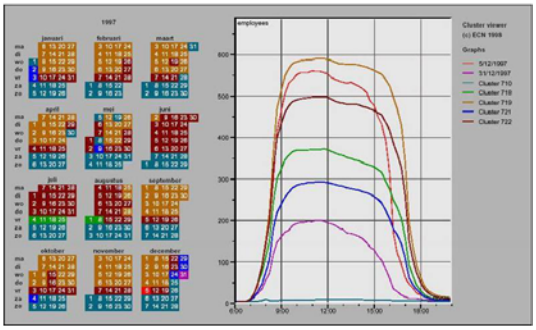

Figure 2: The cluster and calendar-based visualization on employee working hours data. It shows six clusters, representing different working-day patterns.

While the calendar-based approach provides a good overview of the data, its application is limited to calendar-based data, that is to say, data which has some regularity imposed on it by social or financial dependence on the calendar. This approach is of little utility for data without obvious daily/weekly patterns and/or *a priori* knowledge about such patterns. In short, this system works well to find patterns within a specific, known time scale, while our system aims to discover previously unknown patterns with little or no knowledge about the data.

Spiral

Weber et. al developed a tool that visualizes time series on spirals [45]. Each periodic section of time series is mapped onto one "ring" and attributes such as color and line thickness are used to characterize the data values. The main use of this approach is the identification of periodic structures in the data. However, the utility of this tool is limited for time series that do not exhibit periodic behaviors, or when the period is unknown.

We re-implemented the spiral approach and ran it on the power consumption dataset. A screenshot of the resulting spiral is shown in Figure 3

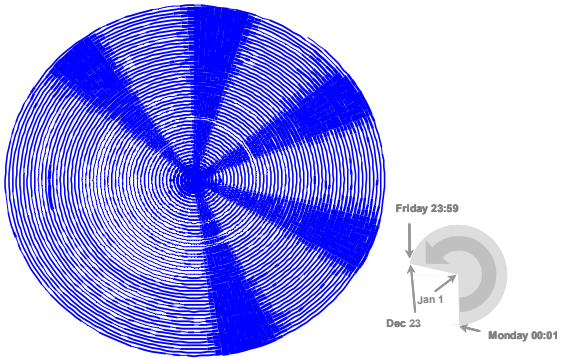

Figure 3: The Spiral visualization approach of Weber et al. applied to the power usage dataset.

Note that one can clearly visualize the normal "9-to-5" working week pattern. In addition, one can see several other interesting events. For example, while it is apparent that no one works during weekends in general, on one Saturday in late summer, there was a power demand suggestive of a full days shift. Surprisingly, this idea for visualizing time series predates computers, with elegant hand drawn examples dating back to at least the 1880's [13, 42].

While the Spiral approach is elegant, it does not meet our requirements for several reasons. As mentioned, it works well only for periodic data (based on the original authors' claims and our own experiments). More importantly, it requires pixel space linear in the length of the time series; this is simply untenable for our purposes.

To sum up, the time series visualization tools described above provide various functionalities, and they were designed under different considerations, with separate objectives in mind. Therefore, there is no easy way to fairly compare and measure their performances quantitatively. However, we can summarize their shortcomings in terms of functionality, conditions on the input datasets, and scalability:

- Functionality: TimeSearcher offers query-by-content capability, but it does not provide any information on the structures of the data itself. Calendar-based approach finds clusters of subsequences, while the spiral approach displays periodic structures in data. The functionalities offered are limited.
- Conditions: With TimeSearcher, users need to know what they are looking for. The calendar-based approach requires calendar data. The spiral approach only works for periodic data, and the periodicity of the data needs to be known in advance.
- Scalability: All three approaches suffer from limited scalability; the pixel space grows linearly with the size of the input.

On the other hand, VizTree is the first visualization tool that provides the capability of discovering non-trivial patterns, while using constant space, without imposing any constraints on the input data.

Our approach: VizTree

Our visualization approach works by transforming the time series into a symbolic representation, and encoding the data in a modified suffix tree in which the frequency and other properties of patterns are mapped onto colors and other visual properties. Before explaining our approach in detail, we will present a simple problem that motivates our work.

Two sets of binary sequences of length 200 were generated: the first set by the pseudo-random-number generator by the computer, and the second set by hand by a group of volunteers. The volunteers were asked to try and make the bit strings as random as possible, and were offered a prize to motivate them. Figure 4 shows one sample sequence from each set.

By simply looking at the original bit strings, it's difficult, if not impossible, to distinguish the computer-generated from the human-constructed numbers. However, if we represent them with a tree structure where the frequencies of subsequences are encoded in the thickness of branches, the distinction becomes clear. For clarity, the trees are pruned at depth three. Each tree represents one sequence from each set, and each node in the tree has exactly two branches: the upper branch represents 1, and the lower branch represents 0. The tree is constructed as follows: starting from the beginning of each sequence, subsequences of length three are extracted with a sliding window that slides across the sequence one bit at a time. So for the first sequence we get a set of subsequences {(0,1,0), (1,0,1), (0,1,1), …}.

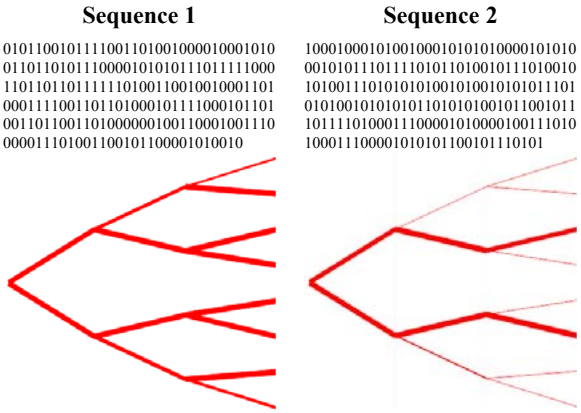

Figure 4: (*Left*) Computer-generated random bits presented as an augmented suffix tree (*Right*) Human-constructed bits presented as an augmented suffix tree.

For the tree shown on the left in Figure 4, the branches at any given level have approximately the same thickness, which means that the probabilities of subsequences at any given level are approximately evenly distributed. In contrast, the tree on the right shows that subsequences of alternating 0's and 1's dominate the whole sequence. The "motifs" for the sequence, 101 and 010, can be easily identified, since they appear more frequently than the other subsequences.

The non-randomness, which can be seen very clearly in this example, implies that humans usually try to "fake" randomness by alternating patterns [18]. Undoubtedly, there exist other solutions to uncover these "patterns" (entropy, Hidden Markov models, etc.). Nonetheless, what this visualization scheme provides is a straightforward solution that allows users to easily identify and view the patterns in a way intuitive to human perception.

The simple experiment demonstrates how visualizing augmented suffix trees can provide an overall visual summary, and potentially reveal hidden structures in the data. Since the strings represented in the tree are in fact "subsequences" rather than "suffixes," we call such trees *subsequence trees.*

This simple experiment motivates our work. Although time series are not discrete, they can be discretized with little loss of information, thus allowing the use of suffix/subsequence trees.

