Mining sensor data from complex systems

Vespier, U.

Chapter 7

Interactive Time-Series Visualization

When approaching a new data science problem, it is most of the time preferable to spend some time to get an idea of the properties and the features of the data at hand. This exploratory phase is not only useful to get a better understanding of the application domain, but it can provide fundamental insight about the data and inform all the subsequent modeling, feature construction and algorithm design choices. Moreover, freely looking at the data before diving into the actual modeling could spot fundamental issues on how it was collected and processed in the first stage, issues which could potentially render any derived analysis flawed if not pointless.

This preliminary exploration phase of a given dataset is widely known in the community under the name of *exploratory data analysis* (EDA) [92, 93]. Typical EDA activities include generating statistical summaries, computing aggregations, fitting distributions and, last but not least, visualizing the data in several ways in order to spot patterns and gain insight, mostly driven by the intuition of the practitioner.

Effective visualization, in fact, is one of the most powerful and immediate ways of analyzing a dataset, relying on the ability of the human brain to abstract, summarize, spot trends and anomalies through visual inspec-

tion [46].

Most data analysis software suites allow practitioners to perform visualization easily on moderately sized datasets. However, when the data at hand is too big to be handled by off-the-shelf software solutions, it becomes difficult to perform effective visualization and more advanced solutions, able to cope with big data, are required.

This chapter discusses visualization in the context of EDA of massive time series data. In particular we focus on the following problem:

> Given a massive time series dataset, how can we support interactive visualization, enabling fast browsing and zooming from coarser to finer levels of detail?

Note how the emphasis is put on the interactivity of the process as required by effective EDA. Although it is reasonably easy to visualize small datasets with current software suites, this simple task becomes quite challenging when dealing with large amount of data, especially when it is required to do this interactively. Throughout the chapter, we will see how this problem can be addressed by working on sampled versions of the original data, organized as a hierarchy.

The solution has been implemented and tested on a real-world scenario, the InfraWatch project, and resulted in a software package called VizTool, which will be introduced throughout the chapter.

The rest of the chapter is organized as follows. Section 7.1 introduces the concept of sub-sampling hierarchy which lays the foundation for the subsequent sections. Section 7.2 discusses how we exploit the sub-sampling hierarchy in order to support interactive visualization of massive time series data. Section 7.3 introduces VizTool, a web application implementing the concepts discussed in the chapter. Finally, Section 7.4 draws the conclusions.

7.1 Hierarchical Time Series Subsampling

When dealing with large univariate time series, there are mainly two properties that affect dimensionality: the extent of the measured period and its sampling frequency. By reducing the considered period, the sampling frequency, or both, one can directly reduce the amount of data to be processed at once, in some cases without losing relevant information for the task at hand.

Take, for example, the case of InfraWatch where data from sensors is sampled at 100 Hz. If we are interested in spotting seasonal trends in strain measurements, it suffices to consider a moderately sub-sampled version of the data, possibly containing one measurement every hour or, even, every day.

In general this means that, depending on the task at hand, it is often enough to consider a sub-sampled version of the data in order to speed up calculations and support interactivity if required.

In this section, we propose a storage scheme for large time series based on progressively sub-sampled versions of the original data to support this idea.

7.1.1 Sub-sampling Hierarchy Construction

The problem of effectively reducing the dimensionality of a time series, interpreted as its number of data points, boils down to producing a reduced representation such that it resembles the original data as much as possible. From now on, and without loss of generality, we will assume we are dealing with constant rate time series[1].

The most trivial way to reduce the dimensionality of such a time series is to consider every nth point, thus reducing its size by a factor n. This approach, however, has a drawback as it is too sensitive to outliers. In fact, rare events, such as spikes in the data or errors in the measurements, could

[1] It is always possible to add or remove points of a time series, by interpolation, to make its sampling rate constant.

be selected in the sampling, rendering the resulting approximation, and its shape, skewed.

A more robust way of producing a low dimensionality approximation is by taking the average of the points to be aggregated. More formally, given a time series $\mathbf{x}$ of length n, we want to compute an approximation of length M

$$A(\mathbf{x}, M) = \hat{\mathbf{x}} = \hat{x}[1], \ldots, \hat{x}[M]$$

where:

$$\hat{x}[i] = \frac{M}{n} \sum_{j=n/M(i-1)+1}^{(n/M)i} x[j] \ .$$

This method of reducing the dimensionality of a time series is also at the base of a well known segmentation technique called Piecewise Aggregate Approximation [47]. In the segmentation task, however, the final outcome is not a time series with a reduced number of points (a lower sampling rate) but a cheaper representation of the same data, obtained by reducing segmented data points to horizontal lines centered around their mean.

