TIME SERIES COMPRESSION FOR ADAPTIVE CHART GENERATION

Giuseppe Burtini, Scott Fazackerley, Ramon Lawrence

University of British Columbia
g.burtini@alumni.ubc.ca, scott.fazackerley@alumni.ubc.ca, ramon.lawrence@ubc.ca

ABSTRACT

Web content is consumed on smart phones, tablets, and computers with a significant variation in device display resolution. Visualizing data is typically performed by extracting data from a database, packaging it as JSON or XML for transmission to the client and then visualizing with a client-side JavaScript library. A major challenge is to retrieve only the required data for visualization. Current approaches require programmers to manually modify their data extraction queries and do not adapt to client display characteristics. The contribution in this work is a configurable data compression method that automatically adapts the amount of data transmitted for client-side visualization based on device characteristics. We evaluate several different techniques for time series summarization and compression and show that the amount of data transmitted can be reduced by between 40% and 80% while preserving pixel-perfect visualization. Reducing data transmitted improves client responsiveness and allows flexible and responsive web content without programmer intervention. The approach is tested on a wide variety of data sets and is implemented as an add-on for a JavaScript visualization library.

Index Terms— data visualization, user interfaces, data compression, optimization, time series analysis

1. INTRODUCTION

With the proliferation of web devices, designers are challenged to construct web sites with content that is adaptive to the characteristics of each device, especially display size and resolution. Client-side libraries [1] allow content to be scaled and modified for best display. However, there has been considerably less work on how to adapt the amount of data provided to the web client. Displaying data is a common task for applications, especially those involving time series data [2].

In these domains, the amount of data to be visualized is often considerably larger than can be displayed, and transmitting all data to the client is costly. The amount of data visualized on a computer with a 27” monitor is different than a smart phone. Retrieving too much data is costly in time and network bandwidth and requires the visualization library perform sampling or averaging. Retrieving too little data may result in the visualization missing key points critical to data analysis. Sending the right amount of data for visualization improves performance and reduces the network bandwidth used. From a developer's perspective, the ideal case is for the server to automatically adapt the amount of data sent to the resolution of the client device.

This work demonstrates a system that takes a time series database, performs various compression algorithms on the data, and uses the compressed summaries to only transmit the necessary data to the client for visualization. There has been considerable work in summarizing and compressing time series data. The focus of this work is not on the particular compression technique chosen, but rather demonstrating how these techniques can be used in conjunction with knowledge of web client display sizes to dramatically reduce the amount of data transmitted for visualization. Our system has been tested on several standard data sets [3]. On average, the amount of reduction on data transmitted is 42%. For smart phones, the reduction is a substantial 77% which is especially significant due to the bandwidth costs on mobile networks.

The organization of this paper is as follows. Section 2 presents background on the compression, visualization and general treatment of time series data. Section 3 provides an overview of the approach including a system architecture. Section 4 describes our implementation of sliding window, piecewise polynomial approximation (PPA) for time series compression. Experimental results are in Section 5, and the paper closes with future work and conclusions.

2. BACKGROUND

A standard solution to the problem of presenting large data is aggregation. For example, if the data consists of sensor readings sampled every 5 seconds, then a chart displaying the data over a year will typically aggregate the samples on a daily basis. The issue is that this aggregation is usually predefined and does not adapt to the display. A larger display can present more data and should not be limited *a priori* on the data that it displays. Conversely, a smart phone display that attempts to display a chart with data better suited to a larger display will end up discarding data that does not fit on its screen or overplotting the data making the visualization difficult to understand. In a web environment, the transmission of the data to the client is costly and time-consuming.