Our system is partly inspired by Visualysis [27], a visualization tool for biological sequences. Visualysis uses a suffix tree to store the biological sequences and, through the properties of the tree, such as bushiness, branch distribution, etc, and user navigation, interesting biological information can be discovered [27]. Visualysis incorporates algorithms that utilize suffix trees in computational biology; more specifically, exact sequence matching and tandem repeat algorithms. At a first glance, our visualization system is similar to Visualysis in the sense that it also has the objective of pattern discovery using a tree structure. However, several characteristics that are unique to our application make it more diversely functional than its computational-biology counterpart. First, although the tree structure needs the data to be discrete, the original time series data is not. Using a time-series discretization method that we introduced in an earlier work [29], continuous data can be transformed into discrete domain, with certain desirable properties such as lower-bounding distance, dimensionality reduction, etc. Second, instead of using a suffix tree, we use a subsequence tree that maps all subsequences onto the branches of the tree. Thus, given the same parameters, the trees have the same overall shape for any dataset. This approach makes comparing two time series easy and anomaly detection possible.

The utility of discretizing time series

In [29], we introduced Symbolic Aggregate approximation (SAX), a novel symbolic representation for time series. It is ideal for this application since, unlike all previously proposed discretization methods for time series, SAX allows lower-bounding distance measures to be defined on the symbolic space. In addition, its dimensionality reduction feature makes approximating large dataset feasible, and its ability to convert the data using merely the local information, without having to access the entire dataset, is especially desirable for streaming time series. The utility of SAX has been demonstrated in [29], and the adaptation or extension of SAX by other researchers further shows its impact in diverse fields such as medical and video [7, 36]. For these reasons, we choose to use SAX as the discretization method for the input time series data.

Before converting a time series to symbols, it should be normalized. The importance of normalization has been extensively documented in the past [24]. Without normalization, many time series data mining tasks have little meaning [24]. Therefore, by default, all subsequences are normalized before converting to symbols by SAX. In the unusual event where it might be more appropriate *not* to normalize, for example, when offset and amplitude changes are important, VizTree provides an option to skip the normalization step. SAX performs the discretization in two steps. First, a time series C of length n is divided into w equal-sized segments; the values in each segment are then approximated and replaced by a single coefficient, which is their average. Aggregating these w coefficients form the Piecewise Aggregate Approximation (PAA) representation of C.

Next, to convert the PAA coefficients to symbols, we determine the breakpoints that divide the distribution space into α equiprobable regions, where α is the alphabet size specified by the user. In other words, the breakpoints are determined such that the probability of a segment falling into any of the regions is approximately the same. If the symbols were not equi-probable, some of the substrings would be more probable than others. As a consequence, we would inject a probabilistic bias in the process. In [9], Crochemore et. al. showed that a suffix tree automation algorithm is optimal if the letters are equiprobable. Table 1 summarizes the major notation used in this and the subsequent sections of the paper.

Table 1: A summarization of the notation used in this paper

C	A time series $C = c_1,\ldots,c_n$ (in VizTree, C is a subsequence extracted by a sliding window of length n)
n	Length of the time series to be converted to string (in VizTree, this is the sliding window length)
w	The number of PAA segments representing time series C
α	Alphabet size (e.g., for the alphabet = {**a**,**b**,**c**}, α = 3)

Once the breakpoints are determined, each region is assigned a symbol. The PAA coefficients can then be easily mapped to the symbols corresponding to the regions in which they reside. In

[29], the symbols are assigned in a bottom-up fashion so the PAA coefficient that falls in the lowest region is converted to "*a*," in the one above to "*b*," and so forth. In this paper, for reason that will become clear in the next section, we reverse the assigning order, so the regions will be labeled top-down instead (i.e. the top-most region is labeled "a," the one below it "b," and so forth). Figure 5 shows an example of a time series being converted to string *acdcbdba*. Note the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction, and the symbols are approximately equiprobable.

The discretization technique can be applied to VizTree by calling SAX repeatedly for each subsequence. More specifically, subsequences of specified lengths are extracted from the input time series and normalized to have a mean of zero and a standard deviation of one. Applying SAX on these subsequences, we obtain a set of strings. From this point on, the steps are identical to the motivating example shown in the beginning of Section 3: the strings are inserted into the subsequence tree one by one.

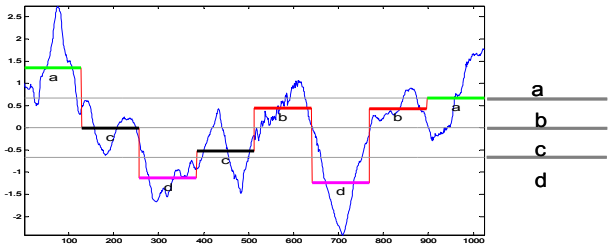

Figure 5: A time series dataset of electrical consumption (of length 1024) is converted into an eight-symbol string "**acdcbdba**." Note that the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction.

We believe that in order to visualize anomalies, one must have a representation that is capable of expressing *similarities*. The connection between similarity and anomalies is inherent in the English idiom. When confronted with an anomalous object or occurrence, people usually exclaim "*I have never seen anything like it*!" or "*It is like nothing you have ever seen before*". We have shown in [29] that the SAX representation is capable of representing similarities. In short, we showed that SAX is as good as, or better than, the other classic representations of time series, including DFT and DWT. Furthermore, we demonstrated the following unintuitive finding: SAX is generally better at discovering similarities than the original data! We refer interested readers to [29] for full detail.

A first look at VizTree

Figure 6 shows a screen shot of VizTree. When the program is executed, four blank panels and a parameter-setting area are displayed. To load a time series dataset, the user selects the input file using a familiar dropdown menu. The input time series is plotted in the top left-hand panel. The ruler drawn in the bottom of the panel shows the scale of the time series. The zoom-in and zoom-out buttons allow the user to view the time series in different scales; and the scrolling buttons allow the user to view different regions of the time series.

Next to the time series plotting window is the parameter setting area; the technician can enter the sliding window length, the number of SAX segments per window, and select alphabet size from a dropdown menu. Once the parameters are entered, the user can click on the "Show Tree" button to display the subsequence tree on the bottom left panel.

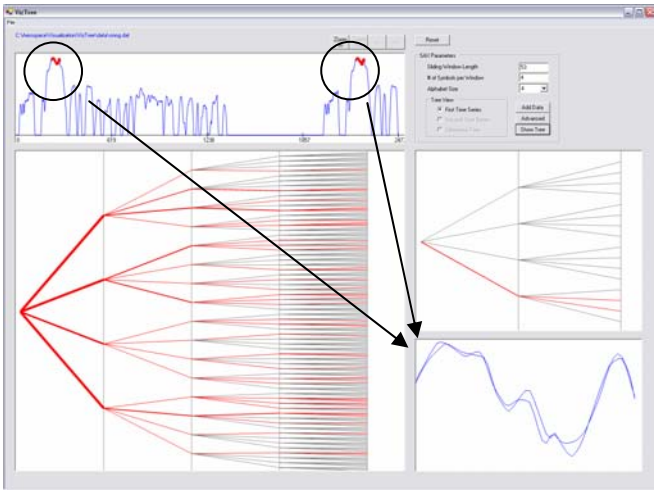

Figure 6: A screenshot of Viztree. The top panel is the input time series. The bottom left panel shows the subsequence tree for the time series. On the right, the very top is the parameter setting area. Next to the subsequence tree panel, the top window shows the zoom-in of the tree, and the bottom window plots the actual subsequences when the technician clicks on a branch.

