VizTree: a Tool for Visually Mining and Monitoring Massive Time Series Databases

Jessica Lin[a] Eamonn Keogh[a] Stefano Lonardi[a] Jeffrey P. Lankford[b] Daonna M. Nystrom[b]

[a]Computer Science & Engineering Department
University of California, Riverside
Riverside, CA 92521
{jessica, eamonn, stelo}@cs.ucr.edu

[b]The Aerospace Corporation
El Segundo, CA 90245-4691
{Jeffrey.P.Lankford, Donna.M.Nystrom}@aero.org

Abstract

Moments before the launch of every space vehicle, engineering discipline specialists must make a critical *go/no-go* decision. 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. The Aerospace Corporation is responsible for providing engineering assessments critical to the *go/no-go* decision for every Department of Defense (DoD) launch vehicle. These assessments are made by constantly monitoring streaming telemetry data in the hours before launch. For this demonstration, we will introduce VizTree, a novel time-series visualization tool to aid the Aerospace analysts who must make these engineering assessments. VizTree was developed at the University of California, Riverside and is unique in that the same tool is used for mining archival data and monitoring incoming live telemetry. Unlike other time series visualization tools, VizTree can scale to very large databases, giving it the potential to be a generally useful data mining and database tool.

1. Introduction

One of the crucial responsibilities of The Aerospace Corporation is to provide engineering assessments for the government engineering discipline specialists who make the critical *go/no-go* decision moments before the launch of every DoD space vehicle. The analyst making these engineering assessments has access to data from previous launches and must constantly monitor streaming telemetry from the current mission. Currently, the analysts 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 analyst.

To reduce the possibility of wrong *go/no-go* decisions, The Aerospace Corporation 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 the massive databases collected during 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 [3].
- Producing better techniques to visualize the streaming telemetry data in the hours before launch. This is particularly challenging because analysts may have to monitor dozens of rapidly changing sensors [3].

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 will demonstrate 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), surprising patterns (anomaly detection), and query by content. The user interactive paradigm allows users to visually explore the time series, and perform real-time hypotheses testing.

2. 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.

In [5], we introduced Symbolic Aggregate approximation (SAX), a novel symbolic representation for time series that transforms a time series into equiprobable symbols. The utility of SAX has been demonstrated in [5], and adaptations or extensions of SAX by other researchers further shows its impact in diverse fields such as medical data mining and video indexing [1, 7]. We refer interested readers to [5] for more details on SAX. Figure 1

* Dr. Keogh is supported by NSF Career Award IIS-0237918

Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004

shows an example of how a time series of length 1024 is converted to a string of length eight: "*acdcbdba*." In this example, the number of SAX symbols is eight, and the cardinality of alphabet is four (i.e. *a, b, c,* and *d*).

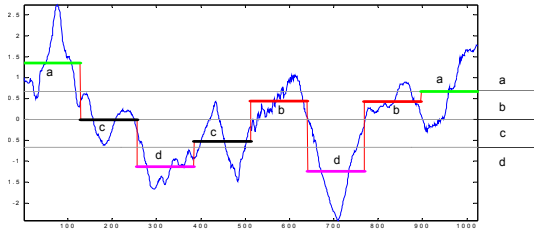

Figure 1: A time series dataset of length 1024 is converted into an eight-symbol string "**acdcbdba**." Note that the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction.

To construct a tree representing the input time series, subsequences of specified lengths are extracted from the time series via a sliding window 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, and these strings are inserted into the tree one by one. Each branch/node represents one symbol. The resulting tree is a complete tree with depth equals to the number of SAX symbols. Each node in the tree has α children, where α is the cardinality of alphabet (i.e. if the alphabet size is four, then each node has children denoting *a, b, c*, and *d*, respectively).

Figure 2 shows a simple example of the tree, representing strings of length three with cardinality of two. If we have a string *aba*, then we insert it into the tree, following the top thick path: the first symbol, *a*, is inserted into the first child node, A, of the root; the second symbol, *b*, is inserted into the second child node, AB, of node A; and the last symbol, *a*, is inserted into the first child node, ABA, of node AB. Each time a symbol is inserted, its frequency of occurrence, which is reflected as the thickness of the branch, is updated. The frequently occurring patterns (motifs) "*aba"* and "*bab"* can be easily identified from the tree, since these two paths are thicker compared to the other branches.