Time series data [2] is one of the most common types of visualized data and is present in many domains including economics, environmental and industrial sensors, and network and computer performance monitoring. Time series data can be very large making efficient visualization and processing critical. Time series summarization or compression can be used for a variety of purposes including time series indexing [4], clustering [5], and querying or pattern matching [6]. Summarization techniques also have the major benefit of producing a summary of the data set that is significantly smaller than the original data set. There are numerous techniques for time series compression including piecewise linear approximation [7], piecewise polynomial approximation [8], Discrete Fourier Transforms [9], Discrete Wavelet Transforms [10], using symbolic encodings [11] and adaptively selecting the best model for segments of a time series [12]. In this paper, we implement a piecewise polynomial approximation approach similar to [13], although other compression techniques may be used.

The closest related work defines perceptually important points (PIPs) [14] to build a system that only transmits the most visually important points to a client. PIPs are an implementation of the well-known line simplification algorithm first proposed in [15]. The algorithm takes a set of n polygonal points $t_1, t_2, ..., t_n$ and approximates the set by the line segment $\overline{t_1 t_n}$. It then finds the farthest point t_i from $\overline{t_1 t_n}$, and if this distance is larger than an error tolerance ϵ recursively approximates the subchains $t_1, t_2, ..., t_i$ and $t_i, ..., t_n$. Although the implementation described in [14] mentions adapting the number of PIPs to display resolution, it does not define a practical stopping condition (ϵ).

Despite the ubiquitous need for web data visualization, there is no prior work or system that adapts data transfer to the device characteristics automatically.

2.1. Definitions

Consider a time series T of n points $t_1, t_2, ..., t_n$. Assume this time series will be visualized on a line chart with vertical height h pixels and horizontal width w pixels. If $n \leq w$, the chart can display the time series with at least one pixel per data point. If $n > w$, then there are more data points than can be displayed and aggregation is required, otherwise overplotting effects will be visible.

Aggregation reduces data series by averaging values in intervals, $d_i = \frac{1}{x}\sum_{j=(i-1)*x+1}^{i*x} t_j$ for $i = 1..\lceil \frac{n}{x} \rceil$. By controlling the parameter x, the time series can be aggregated and reduced in size by any factor. For a display of width w pixels and time series of n points, the minimum $x = \lceil n/w \rceil$.

Consider an example data set consisting of $n = 10$ points $\{5, 7, 9, 7, 5, 5, 1, 9, 3, 5\}$. If $w = 5$, then aggregation will display $\{6, 8, 5, 5, 4\}$. As shown in Fig. 1, aggregation loses detail information. Without any aggregation, displaying the 10 data points in 5 pixels results in an overplotted figure as there is more data points than pixels to display them.

The goal is, given any time series T and display dimensions w and h, transfer the smallest representation of the time series that has identical visual characteristics to aggregation with $x = \lceil n/w \rceil$.

The vertical resolution V is defined as the size of each pixel in the vertical dimension in terms of the data such that

$$V = \frac{\max(t_i) - \min(t_i)}{h} \tag{1}$$

where $i = 1..n$. For the previous example, if the height of the chart is 100 pixels, $V = \frac{9-1}{100} = 0.08$. Each pixel represents a value range of 0.08.

A model m is a representation of a time series with values $m_1, m_2, ..., m_n$. The absolute error is defined as the sum of absolute differences between the data and the model with

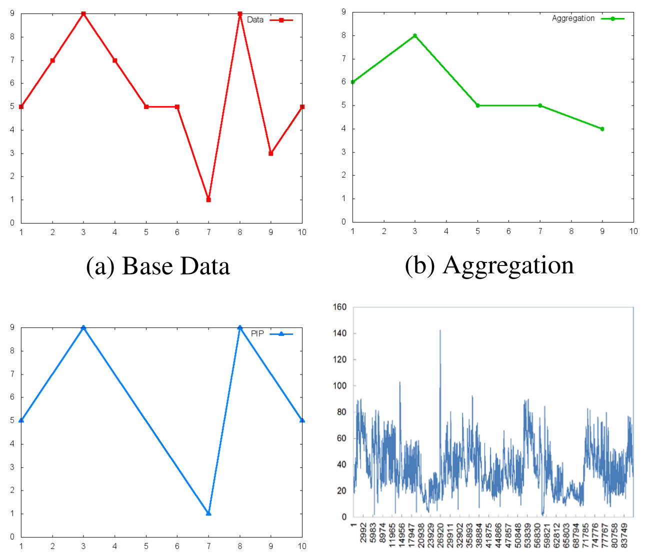
$$AE = \sum_{i=1}^{n} |m_i - t_i|. \tag{2}$$