The time series used for this example is a real, industrial dataset of smog emissions from a motor vehicle. The length of the time series is 2478. The length of the sliding window (i.e. *n* as in Table 1) is set to 53; the number of segments (i.e., the depth of the tree, or *w* as in Table 1) is four, and the alphabet size (i.e., the number of children for each node, or α as in Table 1) is four.

Each branch represents one pattern. As mentioned in the previous section, we reverse the assigning order of the symbols from bottom-up to top-down. The reason is that when the symbols are arranged this way, it is more consistent with the natural shape of the tree. For example, for any given node, a branch at a higher position denotes segments with higher values. Traversing breadth-first from the top-most branch of any given node, the symbols that represent the branches are *a, b, c*, and *d*, respectively. Each level of the tree represents one segment (i.e. one symbol). To retrieve any string, we simply traverse down the appropriate path.

Definition 2 *Pattern*: a pattern *p* is the SAX representation of a subsequence in the time series, denoted by the string *s* formed by following any path down the subsequence tree.

Definition 3 *Frequency*: The frequency of *p* in time series A is denoted by $f(p_A)$, which is the number of occurrences of *p* over the number of all occurrences in A.

The frequency of a pattern is encoded in the thickness of the branch. For clarity, the full tree is drawn. Branches with zero frequency are drawn in light gray, while others are drawn in red with varying thicknesses.

On the right hand side of VizTree, there are two panels. The upper one shows the zoom-in of the tree shown in the left panel. This is very useful especially for deep and bushy trees. The user can click on any node (on the subsequence tree window, or recursively, on the zoom-in window itself) and the sub-tree rooted at this node will be displayed in this upper panel. The sub-tree shown in Figure 6 is rooted at the node representing the string "*abxx*," where the "*xx*" denotes *don't-care* since we are not at the leaf level. If the user clicks on any branch, then the actual

subsequences having the string represented by this particular branch will be displayed in the bottom panel and highlighted in the time series plot window. In the figure, subsequences encoded to "*abdb*" are shown.

Parameter Selection

Three parameters need to be determined: the length of the sliding window, the number of segments, and the alphabet size. In [31] we showed the trade-off between the number of segments and the alphabet size. In general, VizTree works very well even with massive dimensionality reduction, as we will demonstrate later (in the experiments we used no more than 5 segments). The length of the sliding window is data-dependent; however, the ruler in the bottom of the time series window can offer some suggestion on the scale of patterns that might seem interesting to the user.

Subsequence Matching

Subsequence matching can be done very efficiently with VizTree. Instead of feeding another time series as query, the user provides the query in an intuitive way. Recall that each branch for any given node corresponds to one of the equiprobable regions that are used to convert the PAA coefficients to symbols. The top branch corresponds to the region with the highest values, and the bottom branch corresponds to the region with the lowest values. Therefore, any path can be easily translated into a general shape and can be used as a query. For example, the top-most branch at depth one (i.e., string "*axxx*") represents all subsequences that start with high values, or more precisely, whose values in the first segment have the mean value that resides in the highest region. In the previous example, the user is interested in finding a concave-down pattern (i.e., a U-shape). This particular pattern, according to the domain experts, corresponds to a change of gears in the motor vehicle during the smog emission test. From the U shape, the user can approximate the query to be something that goes down and comes up, or a path that starts and ends with high branches, with low branches in the middle. As a result, clicking on the branch representing "*abdb*" as shown in the figure uncovers the pattern of interest.

Motif Discovery & Simple Anomaly Detection

VizTree provides a straightforward way to identify motifs. Since the thickness of a branch denotes the frequency of the subsequences that are encoded to the given string, we can identify approximate motifs by examining the subsequences represented by thick tree paths. A feature unique to VizTree is that it allows users to visually evaluate and inspect the patterns returned. This interactive feature is important since different strings can also represent similar subsequences, such as those that differ by only one symbol. In addition, the user can prune off uninteresting or expected patterns to improve the efficiency of the system and reduce false positives. For example, for ECG data, the motif algorithm will mostly likely return normal heart beats as the most important motif, which is correct but non-useful. Allowing user to manually prune off this dominant pattern, secondary yet more interesting patterns may be revealed.

Figure 7 shows such an example. The dataset used here is a *real*, industrial dataset, "winding," which records the angular speed of a reel. The subsequences retrieved in the lower right panel have the string representation "*dacb*." Examining the motifs in this dataset allowed us to discover an interesting fact: while the dataset was advertised as real, we noted that repeated patterns occur at every 1000 points. For example, in Figure 7, the two nearly identical subsequences retrieved are located at offsets 729 and 1729, exactly 1000 points apart. We checked with the original author and discovered that this is actually a "fake" dataset synthesized from parts of a real dataset, a fact that is not obvious from inspection of the original data.

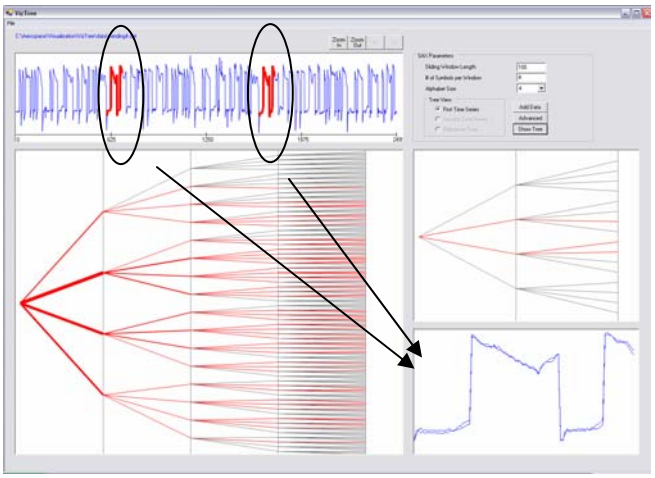

Figure 7: Example of motif discovery on the winding dataset. Two nearly identical subsequences are identified, among the other motifs.

The complementary problem of motif discovery is anomaly detection. While frequently occurring patterns can be detected by thick branches in the Viztree, simple anomalous patterns can be detected by unusually thin branches. Figure 8 demonstrates both motif discovery and simple anomaly detection on an MIT-BIH Noise Stress Test Dataset (ECG recordings) obtained from PhyioBank [14]. Here, motifs can be identified very easily from the thick branches; more remarkably, there is one very thin line straying off on its own (the path that starts with "a"). This line turns out to be an anomalous heart beat, independently annotated by a cardiologist as a premature ventricular contraction.

While anomalies can be detected this way for trivial cases, in more complex cases, the anomalies are usually detected by comparing the time series against a normal, reference time series. Anything that differs substantially from this reference time series can signal anomalies. This is exactly the objective of the Diff-Tree, as described in the next section.