We call such trees *subsequence trees*. Differing from a classic suffix tree, a subsequence tree 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 arbitrarily long time series easy and, as we shall see, it makes anomaly detection possible.

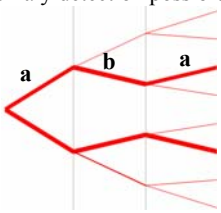

Figure 2: Subsequence tree for strings of length three and cardinality of two. The motifs "aba" and "bab" can be easily identified.

2.1 A first look at VizTree

Figure 3 shows a screen shot of VizTree[1]. 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. Next to the time series plotting window is the parameter setting area; the analyst 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.

The time series used for this example is an industrial dataset of smog emissions from a motor vehicle. The length of the time series is 2478. The length of the sliding window is arbitrarily set to 53; the number of segments (i.e., the depth of the tree) is four, and the alphabet size (i.e., the number of children for each node) is four.

The mappings of the symbols are 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. To retrieve any string, we simply traverse down the appropriate branches.

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.

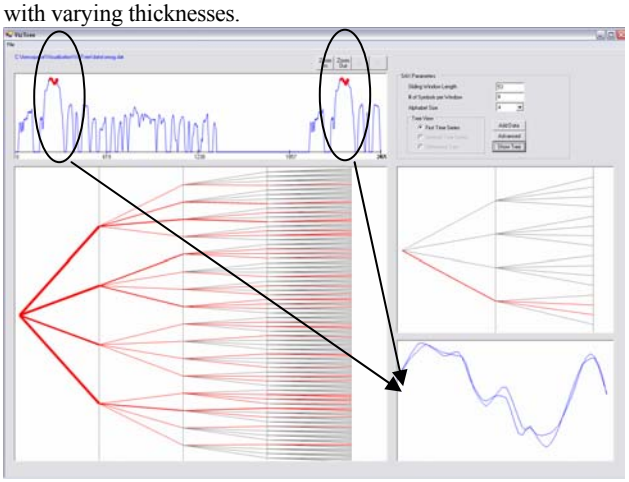

Figure 3: A screenshot of Viztree. The top panel is the input time series. The bottom left panel shows the subsequence tree for the time series. On the right, the very top is the parameter setting area. Next to the subsequence tree panel, the top window shows the zoom-in of the tree, and the bottom window plots the actual subsequences when the analyst clicks on a branch.

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) and the sub-tree rooted at this node will be displayed in this upper panel. The sub-tree shown in Figure 3 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.

[1] We note that all the figures in this text suffer from their small scale and monochromatic printing. We encourage the interested reader to visit [4] to view high-resolution full-color examples.

2.2 Subsequence matching

Similarity search can be done very efficiently with VizTree. Instead of feeding another time series as query, the user provides the query in an intuitive way. 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 string 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.

2.3 Motif discovery

A substantial body of literature has been devoted to techniques to discover frequently recurring, overrepresented patterns in time series; however, each work considered a different definition of *pattern*. In previous work, we unified and formalized the problem by defining the concept of "time series motif" [6].

VizTree provides a straightforward way to identify motifs. Since the thickness of a branch denotes the frequency of the subsequences having the same, corresponding strings, 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. Figure 4 shows an example.

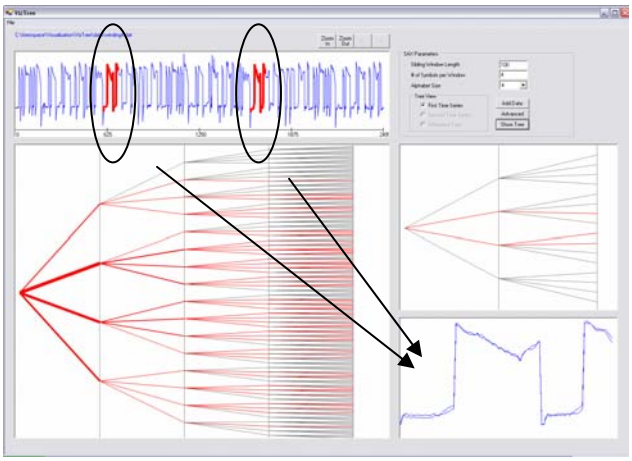

Figure 4: Example of motif discovery on the winding dataset. Two nearly identical subsequences are identified, among the other motifs.

