A Visual Analytical Approach for Model Editing and Testing in Glyph-based Time Series Compression

Eamonn Maguire, Student Member, IEEE, Susanna-Assunta Sansone, and Min Chen, Member, IEEE

Abstract— Time series data is ubiquitous in the present world. In many applications, e.g., finance, meteorology, biology, chemistry, physics and engineering, such data is constantly being recorded. It is highly desirable to speed up the process of viewing time series data by exploring new visual representations to complement the conventional line graph that was invented over a millennium ago. One existing methodology is to detect Frequent Long Patterns (FLPs) and replace them with short and abstract notations, such as glyphs. In many applications, the deployment of this methodology has encountered the familiar problem of separating noise from anomalous features. On the one hand, the detection algorithm needs to tolerate a fair amount of noise to facilitate a high compression ratio. On the other hand, the detection algorithm needs to avoid any false positive at all cost. In this paper, we propose a visual analytics approach to address this paradoxical problem. We first use the conventional FLP detection algorithm to create a rough model that places its emphasis on noise tolerance. This initial model is then used to identify a set of FLPs in a corpus of time series, which serve as the testing results of the model. We transform this set of test FLPs to a parameter space, allowing for in detail analysis using parallel coordinates and network plots. Interactive visualization enables users to identify as many false positive results as possible, and more importantly, to identify features that can be used to filter them out. The brushing interaction with the parallel coordinates is then used to refine the initial model by activating feature-based filtering instructions in a model editing window. This approach can be iteratively applied to a model being developed. We have implemented a prototype system to demonstrate this novel approach.

1 Introduction

In many applications, such as finance, meteorology, biology, chemistry, physics and engineering, users have collected a large volume of time series data. A *time series corpus* $\mathscr{T}$ is a collection of time series that record phenomena of a similar nature. On the one hand, users often find that it is time consuming and cognitively demanding to browse many time series in a time series corpus. On the other hand, the corpus provides an opportunity to identify frequently occurring patterns. When such patterns are of a reasonable length, it is advantageous to replace them with a short abstract representation, such as a text label, a signature pattern, a glyph or a combination of these (e.g., [18]). This is a form of *visual compression*, which is commonly seen in real life. For example, in signage management, frequently encountered restrictions and events are encoded as signs and icons, while those of an occasional or exceptional nature are written in text. From an information-theoretic perspective, such a strategy is fundamentally the same as that used in entropy encoding and dictionary encoding [10, 31].

In time series processing and visualization, the algorithm for identifying *Frequent Long Patterns* (FLPs) plays a critical role. It has to be sufficiently tolerant to noise that causes variations to the patterns in the corpus. Without a high-degree of noise tolerance, the criteria of 'frequent' and 'long' could not be met, and the objective of visual compression could not be fulfilled. However, paradoxically the FLP algorithm is also required to exclude any anomalous patterns, which would easily be mistaken as frequently occurring patterns once they are visually compressed. In many situations, an anomalous pattern differs from frequent occurring patterns in a subtle way, and a conventional FLP algorithm may not handle the features that characterize such differences.

Fig. 1(A) shows six segments of a time series. They all appear to be fairly similar, and a typical FLP algorithm (Fig. 1(B)) would consider them to be the same. However, the 4th segment is an anomaly. An ideal visual compression should not replace this segment with a glyph, while compressing as many others as possible, as illustrated in Fig. 1(C).

In this paper, we present a visual analytics approach to address this paradoxical conflict of requirements upon an FLP algorithm. Consider that an FLP identified by an FLP algorithm is a model. The basic idea is to add an additional filtering capability to this model. As the filtering does not take place within the parameter space of the original FLP algorithm, it can handle features that the FLP algorithm cannot. The function of visual analytics is to support the following in an iterative manner:

- discover any anomalous patterns that may have been included by an FLP model;
- compute a bags of features that may be used to characterize different time series segments identified by an FLP model, and to visualize them in the feature space;
- assist users in a number of analytical tasks for identifying the appropriate filters, e.g., results clustering and tagging, and observing the effects of feature selection on filtering results; and
- facilitate model editing by transforming interaction in visualization to text-based instructions in the FLP model.

In the remainder of the paper, we first give a brief overview of time series analysis and visualization in Section 2. In Section 3 we outline a visual analytics loop for analyzing, visualizing, testing, and editing an FLP model, and we describe a prototype system that supports such a visual analytics loop. In Section 4, we describe an FLP algorithm that serves as the core component of an FLP model. In Section 5, we describe a collection of features that have been implemented to support model analysis and editing. In Section 6 we describe the model editing system and introduce the logical operators that can be used to combine models as well as build up rules for parameter refinement. Finally, in Section 7 we present a visualization to visualize the results of the time series compression.

2 Related Work

The related work can be divided in to 'time series analysis', focusing specifically on 'normalization', 'approximation' and 'similarity', and 'time series visualizations'.