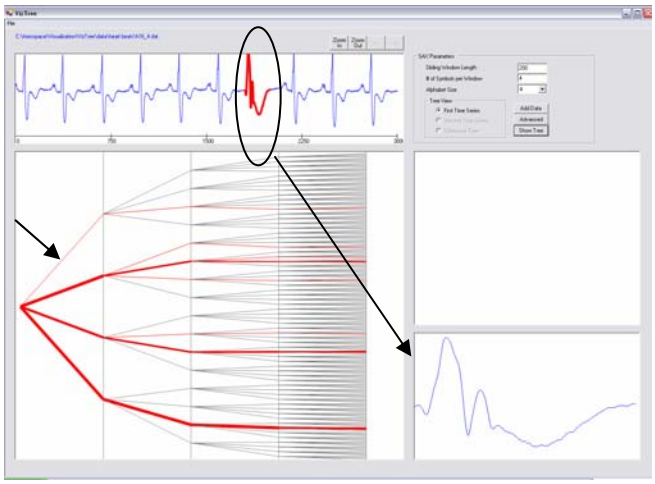

Figure 8: Heart-beat data with anomaly is shown. While the subsequence tree can be used to identify motifs, it can be used for simple anomaly detection as well.

Diff-Tree and anomaly detection

We have described how global structures, motifs, and simple anomalies can be identified by a subsequence tree. In this section, we extend these ideas to further allow the comparison of two time series by means of a "Diff-Tree." A Diff-Tree is short for "difference tree." As the name implies, it shows the distinction between two subsequence trees built for different time series. The construction of a Diff-Tree is fairly straightforward with the use of subsequence tree, since the overall tree shape is the same regardless of the strings, provided that the parameters selected (i.e., alphabet size, number of segment, etc) are the same. The Diff-Tree is constructed by computing the difference in thickness (i.e., frequency of occurrence) for each branch. Intuitively, time series data with similar structures can be expected to have similar subsequence trees, and in turn, a sparse Diff-Tree. In contrast, those with dissimilar structures will result in distinctively different subsequence trees and therefore a relatively dense Diff-Tree.

One or two datasets can be loaded to VizTree simultaneously. If only one is loaded, then its subsequence tree will be shown. If two datasets are loaded, the user has the option of viewing the subsequence tree of either one, or their diff-tree. The branches in the difference tree are color-coded to distinguish between the overrepresented and underrepresented patterns. Given two time series A and B, where A is the basis for comparison (the reference time series), and B is the second time series, we can define the following terms:

Definition 4 *Overrepresented pattern:* a pattern is overrepresented in B if it occurs more frequently in B than it does in A.

Definition 5 *Underrepresented pattern*: a pattern is underrepresented in B if it occurs more frequently in A than it does in B.

Two physical properties of the tree are used to denote the similarity/dissimilarity between two patterns:

- Thickness of a branch: denotes how different the given pattern is between time series A and B.
- Color intensity of a branch: denotes how significant the difference is.

We formally define the encoding schemes for these two properties below:

Definition 6 *Support*: the support of the difference for any pattern p between A and B, encoded as the thickness of a branch, is defined as follows:

$$Sup_p = \frac{f(p_B) - f(p_A)}{\max(f(p_A), f(p_B))} \quad (1)$$

Simply stated, Sup_p measures how a pattern (i.e. branch) differs from one time series to another, by computing the percentage of *non-overlap* of p between A and B. It is encoded in Diff-Tree as the thickness of a branch.

If a pattern p occurs less frequently in B than in A, then the pattern is underrepresented and $Sup_p < 0$, otherwise it is overrepresented and $Sup_p > 0$.

As an example, suppose the frequency of a pattern *p1* in A is 0.3 in A and 0.4 in B, then the support of difference for *p1* in A and B is

$$Sup_{p1} = \frac{0.4 - 0.3}{0.4} = 0.25$$

The lengths of the time series are implicitly taken into consideration as well, since f is scaled by the length of the data. For example, suppose |A| = 100, and suppose B is created by concatenating two copies of A, i.e. |B| = 200. Then by definition of f, any pattern will have the same frequency in A and B.

However, if there is another pattern *p2* in the same datasets such that $f(p2_A) = 0.03$ and $f(p2_B) = 0.04$, the formula will arrive at the same identical measure for both *p1* and *p2*! Logically, we would want to place more weight on *p1* since 30% occurrences is definitely more significant than 3% occurrences. Therefore, we define $conf_p$, the confidence of the difference for pattern p.

Definition 7 *Confidence*: the confidence of a pattern p describes how interesting or significant p is in each time series.

$$conf_p = \left(\frac{f(p_A)}{\max\_f_A} + \frac{f(p_B)}{\max\_f_B}\right)/2 \quad (2)$$

The measure *conf* takes on values between 0 and 1 -- 1 denotes the most significant and 0 being the least significant. Suppose the maximum frequency for both A and B is 0.45 (i.e. the most frequent pattern comprises 45% of all subsequences). Continuing with the previous examples, the first case ({0.3, 0.4}) will yield (0.3/0.45 + 0.4/0.45) / 2 = 0.9877, while the second case ({0.03, 0.04}) will yield (0.03/0.45 + 0.04/0.45) / 2 = 0.09877.

Note the measures *support* and *confidence* defined here are fundamentally different from, and thus should not be confused with those in the literature such as in association rule mining [2].

In the earlier version of VizTree [30], discrete colors are used to distinguish between overrepresented patterns and underrepresented patterns. We now add the confidence measures, encoded as the color intensity of a branch by varying the green and blue components of the RGB color value, to highlight the significance of pattern. So the overrepresented patterns will be drawn in different shades of green, whereas underrepresented patterns will be drawn in different shades of blue. When a pattern is neither overrepresented nor underrepresented (i.e. equal frequency in the test and reference time series), a neutral greenish-blue color is used.

Of course, alternative ways of computing the support and confidence are possible (e.g. for support, it's also possible to

compute just the absolute difference in frequencies rather than the *percentage* of the difference). For example, we might want to modify the definitions above for health surveillance applications, where we want to focus on overrepresented patterns that have a sudden increase in frequencies, since it could be indicative of a disease outbreak. However, from our experiments, we can safely say that in most cases, the choices of the formulas do not make much difference in the outcome, as long as they adhere to the general definitions of support and confidence: support measures how much non-overlap a pattern is between two time series, and confidence measures how significant the pattern is.

Diff-Tree shows all the differences between the reference and the test time series. However, in most anomaly detection applications, the user is interested in finding the most anomalous patterns. One way to prune off the insignificant differences is to display only the most surprising patterns. However, a better alternative is to display all of the differences, and rank the top, say ten, surprising patterns to aid user in the browsing process. Having defined the support and the confidence of the difference, we can now combine these two measures to define our ranking scheme, the measure of *surprisingness*. It is simply the support of a pattern weighted by its significance (*confidence*) to prevent two patterns of distinctive occurrence counts arriving at identical support measure (as described in the example provided before Definition 7).

Definition 8 *Surprisingness*: the degree of difference for pattern *p*, weighted by the confidence of the pattern, in two time series.

$$Surprise_p = \sup_p \times conf_p \qquad (3)$$

Predicting the frequency of a pattern

If a pattern does not exist in the reference time series (i.e. $f(p_A) = 0$), then its frequency can be predicted by using a Hidden Markov Model (HMM). A Markov model is a stochastic process over a set of finite states. Each state is associated with a specific probability distribution that governs the transition from one state to another. A Markov model of order M uses M previous states to determine the probabilities of possible outcomes of the next state [26].