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 4, the two nearly identical subsequences retrieved are located at offsets 599 and 1599, exactly 1000 points apart. We checked with the original author and discovered that this is actually a synthetic dataset, composed from parts of a real dataset, a fact that is not obvious from inspection of the original data.

2.4 Simple anomaly detection

The complementary problem of motif discovery is anomaly detection. While frequently occurring patterns can be detected by thick branches in the VizTree, unusually thin branches can signal simple anomalous patterns. Figure 5 demonstrates both motif discovery and simple anomaly detection on an MIT-BIH Noise Stress Test Dataset (ECG recordings) obtained from PhyioBank [2]. 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 heartbeat, independently annotated by a cardiologist as a premature ventricular contraction.

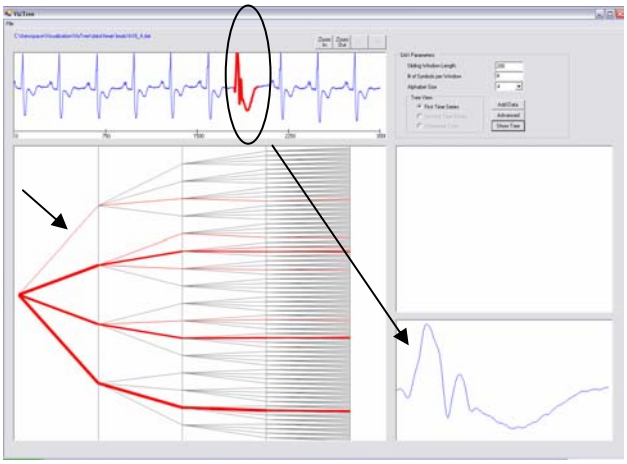

Figure 5: Heartbeat data with anomaly. The thick lines represent the reoccurring normal heartbeat; the thin line pointed to by the short arrow suggests an infrequently occurring pattern, an *anomaly*. Simply by clicking on this line the source of the data is highlighted in the top panel, and a zoom-in is shown in the bottom right panel.

As another motivating example, we used a power demand dataset provided by a Dutch research facility. Electricity consumption is recorded every 15 minutes; therefore, for the year of 1997, there are 35,040 data points. Figure 6 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. 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"[2] such as New Year's Day, Good Friday, Queen's Birthday, etc.

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.

[2] "Anomalies" in the sense that the electricity consumption is abnormal given the day of the week.

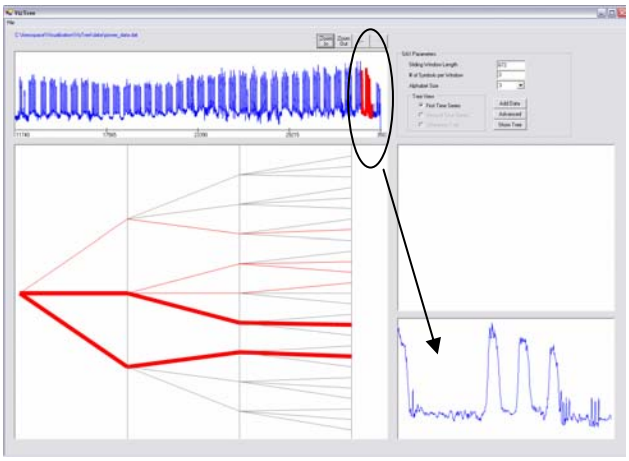

Figure 6: Anomaly detection on power consumption data. The anomaly shown here is a short week during Christmas.