Visual error for model m is defined as the sum of the number of differing pixels between the resulting visualization compared to the visualization achieved by aggregation with $x = \lceil n/w \rceil$. Visually relevant error is defined as

$$VRE = \sum_{i=1}^{n} \lfloor \frac{|m_i - d_{\lfloor (i-1)/x \rfloor + 1}|}{V} \rfloor. \tag{3}$$

With our example, absolute error on aggregation with $x = 2$ is 12. Visual error is defined to be 0 as aggregation is the baseline model. We are interested in algorithms that perform lossless summarization in terms of visual error. Performing PIP with $w = 5$ results in the five PIPs: $\{(1, 5), (3, 9), (7, 1), (8, 9), (10, 5)\}$ with absolute error of 6 and visual error compared to aggregation of 3.

Assume the example 10 data point set is part of a larger set such that for visualization only 5 pixels are available to display the data ($w = 5$). A system that sends the entire data set to the client for visualization is leaving it up to the visualization library to handle the issue. The library may perform sampling, aggregation, or overplotting, so developers will perform aggregation on the data to ensure it can be visualized properly and consistently. The minimum amount of aggregation is averaging two points as shown in Fig. 1(b). Aggregation loses some information (e.g. big change between times 7 and 8) which is measurable by absolute error. However, it is the best that is possible given the display characteristics. In Fig. 1(c) is the output for PIP which models the base data series with no aggregation but may still be subject to overplotting (e.g. points 7 to 8 will occupy the same pixel which would leave it up to the visualization library to resolve). PIP produces visually different charts compared to standard aggregation. In Fig. 1(d) is an example of overplotting where too much data is attempted to be displayed.



(a) Base Data (b) Aggregation

(c) PIP (d) Overplotting

Fig. 1: Time Series Summarization

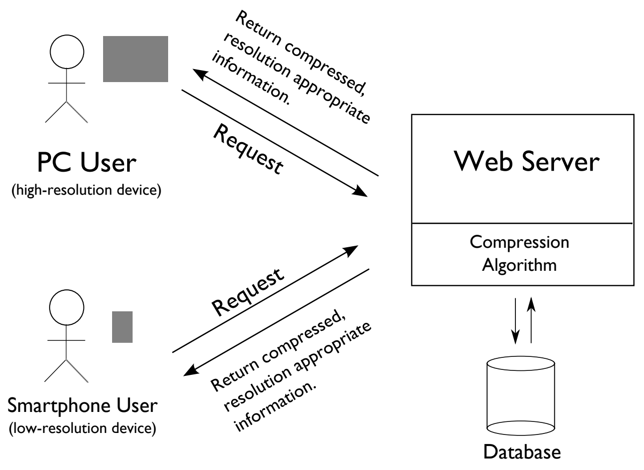


Fig. 2: High-Level Architecture Diagram

3. OVERVIEW OF THE APPROACH

An architecture diagram for the approach is in Fig. 2. When a user requests a web page containing a visualization of time series data, as part of that request the browser provides information on the client environment, including display resolutions. The web server receives that request, and the module responsible for retrieving the time series data queries and retrieves the data from the database. Instead of passing the data directly back to the client, the data is passed through a compression module that takes the raw time series data as input and outputs a compressed data stream specific for the client request. Thus, a request by a smart phone user will receive a different compressed stream than a request by a PC user due to the different visualization properties of each device. For efficiency, the system also caches previous requests so that multiple users requesting the same time series at the same resolution can retrieve the data directly from cache rather than extracting it from the database and compressing it. When the client receives the compressed data, a small JavaScript routine uncompresses the data before passing it to the visualization library. The system is transparent to the visualization library and functions above the database layer.