The average function, however, is not the only possible choice for aggregating time series data. Ideally, the choice of an aggregation function should be based on its ability to preserve the perceptual features of the data, although this ability may depend on the properties of the data itself and how it evolves over time. A review of several possible aggregation functions and analysis of how well they cope with the task of visualization can be found in [6].

We also note that even basic aggregation functions can be of practical importance when dealing with time series data, especially for visualization purposes. Consider, for example, the case of temperature sensors where one is often interested in knowing what are the changes in temperature at any given time frame. In such cases, computing multiple types of aggregations, such as minimum, maximum and average, can support *band visualizations*.

Having a way to reduce the length of a time series, we can now define a hierarchical storage scheme which materializes different levels of approximation given an input time series.

Definition (Hierarchical Storage Scheme). Given a time series $\mathbf{x}$ of length n, a hierarchical storage scheme is defined by materializing a sequence of $log_2 n + 1$ consecutively coarser approximations $A(\mathbf{x}, n/2^i)$, for $0 \leq i \leq log_2 n$.

Note that we apply the floor operation on $log_2 n$ to handle time lengths that are not strictly a power of two.

In other words, we store consecutively coarser versions of the original time series, where every version has half the size of the immediately finer one. This approach creates a pyramid of approximations which, when fully stored on disk, permits to quickly retrieve portions of a time series at a resolution that is as close as possible to the desired one. Figure 7.1 depicts this concept visually.

We note that, for any given time series $\mathbf{x}$ of length n, the total number of data points of all levels of approximation in the storage scheme is $2n$. As the storage capacity quickly increases over time and its price quickly drops, a twofold increase of the required storage represents a good compromise, especially considering the huge gain in retrieval speed provided by this approach for visualization purposes.

The concept of considering data stored at different resolutions and complexities in order to speed up retrieval and rendering is also at the base of many computer graphics methods. In texture mapping [39], for example, a technique called mipmapping [102] involves pre-calculating sequences of progressively lower resolution versions of the same texture image, each of which is half the size of the previous one. The approach permits to reduce render time and reduce artefacts, such as aliasing, by choosing the right level of resolution depending on the pixel density of the object. Objects closer to the camera will be rendered with high resolution textures while, on the contrary, for distant objects, using less-defined texture images will suffice.

The same concept, again in computer graphics, is behind LOD (level of detail) based 3D rendering [66], where the polygonal complexity of an object is lowered, by employing polygonal reduction algorithms [65], as it moves away from camera. Although more advanced continuous LOD methods exist, basic (discrete) LOD approaches are based on storing pre-computed polygonal

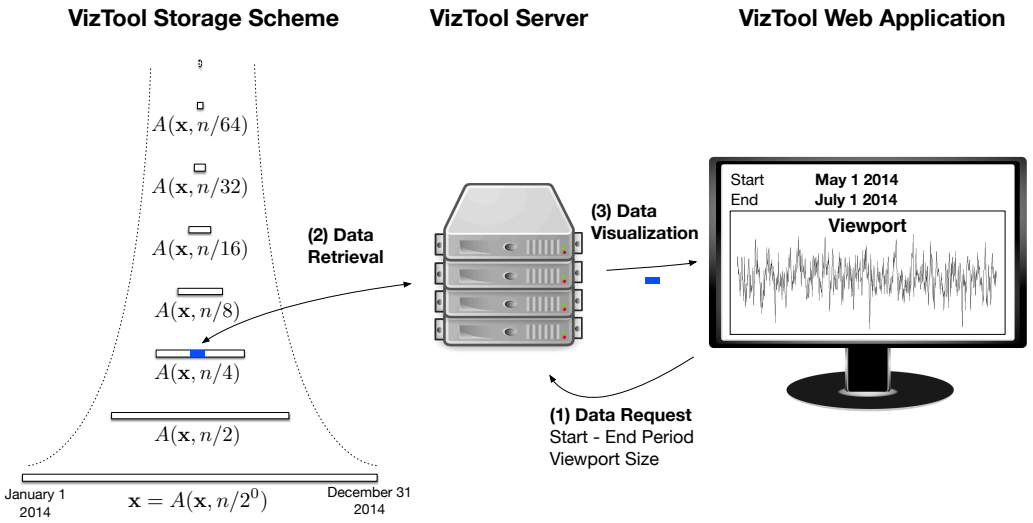

Figure 7.1: The three main components of VizTool. On the left, it is shown the pyramidal storage scheme for time series introduced in 7.1.1. A schematic example of a VizTool user session is shown on the right side. The web application makes a data retrieval request to the VizTool server providing the desired period and the size of the viewport. Given these parameters, the server accesses the best resolution level from the storage and sends the approximated data back to the web application for visualization.

representations of objects at different resolutions.