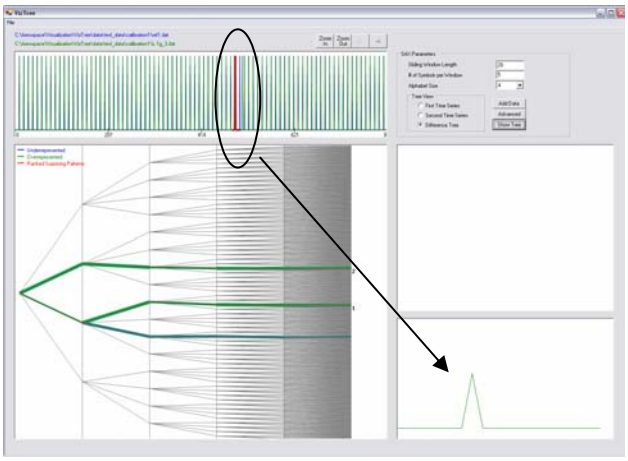

Figure 8: Diff-tree on the datasets shown in the previous figure. The gap is successfully identified.

3. Diff-tree

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 shows the distinction between two time series. It is constructed by computing the difference in thickness (i.e., frequency of occurrence) for each branch between two subsequence trees. 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.

3.1 Anomaly detection

The datasets used for anomaly detection, constructed independently of the current authors and provided by The Aerospace Corporation for a sanity check, are shown in Figure 7. 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 8 shows a screenshot of the anomaly detection by diff-tree. The tree panel shows the diff-tree between the two datasets. The two thick paths denote the beginning and the end of the anomaly, respectively. This is a very trivial example for demonstration purpose. However, the effect is similar for more complex cases.

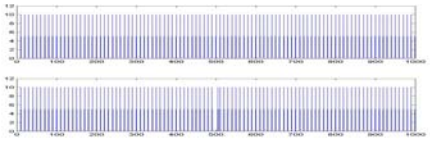

Figure 7: The input files used for anomaly detection by diff-tree. *(Top)* Normal time series. *(Bottom)* Anomaly is introduced as a gap in the middle of the dataset.

3.2 Scalability

The pixel space of the subsequence 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 to the length of time series. With a slider on the time series-viewing panel (not shown on the simple examples in this paper), VizTree can accommodate *massive* time series with a constant-size tree. This desirable property makes it easy to view and summarize large time series database on one screen. We have already shown that large amounts of dimensionality reduction do not greatly affect the accuracy of our results (in Section 2.4, the dimensionality is reduced from 672 to 3, a compression ratio of 224-to-1). The size of the database plays a role in memory requirements only for subsequence retrieval purpose, and here we use modified B-trees to allow real time retrieval.

4. References

[1] Chen, L., Ozsu, T. & Oria, V. (2003). Symbolic Representation and Retrieval of Moving Object Trajectories. Univ. of Waterloo. 2003.

[2] Goldberger, A. L., et. al. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Componenets of a New Research Resource for Complex Physiologic Signals. *Circulation*. vol. 101(23), June 13. pp. e215-e220. http://circ.ahajournals.org/cgi/content/full/101/23/e215]

[3] Lankford, J. P. & Quan, A. (2002). Evolution of Knowledge-Based Applications for Launch Support. In *proceedings of Ground System Architecture Workshop*. El Segundo, CA.

[4] Lin, J. VizTree Website. *http://www.cs.ucr.edu/~jessica/VLDB04.htm*

[5] Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003). A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, *the 8th ACM SIGMOD*. San Diego, CA. June 13, 2003.

[6] Lin, J., Keogh, E., Patel, P. & Lonardi, S. (2002). Finding Motifs in Time Series. In *the 2nd Workshop on Temporal Data Mining*, *the 8th ACM Int'l Conference on Knowledge Discovery and Data Mining*. Edmonton, Alberta, Canada. July 23-26, 2002.

[7] Ohsaki, M., Sato, Y., Yokoi, H. & Yamaguchi, T. (2003). A Rule Discovery Support System for Sequential Medical Data, in the Case Study of a Chronic Hepatitis Dataset. In *Discovery Challenge Workshop*, *the 14th ECML/the 7th PKDD*. Cavtat-Dubrovnik, Croatia. Sep 22-26, 2003.