The key benefit of the approach is that the web server is often able to transmit significantly less data to the client to achieve an equivalent visualization compared to transmitting the whole data set. This saves bandwidth which reduces the time and cost for the user and the service provider.

The compression algorithm relies on device characteristics (width and height of the display). Utilizing these characteristics allows for better compression while still remaining lossless. We have experimented with several compression algorithms including: no compression, run-length encoding (RLE), visual aggregation, perceptually important points (PIP), piecewise linear approximation (PLA), and piecewise polynomial approximation (PPA). Implementation details of these algorithms follow.

3.1. No Compression and RLE

Both the no compression and run-length encoding (RLE) algorithms provide the complete set of data to the client in lossless fashion. If there are more data points than can be displayed, the visualization library is left to handle this issue which typically results in overplotting and unreadable charts. In practice, fixed aggregation is performed before data visualization to prevent these issues. Note that all algorithms could further compress the data using a Lempel-Ziv type algorithm [16]. This type of compression is typically performed at the HTTP level using DEFLATE or gzip and is not considered in this work.

3.2. Visual Aggregation

Aggregation takes time series data and aggregates it over the time dimension to produce a smaller data set. Aggregation is commonly performed statically by averaging over time to reduce the data points. Our implementation performs dynamic, visual aggregation. Given the display width w and time series of length n, the time series is aggregated by averaging $\lceil n/w \rceil$ adjacent points. This guarantees one point per pixel in the visualization (the minimum possible) and maximizes the information displayed.

3.3. Visual RLE

A natural extension to RLE is to perform RLE by considering points equivalent if they are within a given error bound ϵ. This would allow points 1.1, 1, 0.9, 1 to be considered equivalent with an error bound of 0.1. Thus, each pixel represents a possible range of values from $[X-R/h, X+R/h]$. By setting $\epsilon = V$, as defined in Equation (1), considerable compression may be achieved as similar points may be deemed equivalent in terms of visual representation. Visual RLE produces an output that is visually equivalent to visual aggregation.

3.4. Perceptually Important Points (PIPs)

The work on PIPs [14] is one of the only works to consider summarizing a time series data by its visual characteristics. As explained in Section 2, PIPs are determined iteratively by calculating the next point that is furthest from the current representation. Although [14] indicates that PIPs can be used to adapt to display characteristics, the authors do not state specifically how this is achieved in the algorithm. The major drawback is that there is no stopping criteria defined for PIPs. Our implementation provides a parameter n denoting how many PIPs to use for data summary. The algorithm can also continually calculate PIPs until a given error bound ϵ is hit. Unlike other algorithms, the output of PIP is not guaranteed to provide the same visual representation as aggregation.

4. PIECEWISE POLYNOMIAL APPROXIMATION

Our implementation combines piecewise linear and polynomial approximation and functional compression. The algorithm models the data using a series of piecewise linear and polynomial functions [17] as shown in Algorithm 1. To construct each piecewise segment, functional compression is performed on the data. A fitting function performs a linear scan through the data with the goal of fitting a piecewise function of a functional form F. Fitting continues until the stopping condition where the error exceeds ϵ. The algorithm then stores the last good fit as a piecewise section for the model and continues to construct a new piecewise segment starting from the point where the previous segment failed to meet the stopping condition. During the fitting process for a single segment, if it is not possible to fit F with a given degree n, the code backtracks and attempts to fit a degree $n-1$ function. This process continues until reaching the base case where F is a singular data point itself to ensure that no point is left unfitted.