7.2 Interactive Visualization

We will now show how the time series storage scheme introduced in Section 7.1 can be leveraged to support interactive visualization of large time series.

First, let us consider a typical time series visualization scenario. The following parameters affects the produced visualization at a given time:

- the requested time period (t_{start}, t_{end}),
- the sampling rate of the original data,

- the width in pixels of the visualization viewport w.

Any given configuration of time period and sampling rate results in a number of data points, from the original time series, to be visualized. As this number of points can potentially exceed the pixel width of the viewport, some sort of aggregation has to take place. Performing such an aggregation task on the fly, starting from the original time series, can be an expensive process and could severely harm the interactivity of the visualization.

In order to speed up the process above, we can employ the storage scheme introduced in Section 7.1.1 to directly retrieve the data from the most suitable aggregation level, without performing on the fly expensive operations.

Ideally, given a configuration of time period and sampling rate, we would like to retrieve a number of points that takes into account the size of the viewport in order to reduce the amount of data to be aggregated at render time.

Given a time series $\mathbf{x}[t_{start}, t_{end}]$ to be visualized, let $R = (t_{end} - t_{start})/w$ be the ratio between the requested number of points in the original time series and the viewport width. The value of R indicates how many data points per pixel are to be shown using the original time series. We can reduce this factor by retrieving the data from the right approximation level in the storage scheme. More formally, the best option is to retrieve the data at level $A(\mathbf{x}, n/2^k)$ where $k = \lfloor log_2 R \rfloor$.

This approach links the size of selected data to the resolution of the screen by retrieving the largest aggregation level that has enough data points for the current viewport. One clear advantage is that data retrieval time is reduced as the complexity of the visualization is now dependent on the actual resolution of the screen used.

7.3 VizTool Software

The storage scheme and the interactive visualization method introduced in the previous sections are the core concepts behind VizTool, a visualization

software for large time series data. VizTool has been developed in the context of the InfraWatch project to aid interactive visualization of the bridge's data and facilitate the discussions with the domain experts.

The software has been designed with the following goals in mind:

- fast and interactive visualization of large time series data collected from sensors,
- ability to adapt the details in the visualization to the actual viewport size,
- possibility of comparing data from multiple sensors by stacking multiple time series line charts,
- support for band visualization based on minimum, maximum and average aggregation functions,
- ability to export any portion of the data at any given sampling rate,
- ability to bookmark and add notes to portions of the data to support data annotation activities,
- possibility to compute correlations between any chosen set of sensors.

The second goal, in particular, was fundamental to run VizTool effectively on large monitors or multi-screen setup when discussing data and presenting projects results.

Moreover, because of the sensitiveness and size of InfraWatch's data, VizTool's architecture is based on a client-server model which permits to host all the data on the server side, while allowing client hosts to browse and export portions of it.

VizTool's server-side application has been developed in Python [77] using the web framework Django [23] and HDF5 [38] as data storage library to support effective and efficient caching of data from disk into memory. On the client-side, the web application is written in pure Javascript and HTML/CSS employing the Highcharts [40] graphing library for plotting the time series.

VizTool has been used to visualize InfraWatch's data at scale, allowing one

Figure 7.2: The VizTool web application interface.

to browse and inspect a terabyte sized dataset of sensor time series sampled at 100 Hz for a period of three months.

An example of a VizTool session is shown in Figure 7.2. Figure 7.3 shows VizTool running on a multi-screen setup.

7.4 Conclusions

In this chapter, we introduced the task of large time series visualization in the context of Exploratory Data Analysis (EDA) and we discussed the challenges linked to effective interactive visualization of big data.

As we observed that the amount of visible data points is always limited by

the size of the visualization viewport, we proposed a storage scheme to hold sub-sampled versions of the original data in order to speed up data retrieval at different levels of resolution.