When the true model is unknown, we have a Hidden Markov Model (HMM). A HMM is simply a Markov model where only the observations are known and the hidden state sequence is inferred. In our case, the expected frequency of any substring can be estimated from the observed sequence. Let *y* be a substring of *x*, and $m = |y| \geq M+2$, where M is the order of the model. A frequently used choice for the order of the chain is M = m-2, of which the model is called the maximal model. The expected frequency of *y*, $\hat{E}(Z_y)$, can be estimated as follows:

$$\hat{E}(Z_y) = \frac{\prod_{i=1}^{m-M} f_x(y_{[i...i+M]})}{\prod_{i=2}^{m-M} f_x(y_{[i...i+M-1]})} \qquad (4)$$

Once the first time series is loaded, the tree is annotated with the expected frequencies in the breadth-first order. The expected frequency for a pattern (or a partial pattern, if we are at the non-leaf level of the tree) is either the actual frequency if the pattern exists or the frequency predicted by HMM otherwise.

When predicting the frequency for a pattern, the appropriate order of the model needs to be determined. We start with the maximal model M, and if any of the substrings in Eq. 4 does not exist, then we fall back to the lower order until all the information needed to compute the estimate is sufficient. In other words, we look for the Markov model M in the interval [1, m-2] such that all the substrings $y_{[i...i+M]}$ occur in the reference time series, for all $1 <= i <= m-M$.

Having annotated the tree, we can discretize the subsequences and simply update the frequencies of the corresponding patterns in the existing tree when the second time series is loaded,

Diff-Tree example

The datasets used to demonstrate anomaly detection, constructed independently of the current authors and provided by the Aerospace Corporation for sanity check, are shown in Figure 9. The one on the top is the normal time series, and the one below is similar to a normal time series, except it has a gap in the middle as anomaly. Figure 10 shows a screenshot of the anomaly detection by diff-tree. The tree panel shows the Diff-Tree between the two datasets. The top two thick paths denote the beginning and the end of the anomaly, respectively. The last path denotes the normal pattern that occurs in both time series. Note that this pattern is not ranked since it is not an anomaly (i.e. equal occurrences in both time series).

This is a very trivial example for demonstration purpose. However, the effect is similar for more complex cases.

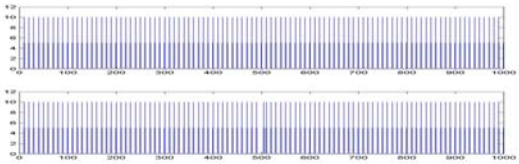

Figure 9 The input files used for anomaly detection by diff-tree. *(Top)* Normal time series. *(Bottom)* Anomaly is introduced as a gap in the middle of the dataset.

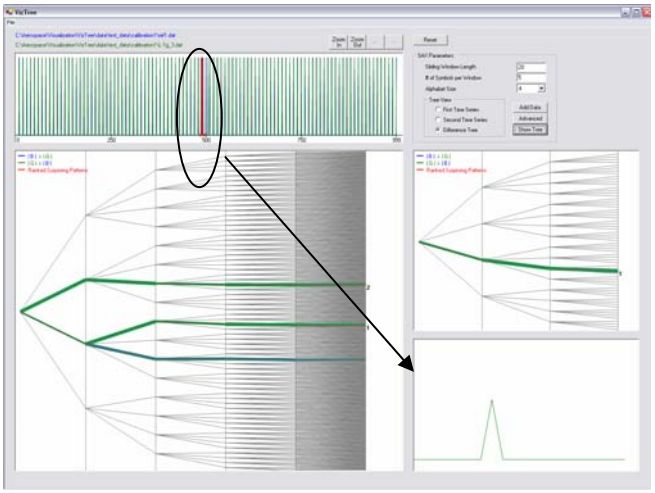

Figure 10: Diff-tree on the datasets shown in the previous figure. The gap is successfully identified.

Diff-Tree dissimilarity coefficient

Eq. 3 defines the surprisingness of each pattern. An intuitive question to ask is, can we quantify a Diff-Tree by combining the surprise measures of *all* patterns? We formalize this idea as follows:

$$DSim_{A,B} = \sum_{p} Surprise_p, \forall p \in A, B \qquad (5)$$

Simply stated, $DSim_{A,B}$ measures the dissimilarity between time series A and B. Before continuing with the implication of this measurement, we will demonstrate how well the coefficient captures the dissimilarity between two time series by applying it on time series clustering.

The choice of clustering algorithm for demonstration of the coefficient is hierarchical clustering, since its great visualization power helps for the validation purpose. As a quick review, hierarchical clustering produces a nested hierarchy of similar groups of objects, according to a pairwise distance matrix of the objects. Table 2 outlines the basic hierarchical clustering algorithm. In our case, we'll use the similarity measures (i.e. 1–DSim) for the matrix instead of the actual distances.

Table 2: An outline of hierarchical clustering.

Algorithm Hierarchical Clustering	
1.	Calculate the distance between all objects. Store the results in a distance matrix.
2.	Search through the distance matrix and find the two most similar clusters/objects.
3.	Join the two clusters/objects to produce a cluster that now has at least 2 objects.
4.	Update the matrix by calculating the distances between this new cluster and all other clusters.
5.	Repeat step 2 until all cases are in one cluster.

Figure 11 shows the dendrogram of clustering result using the Diff-Tree (dis)similarity coefficient. It clearly demonstrates that the coefficient captures the dissimilarity very well and that all clusters are separated perfectly.

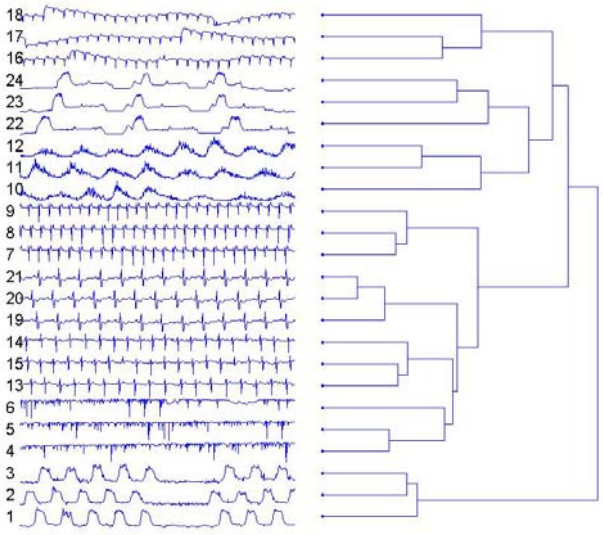

Figure 11: Clustering result using the Diff-Tree dissimilarity coefficient

As a reference, we ran the same clustering algorithm using the widely-used Euclidean distance. The result is shown in Figure 12.