Using the example data set $\{5,7,9,7,5,5,1,9,3,5\}$ PPA would iteratively perform a series of least squares fits, producing a new fit each time the error threshold is exceeded. For pixel-perfect modeling, the error threshold $\epsilon = V$ from Equation 1. The resulting model is

$$f(t) = \begin{cases} 0.6 + 5.14t - 0.86t^2 & t \leq 5 \\ 281 - 82t + 6t^2 & 5 < t \leq 8 \\ -24.5 + 7.7t - 0.5t^2 & 8 < t \end{cases} \quad (4)$$

Once the algorithm has constructed the piecewise model where the encoding specifies the function degree and coefficients for each piece, the model is transmitted to the client. The piecewise model is then evaluated at an interval based on client resolution requirements to determine y-axis values.

Algorithm 1 PPA(Series T, Form F, Threshold ϵ)

```
data = [], pieces = [], model = null
previousModel = null, pieceCounter = 0
for $i = 0$; $i <$ T.length; $i++$ do
    data.append($t_i$)
    previousModel = model
    model = fit(data, F)
    error = calculateError(model, data)
    if error $\geq \epsilon$ then
        pieces[pieceCounter] = model
        pieceCounter += 1
        $i$ -= 1
        data = []
    end if
end for
return pieces
```

5. EXPERIMENTAL RESULTS

It is well-known that algorithms have widely different performance on different data sets. The data sets tested are listed in Table 1. Data sets flagged as "subset" have been reduced into smaller chunks so that visualization makes sense. Trying to visualize very large data sets just results in a lot of aggregation. The data sets chosen have between 500 and 150,000 data points of one dimensional data. The chart resolution was varied between common device sizes starting from 100x100 (thumbnail charts) to 1440x960 (HD laptop screen). The algorithms are evaluated based on their absolute error, visually relevant error, and size of data transferred in bytes. The raw time series data has a size of 4 bytes per entry (size of an integer). Compression routines like PIP also must store the time dimension so each PIP entry has 8 bytes. Functional compression using degree 2 polynomials uses 16 bytes per functional piece.

Fig. 3 shows the absolute error of each encoding technique on various screen sizes. The numbers in brackets in the legend indicate the average relative data size for each algorithm, where a relative data size of 1 indicates the size of the data itself. The absolute error is the difference between the time series data and the model representation of the time series. Techniques such as sending the entire data set and RLE by definition have no absolute error as they send the entire data set. PIP has a very high absolute error until it sends all the data points. Functional compression has considerably less error.

Even more important than absolute error is visually relevant error shown in Table 2 as VRE. Visually relevant error occurs due to inaccurate data caused by summarization. We have modified the selected algorithms to exit when there is no visually relevant error, so the interesting variable to compare is the relative data size, that is, the amount of data required

Table 1: Experimental data sets.

Data Set	Size	Source
CPI Seasonal Data	787	[18]
Minutely Soil Moisture Readings	136 503	[19]
Variable Star Magnitude	600	[20]
Artificial AR(1) Process	500	[21]
Dow Jones Industrial Avg. Closing	641	[21]
Central England Temp. (1659-1989)	3 972	[22]
Keogh: fortune500.dat	128 499	[3]
Keogh: slips.dat (subset)	2 500	[3]
Keogh: katafu.dat (subset)	5 000	[3]
Keogh: stocks-n6480.dat (subset-1)	10 000	[3]
Keogh: stocks-n6480.dat (subset-2)	10 000	[3]
Child EEG Readings	49 999	[3]

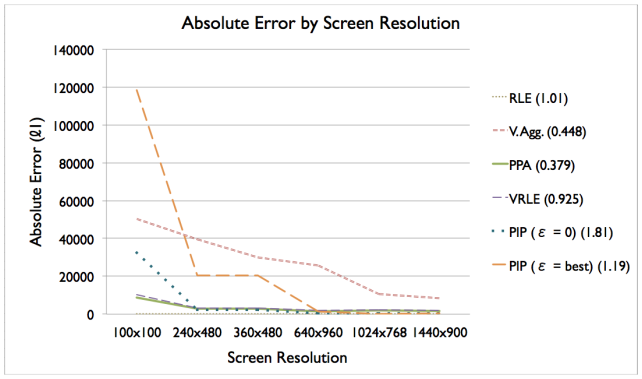


Fig. 3: Absolute Error by Screen Resolution

to achieve no visually relevant error with a given algorithm. Aggregation performing one value per pixel has visual error of zero. All of the functional compression techniques have a visual error of zero as the functional summaries are designed to match the aggregated values. Only aggregation and functional compression techniques allow for pixel-perfect display that adapts to the screen resolution.