We introduced a data retrieval mechanism for visualization based on such storage scheme and presented VizTool, a software solution that leverages these concepts to support fast and interactive visualization of large time series data collected from sensors.

VizTool proved to be an effective tool for the practical exploration of the InfraWatch data, not only supporting the EDA process but also serving as a practical demonstration for public exposition of the project's results.

VizTool was also instrumental in discovering important properties and events in the data such as dead sensors, re-calibration activities, correlation between temperature and strain response, differences of traffic activity between work days and weekend days. Moreover, VizTool made even more evident the multi-scale nature of InfraWatch data and how different scales (monthly, daily, hourly) reveal new complex phenomena and events inherent to certain time resolutions.

Future work includes the extension of the VizTool software to support on the fly operations that leverage the storage scheme we presented. For example, it would be possible to compute approximate correlations between time series in an anytime fashion by starting the computation at the coarsest level, progressively refining the results while moving to finer levels.

The same concept could be applied to motif discovery. Anytime approximate motif discovery could be implemented by exploiting the sub-sampling hierarchy. A naive solution would involve running the exact MK algorithm [74] at each level of the hierarchy, from coarser to finer resolutions, and presenting intermediate results to the user.

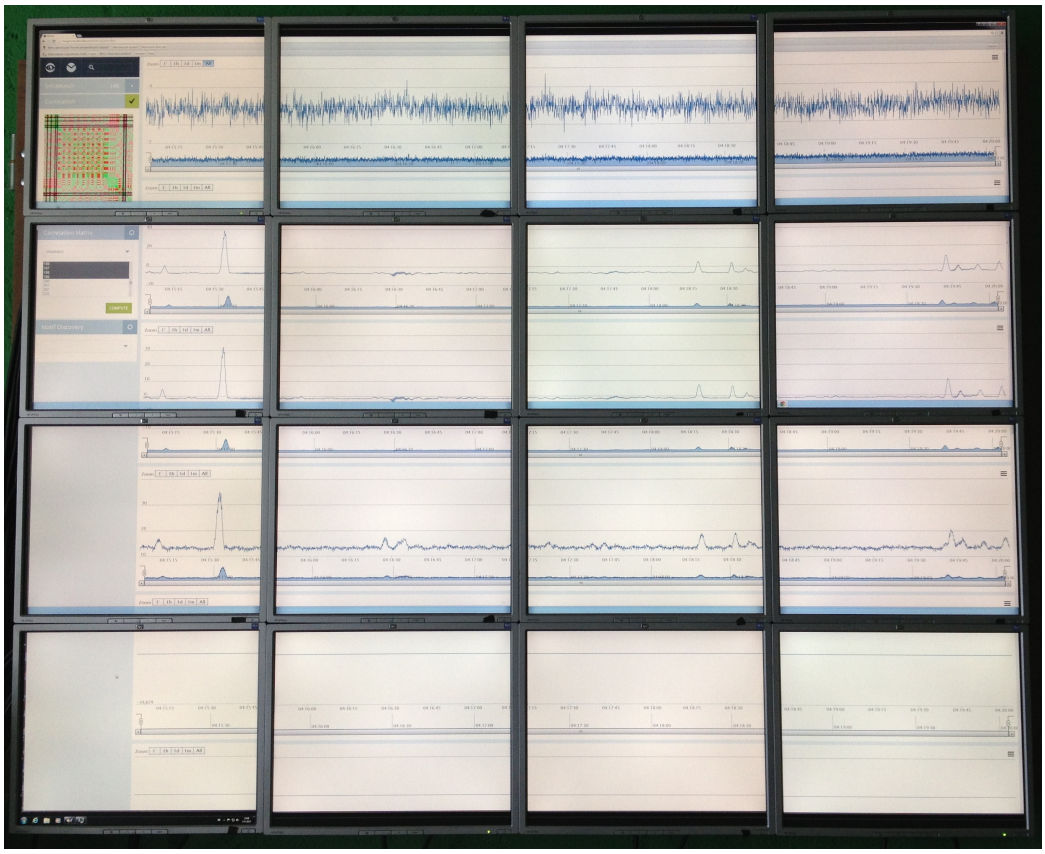

Figure 7.3: VizTool running on a multi-screen setup at the Leiden Institute for Advanced Computer Science. The setup mounted a total of 16 screens (4 by 4) for a total resolution of 5120x4096.