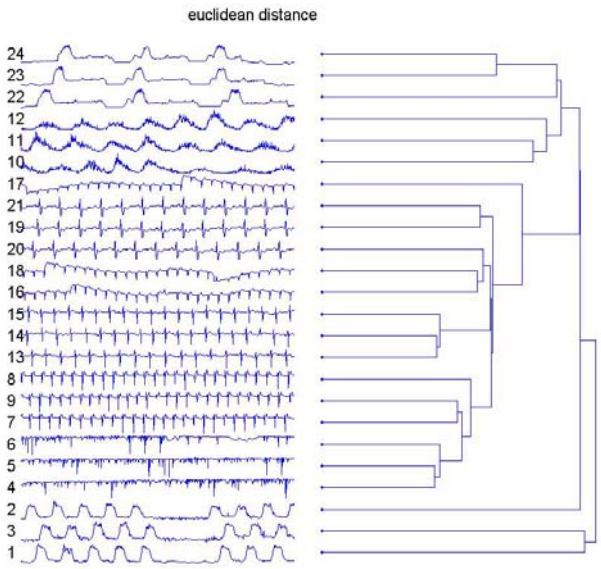

Figure 12: Clustering result using Euclidean distance.

While we are not claiming that the coefficient is the best measure for clustering time series, the demonstration shows that it is at least competitive with one of the most popular distance measure. In addition, it indirectly validates the correctness of the measures used in Diff-Tree.

Experimental evaluation

We have done an intensive empirical evaluation on VizTree. It would make little sense for us to test our approach on datasets where we have little expertise or intuition. We have therefore chosen to evaluate (and demonstrate) our approach on datasets which are either very intuitive to the average person or have been extensively annotated by domain experts. In particular, we will evaluate our work on human motion data, the power demand data, and the ECG data Note that all datasets used here are available for free from the UCR archive [22].

Subsequence matching

This experiment incorporates both subsequence matching and motif discovery. The dataset used is the human motion data of yoga postures. A model postured yoga routines in front of a green screen, and her motion was captured by various sensors. The motion capture is transformed into a time series by computing the aspect ratio of the minimum-bounding rectangle formed around her body. The length of the time series is approximately 26,000 (i.e. there are approximately this many frames in the original video).

Suppose we are interested in finding a yoga sequence like the one in Figure 13:

Figure 13: A sample yoga sequence for approximate subsequence matching.

Then we would expect the shape of the query to descend rapidly after the first position (the width-to-height-ratio decreases), ascend

slowly after the second position, descend again, and finally ascend once more. Assume that we set the number of segments to be five (an arbitrary choice), then a reasonable start would be the branch "*adxxx*." Since there are only two paths extending from the node "*ad*," the matches are found very quickly without much refinement in the search space. The result is shown in Figure 14 and the actual yoga sequences for the matches are outlined in Figure 15. The subsequence length is 400 (i.e. about 6.5 seconds). As the figure shows, the two sequences are very similar with only very minor distinction.

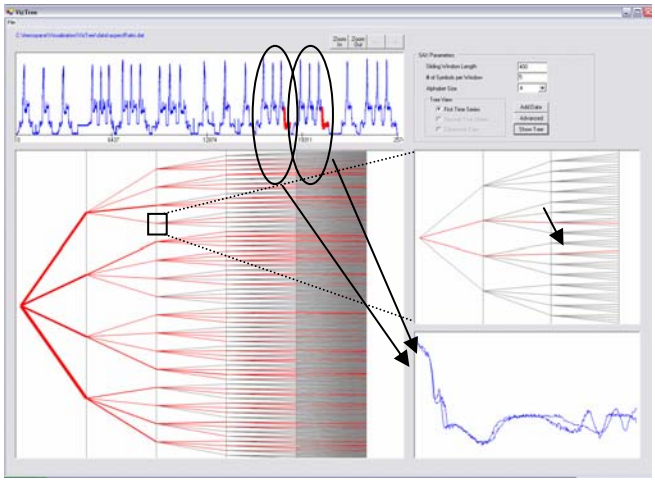

Figure 14: Matches for the yoga sequence in Figure 11. The bottom right corner shows how similar these two subsequences are.

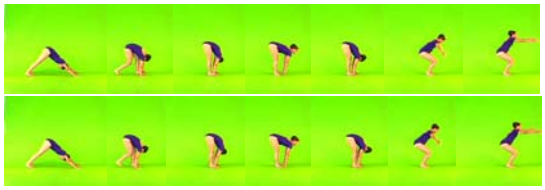

Figure 15: Outline of the actual yoga sequences that match the query.

There are several advantages of the approximate subsequence matching by VizTree. One is that this feature is built-in to the application, and it is relatively easy to specify the query without explicitly providing it. More importantly, the system retrieves the results very efficiently since the information is already stored in the tree. With the current state-of-the-art *exact* subsequence matching algorithms, retrieval is much too slow for a real time interaction.

Motif Discovery

For the motif discovery experiment, we will continue with the previous human-motion example. There are obviously some noticeable motifs such as the long spikes that occur throughout the sequence (see the time series plot in Figure 14). They denote the posture where the model is lying flat on the ground, when the aspect ratio is at its maximum. However, one of the desirable features of VizTree is that it allows users to visually identify secondary yet more interesting motifs. The matches found in the previous section are such example. We can zoom-in on these subsequences and examine their similarity.

From Figure 16 we can see that these two subsequences are indeed very similar to each other. Note that they both have a small dip towards the end of the sequence. However, there is a slight difference there – the dip for the first sequence occurs before that for the second sequence, and is followed by a plateau. Examining the motion captures we discover that the dip corresponds to the 6th position shown in Figure 15, right before the model stretched her arms straight in front of her. In addition, for the first sequence, the model held that last position for a longer period of time, thus the plateau following the dip. These subtle differences are difficult to notice without the motif discovery and/or the subsequence matching features in VizTree.

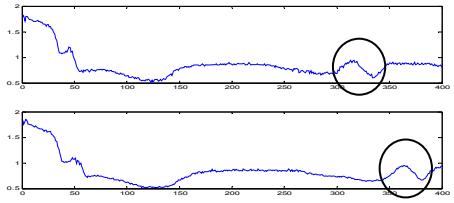

Figure 16 Zoom-ins of the two matches found in the yoga subsequence match example. Note that they both have a dip towards the end of the sequences.

For comparison, we ran the fastest known *exact* motif discovery algorithm [31] . Although the same motif can also be successfully identified, it takes minutes to compute, while VizTree gives instant (less than one second) feedback on the results. Even with the *approximate* motif discovery algorithm [8], it takes tens of seconds to complete. In addition, the visualization power of VizTree allows the user to see exactly where the motif occurs and how it maps to the original time series.

Anomaly detection

For anomaly detection, we used the power demand data that was also used in Figure 3. Electricity consumption is recorded every 15 minutes; therefore, for the year of 1997, there are 35,040 data points. Figure 17 shows the resulting tree with the sliding window length set to 672 (exactly one week of data), and both alphabet size and number of segments to 3. The majority of the weeks follow the regular Monday-Friday, 5-working-day pattern, as shown by the thick branches. The thin branches denote the anomalies (anomalies in the sense that the electricity consumption is abnormal given the day of the week). The one circled is from the branch "*bab*." The zoom-in shows the beginning of the three-day week during Christmas (Thursday and Friday off). The other thin branches denote other anomalies such as New Year's Day, Good Friday, Queen's Birthday, etc.

While other anomaly detection algorithms such as the TSA-Tree Wavelet-based algorithm by Shahabi et. al. [39] and the Immunology-based algorithm (IMM) by Dasgupta and Forrest [10] can potentially find these anomalies as well given the right parameters, both are much more computationally intensive. While VizTree requires input of parameters, the results are almost instant. In the contrary, the TSA-Tree takes tens of seconds, and IMM needs re-training its data with every adjustment of parameters, with each training session taking several minutes. This is clearly untenable for massive datasets.