The goal is to allow adaptive pixel-perfect charts with minimal data transfer. In Fig. 4 is the relative amount of data transferred by the different techniques. The relative data transferred is shown with the entire data set being 1. RLE compression has little savings, although visual RLE does provide very good performance as values equivalent in the vertical dimension to the error bound of a pixel can be considered equivalent. PIPs transfers more bytes than the data set itself. Although it is possible to send fewer PIPs than the horizontal resolution as recommended by [14], the amount of absolute and visual error is extreme which renders the visualization useless. Functional compression sends, on average, 62% less data than the entire data set and adapts to the screen resolution. As expected there is significant savings in data transmitted for lower resolutions. The compression can be performed in less than a second for real-time web visualization.

Table 2: Average results for a representative set of algorithms over all the test data sets and resolutions.

Algorithm	Relative Data	Avg. AE	Avg. VRE
RLE	1.01	0	0%
V. Agg.	0.448	27 366	0%
PIP ($\epsilon = 0$)	1.808	6 373	0%
PIP ($\epsilon = best$)	1.185	26 767	3.77%
VRLE	0.925	3 785	0%
PPA	0.379	3 169	0%

Functional compression utilizing Piecewise Polynomial Approximation has particularly strong performance in data which has a predictable underlying structure, trends or functional form. As an example, consider the econometric data presented in the test sets: CPI Seasonal (monthly, inflation data) and the Dow Jones Industrial Average (monthly, stock data): both data sets have an upward trend that occurs at a relatively consistent rate. Functional compression with PPA at any resolution provides a substantial savings over any of the other methods on these datasets even where visual aggregation does not. For these two data sets in particular, PPA performs an average of 57% better than visual aggregation, while other methods perform no better than average.

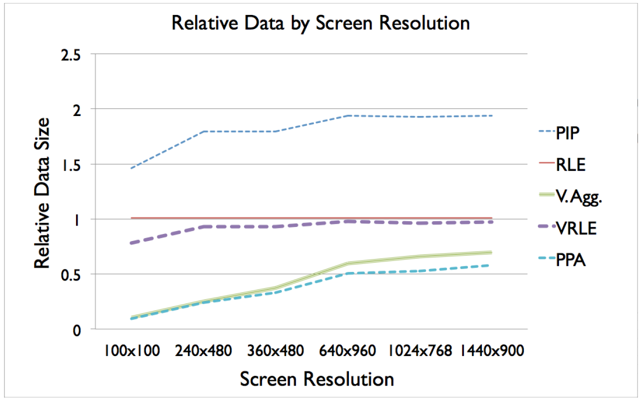


Fig. 4: Relative Data Size by Screen Resolution

6. CONCLUSIONS

Visualization of time series data is very common in web applications. Performance of the visualization is primarily based on the amount of data sent from the server to the client. Although applying static aggregation in the time dimension is a common technique to reduce the data transferred, it does not adapt to device characteristics. Consequently, smaller devices may receive more data than required, resulting in increased bandwidth usage and the visualization library overplotting or sampling the data sent. In this work, we demonstrated a technique that uses knowledge of the device characteristics as input for time series compression algorithms, in order to reduce the amount of data transferred while at the same time achieving pixel-perfect visualizations. Across a wide variety of different time series and screen resolutions, the size-weighted average savings is 42%. Future work will examine low/high-pass style filtering to extract structural components, find appropriate functional forms for specific data generating processes or apply a PPA-style technique to higher dimensional data.

7. REFERENCES

[1] Vadim Ogievetsky and Jeffrey Heer, “D^3 data-driven documents,” *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, 2011.