In addition to the fast computational time, anomaly detection by VizTree does not always require a training dataset. As

demonstrated, simple anomalies can be identified as an inverse to the motifs.

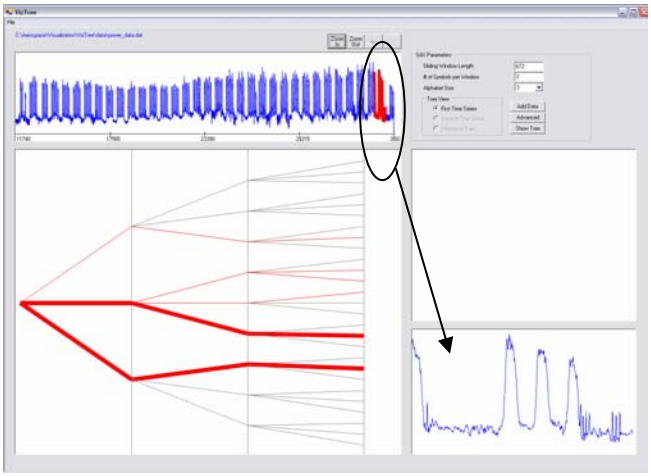

Figure 17 Anomaly detection on power consumption data. The anomaly shown here is a short week during Christmas.

Anomaly detection by Diff-Tree

In Figure 18, two ECG datasets are loaded to the application. The reference time series is a normal heartbeat dataset, and the test time series has an anomalous heartbeat. The top ten surprising patterns, ranked by their levels of surprisingness, are shown, and red lines are drawn on top of the surprising branches to make them more easily spotted. Clicking on the number-one ranking pattern displays the anomalous heartbeat in the test time series.

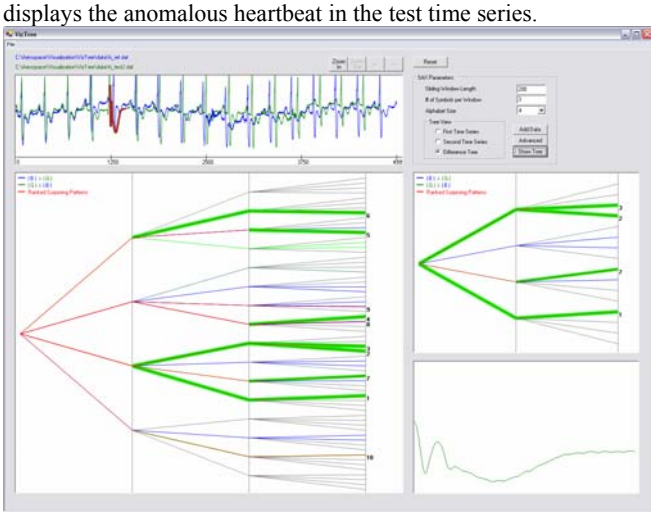

Figure 18: An abnormal heartbeat dataset is compared against a normal heartbeat dataset, and the anomaly is successfully identified.

Complexity and scalability

VizTree is very efficient in terms of both time and space. The discretization step and the construction [26] of the tree both take linear time. The pixel space of the tree is determined solely by the number of segments and alphabet size. In particular, we note that the pixel size of the tree is *constant* and independent of the length of time series. We have already shown that large amounts of dimensionality reduction do not greatly affect the accuracy of our results (in Section 5.3, the dimensionality is reduced from 672 to 3, a compression ratio of 224-to-1). However, the size of the dataset plays a role in memory space, since each node in the tree stores the offsets of its subsequences. However, SAX allows efficient numerosity reduction to reduce the number of subsequences being included into the tree, in addition to alleviating the problem associated with trivial matches (see below) [25, 31].

Numerosity Reduction

In [31] we showed that the best matches for a subsequence tend to be its immediate neighbors: the subsequence one point to the right and the subsequence one point to the left. We defined these matches to be the "trivial matches." In the smooth regions of the time series, the amount of trivial matches might be large. If we include them in any sliding-window algorithms, the trivial matches will dominate over the true patterns due to over-counting, and the results will likely be distorted, or worse, become meaningless [25]. Therefore, when extracting subsequences from the time series by a sliding window, the trivial matches should be excluded.

Different definitions can be used to identify trivial matches. The easiest way is to compare the SAX strings and only record a subsequence if its string is different from the last one recorded. In other words, no two consecutive strings should be the same.

Additionally, we can also check two consecutive strings symbol-by-symbol and consider them trivial matches of one another if no pair of symbols is more than one alphabet apart. This extra check is based on the same idea as the previous numerosity reduction option, that similar subsequences have the same SAX representation. However, it is also likely that similar subsequences do not have exactly the same SAX representations; rather, they might have alphabets that differ by at most one at any given position (i.e. the values could be very close but reside on different sides of a breakpoint).

Furthermore, the second option can be extended to also exclude non-monotonic strings. Depending on the nature of the datasets, users might only be interested in finding patterns with ups and downs.

Finally, the ultimate numerosity reduction can be achieved by chunking, which allows no overlapping subsequences. This has been used for many approaches; however, we note that it is only useful if the dataset exhibits regular patterns, either by shape or by period. For example, if we use chunking for the power consumption data, then we get an even more distinctive tree.

Conclusions and future work

We proposed VizTree, a novel visualization framework for time series that summarizes the global and local structures of the data. We demonstrated how pattern discovery can be achieved very efficiently with VizTree.

While we mainly focus on the "mining" aspect in major part of this paper, we mentioned that VizTree can potentially be used for monitoring purposes. There are typically two types of streaming algorithms: append-only and amnesic. The former maintains some global statistics or information about the overall data as new data arrive, while for the latter, more recent data are treated with higher weights. For the append-only case, the tree can be updated very easily in constant time (of course, discretization still takes linear-time, although some optimization can be achieved by updating the PAA values instead of re-computing the whole thing). For the amnesic case, information on older data needs to be

stored in order to update the tree. We defer this part for future research.

We believe that researchers from other sectors of the industry can greatly benefit from our system as well. For example, it could potentially be used for indexing and editing video sequences.

Currently, we are working on extending VizTree/Diff-Tree for epidemic outbreak detection [44] on a collaborative effort with AT&T Research. Most outbreaks can be noticed in the late stage of the epidemic; however, the goal is to detect the outbreak in its early stage and mitigate its effect. In addition, it is also useful for bioterrorism detection [32], public health surveillance, and learning disease patterns.

Acknowledgments

Thanks to Victor Zordan and Bhrigu Celly for providing the yoga postures data.

References

1 Aggarwal C, *Towards Effective and Interpretable Data Mining by Visual Interaction*, in *SIGKDD Explorations*. 2002.

2 Agrawal R, Imielinski T, and Swami A. Mining Association Rules Between Sets of Items in Large Databases. *the 1993 ACM SIGMOD Int'l Conference on Management of Data* 1993 (Washington, D.C.); 207-216.

3 Apostolico A, Bock ME, and Lonardi S. Monotony of Surprise in Large-Scale Quest for Unusual Words. *the 6th Int'l Conference on Research in Computational Molecular Biology* 2002 (Washington, D.C.); 22-31.

4 Caraca-Valente JP and Lopez-Chavarrias I. Discovering Similar Patterns in Time Series. *the 6th Int'l Conference on Knowledge Discovery and Data Mining* 2000 (Boston, MA); 497-505.

5 Cardle M, *Ph.D Thesis, in progress*. University of Cambridge, 2004.

6 Chang CLE, Garcia-Molina H, and Wiederhold G. Clustering for Approximate Similarity Search in High-Dimensional Spaces. *IEEE Transactions on Knowledge and Data Engineering* 2002; **14**(4): 792-808.

7 Chen L, Ozsu T, and Oria V, *Symbolic Representation and Retrieval of Moving Object Trajectories*. 2003, University of Waterloo.

8 Chiu B, Keogh E, and Lonardi S. Probabilistic Discovery of Time Series Motifs. *the 9th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* 2003 (Washington DC, USA); 493-498.

9 Crochemore M, et al. Speeding Up Two String-Matching Algorithms. *Algorithmica* 1994; **12**(4/5): 247-267.

10 Dasgupta D and Forrest S. Novelty Detection in Time Series Data Using Ideas from Immunology. *the 8th Int'l Conference on Intelligent Systems* 1999 (Denver, CO).

11 Durbin R, et al., *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

12 Faloutsos C, Ranganathan M, and Manolopulos Y. Fast Subsequence Matching in Time-Series Databases. *SIGMOD Record* 1994; **23**(2): 419-429.

13 Gabglio A, *Theoria Generale Della Statistica*. 2nd ed: Milan, 1888.

14 Goldberger AL, et al. PhysioBank, PhysioToolkit, and PhysioNet: Componenets of a New Research Resource for Complex Physiologic Signals. *Circulation* 2000; **101**(23): e215-e220.

15 Hochheiser H and Shneiderman B. Interactive Exploration of Time-Series Data. *the 4th Int'l Conference on Discovery Science* 2001 (Washington D.C.), Springer-Verlag; 441-446.

16 Hochheiser H and Shneiderman B. Dynamic Query Tools for Time Series Data Sets: Timebox widgets for interactive exploration. *Information Visualization* 2004; **3**: 1-18.

17 Huang YW and Yu PS. Adaptive Query Processing for Time-Series Data. *the 5th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* 1999 (San Diego, CA); 282-286.

18 Huettel S, Mack PB, and McCarthy G. Perceiving Patterns in Random Series: Dynamic Processing of Sequence in Prefrontal Cortex. *Nature Neuroscience* 2002; **5**: 485-490.

19 Isaac D and Lynnes C, *Automated Data Quality Assessment in the Intelligent Archive, White Paper prepared for the Intelligent Data Understanding program*. 2003. p. 17.

20 Jin X, et al. Indexing and Mining of the Local Patterns in Sequence Database. *the 3rd Int'l Conference on Intelligent Data Engineering and Automated Learning* 2002 (Manchester, UK); 68-73.

21 Keim DA. Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics* 2002; **8**(1): 1-8.

22 Keogh E, *The UCR Time Series Data Mining Archive*. 2002, Computer Science & Engineering Department, University of California: Riverside, CA.

23 Keogh E, Chakrabarti K, and Pazzani M. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *SIGMOD Record* 2001; **30**(2): 151-162.

24 Keogh E and Kasetty S. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* 2002 (Edmonton, Alberta, Canada); 102-111.

25 Keogh E and Lin J. Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research. *Knowledge and Information Systems Journal* 2004.

26 Keogh E, Lonardi S, and Chiu B. Finding Surprising Patterns in a Time Series Database in Linear Time and Space. *the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* 2002 (Edmonton, Alberta, Canada); 550-556.

27 Kim S, et al. Visualysis: A Tool for Biological Sequence Analysis. *the 4th Int'l Conference on Computational Molecular Biology* 2000 (Tokyo, Japan).

28 Lankford JP and Quan A. Evolution of Knowledge-Based Applications for Launch Support. *Ground System Architecture Workshop* 2002 (El Segundo, CA).

29 Lin J, et al., *A Symbolic Representation of Time Series, with Implications for Streaming Algorithms*, in *Workshop on Research Issues in Data Mining and Knowledge Discovery*. 2003: San Diego, CA.

30 Lin J, et al. Visually Mining and Monitoring Massive Time Series. *the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 2004 (Seattle, WA); 460-469.

31 Lin J, et al., *Finding Motifs in Time Series*, in *the 2nd Workshop on Temporal Data Mining*. 2002: Edmonton, Alberta, Canada.

32 Lober WB, et al. Roundtable on Bioterrorism Detection: Information System-Based Surveillance. *J AM Med Inform Assoc* 2002; **9**(2): 105-115.

33 Ma J and Perkins S. Online Novelty Detection on Temporal Sequences. *the 9th Int'l Conference on Knowledge Discovery and Data Mining* 2003 (Washington D.C.).

34 Oates T. Identifying Distinctive Subsequences in Multivariate Time Series by Clustering. *the 5th Int'l Conference on Knowledge Discovery and Data Mining* 1999 (San Diego, CA); 322-326.

35 Oates T, Schmill M, and Cohen P. A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgements. *the 17th National Conference on Artificial Intelligence* 2000; 846-851.

36 Ohsaki M, et al., *A Rule Discovery Support System for Sequential Medical Data, in the Case Study of a Chronic Hepatitis Dataset*, in *Discovery Challenge Workshop*. 2003: Cavtat-Dubrovnik, Croatia.

37 Park S, et al. Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases. *the 16th IEEE Int'l Conference on Data Engineering* 2000 (San Diego, CA); 22-32.

38 Reinert G, Schbath S, and Waterman MS. Probabilistic and Statistical Properties of Words: An Overview. *Journal of Computational Biology* 2000; **7**: 1-46.
39 Shahabi C, Tian X, and Zhao W. TSA-Tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries. *the 12th Int'l Conference on Scientific and Statistical Database Management* 2000 (Berlin, Germany); 55-68.
40 Shneiderman B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *the IEEE Symposium on Visual Languages* 1996 (Boulder, CO), IEEE Computer Society Press; 336-343.
41 Tanaka Y and Uehara K. Discover Motifs in Multi Dimensional Time-Series Using the Principal Component Analysis and the MDL Principle. *the 3rd Int'l Conference on Machine Learning and Data Mining in Pattern Recognition* 2003 (Leipzig, Germany); 252-265.
42 Tufte ER, *The Visual Display of Quantitative Information*. Graphics Press: Cheshire, CT, 1983.
43 van Wijk JJ and van Selow ER. Cluster and Calendar Based Visualization of Time Series Data. *1999 IEEE Symposium on Information Visualization* 1999 (San Francisco, CA); 4-9.
44 Wagner MM, et al., *Syndrome and Outbreak Detection Using Chief-Complaint Data - Experience of the Real-Time Outbreak and Disease Surveillance Project*, in *Morbidity and Mortality Weekly Report*. 2004, Center for Disease Control and Prevention. p. 28-31.
45 Weber M, Alexa M, and Muller W. Visualizing Time Series on Spirals. *2001 IEEE Symposium on Information Visualization* 2001 (San Diego, CA); 7-14.