[2] Eamonn Keogh and Shruti Kasetty, “On the need for time series data mining benchmarks: A survey and empirical demonstration,” *Data Mining and Knowledge Discovery*, vol. 7, pp. 349–371, 2003.

[3] E. Keogh, Q. Zhu, B. Hu, Hao. Y., X. Xi, L. Wei, and C. A. Ratanamahatana, “The UCR Time Series Classification/Clustering Homepage: `www.cs.ucr.edu/~eamonn/time_series_data/`,” 2011.

[4] E. Fink, K.B. Pratt, and H.S. Gandhi, “Indexing of time series by major minima and maxima,” in *IEEE International Conference on Systems, Man and Cybernetics*, 2003, pp. 2332–2335.

[5] T. Warren Liao, “Clustering of time series data - a survey,” *Pattern Recognition*, vol. 38, no. 11, pp. 1857 – 1874, 2005.

[6] Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh, “Searching and mining trillions of time series subsequences under dynamic time warping,” in *KDD*, 2012, pp. 262–270.

[7] Manjula A. Iyer, Morgan M. Harris, Layne T. Watson, and Michael W. Berry, “A performance comparison of piecewise linear estimation methods,” in *Proceedings of the 2008 Spring Simulation Multiconference*, 2008, pp. 273–278.

[8] Erich Fuchs, Thiemo Gruber, Jiri Nitschke, and Bernhard Sick, “Online segmentation of time series based on polynomial least-squares approximations.,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2232–45, Dec. 2010.

[9] Kelvin Chu and Man Hon Wong, “Fast time-series searching with scaling and shifting,” in *PODS*, 1999, pp. 237–248.

[10] Ivan Popivanov and Renée J. Miller, “Similarity search over time-series data using wavelets,” in *ICDE*, 2002, pp. 212–221.

[11] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Jeffrey P. Lankford, and Donna M. Nystrom, “Visually mining and monitoring massive time series,” in *ACM SIGKDD*, 2004, pp. 460–469.

[12] Tian Guo, Zhixian Yan, and Karl Aberer, “An adaptive approach for online segmentation of multi-dimensional mobile data,” in *ACM MobiDE*, 2012, pp. 7–14.

[13] Zhenghua Xu, Rui Zhang, Kotagiri Ramamohanarao, and Udaya Parampalli, “An adaptive algorithm for online time series segmentation with error bound guarantee,” in *EDBT*, 2012, pp. 192–203.

[14] Tak-Chung Fu, Korris Fu-Lai Chung, Chun-Fai Lam, Robert Wing Pong Luk, and Chak man Ng, “Adaptive data delivery framework for financial time series visualization,” in *ICMB*, 2005, pp. 267–273.

[15] D. H. Douglas and T. K. Peucker, “Algorithms for the reduction of the number of points required to represent a line or its caricature,” *The Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, 1973.

[16] Jacob Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” *IEEE Transactions on Information Theory*, vol. IT-24, 1978.

[17] Jacques Carette, “A canonical form for piecewise defined functions,” in *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, New York, NY, USA, 2007, ISSAC, pp. 77–84, ACM.

[18] Bureau of Labour Statistics, “CPI Seasonally Adjusted, Monthly, 1913-2012,” `http://www.bls.gov/cpi/`, 2012.

[19] Scott Fazackerley and Ramon Lawrence, “Reducing turfgrass water consumption using sensor nodes and an adaptive irrigation controller,” *IEEE Sensors Applications Symposium*, pp. 90–94, 2010.

[20] E. Whitaker and G. Robinson, *The Calculus of Observations*, p. 341, Blackie and Son Ltd., 1941.

[21] The University of York, Math Department, ,” `http://www.york.ac.uk/depts/maths/data/ts/`, 2004.

[22] G. Manley, “Central england temperatures: monthly means 1659 to 1973,” *Quarterly Journal of the Royal Meteorological Society*, pp. 389–405, 1974.