- *Eamonn Maguire is with the Oxford e-Research Centre and Department of Computer Science, University of Oxford. E-mail: eamonn.maguire@st-annes.ox.ac.uk.*
- *Susanna-Assunta Sansone is with the Oxford e-Research Centre, University of Oxford. E-mail: sa.sansone@gmail.com.*
- *Min Chen is with the Oxford e-Research Centre, University of Oxford. E-mail: min.chen@oerc.ox.ac.uk.*

Manuscript received 31 March 2014; accepted 1 August 2014; posted online 13 October 2014; mailed on 4 October 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

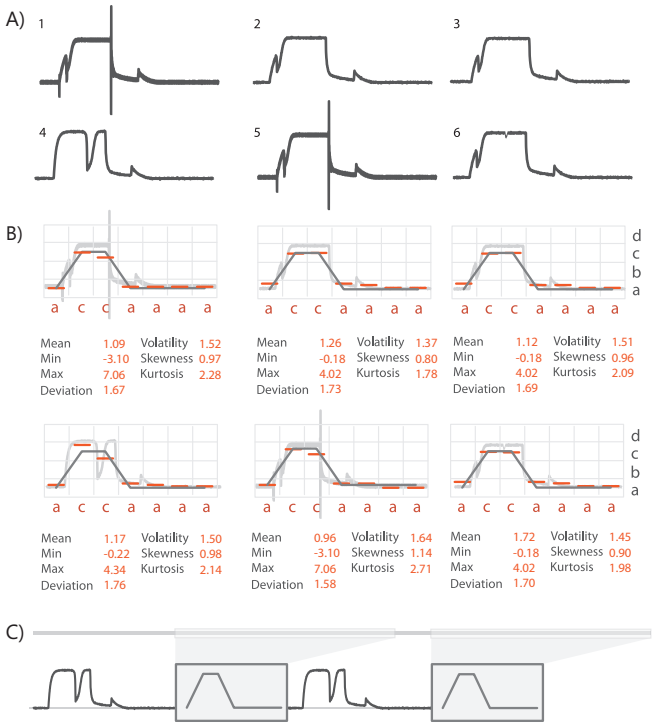

Fig. 1. A) Six time series subsections identified in a time series corpus of space shuttle valve time function taken from [27] B) Symbolic representation of each time results in all motifs being seen as the same, however their more detailed metrics/features differ C) Leaving out the anomalous pattern (A4) from the compression as requested by a user means that the final time series compresses the actual common motifs and leaves the anomalous pattern in plain view.

2.1 Time Series Analysis

Time series analysis encompasses five main sub-categories of operations: indexing, clustering, classification, summarization and anomaly detection [34].

Indexing: This is to provide support to storing, searching and retrieving time series in relation to a data repository.

Clustering: This is to determine how a time series compares with other time series in a database.

Classification: Given a predefined categorization, this is to determine if a new time series belongs to a specific class.

Summarization: Given a large time series dataset, this is to summarize the time series and/or its main attributes using a more compact representation (e.g., a feature vector or a visualization).

Anomaly detection: Given some time series, highlight anomalous incidents based on the profiles of a set of normal events (e.g., [28]).

For all of the above tasks, computing the similarity of between time series data is key. This area of research has a large amount of literature dedicated to it. Here we focus on some of the most relevant and commonly used algorithms that lend themselves to the task of time series comparison, both for whole and subsequence comparison.

Normalization. Before comparing two or more time series, an adjustment may need to be made to the data sets so that the data itself is comparable. For example, if we are looking for commonality in trends, the similarity score should be comparing the overall topology between points, rise and falls rather than being too concerned with where on the y-axis those trends reside. This step is called normalization. One of the most common approaches used for this step is Z-normalization, where the time series are manipulated to have a mean of zero and a standard deviation of one.

Approximation. Following normalization, we may wish to reduce the dimensionality of the data. This means that instead of comparing a time series in terms of real data points, we compare approximations that are smaller in size but largely retain the important features of the series. There are numerous techniques available for this process, all with their own faults and merits. Here we focus on the most relevant and commonly used techniques for approximation of a time series, those being: Discrete Fourier Transformation (DFT) [12] is an algorithm capable of representing the overall 'features' of a time series through spectral decomposition. This means that the signal represented by the time series is decomposed into a series of sine (and/or cosine) waves each represented by a Fourier coefficient [24] - this proves to be a very efficient way of compressing data; Discrete Wavelet Transformation (DWT) [8] is an algorithm similar in principle to DFT, however it has one key difference in that some of the wavelet coefficients represent small local regions of the series meaning that wavelets lend themselves to providing a multi-resolution view of a time series. The results of DWT do not lend themselves to being indexed (for comparison purposes). However an extension called the Haar Wavelet can be calculated efficiently and it's outputs can be indexed for efficient time series matching [7]. The key drawback with DWT is that the length of the time series must be an integral power of two [24]; Piecewise Linear Appromixation (PLA) [6] or Segmented Regression is an algorithm that breaks a time series up in to 'windows', then represents that window with a line segment that best fits the values in that window; Piecewise Aggregate Approximation (PAA) [24] works by splitting a time series up in to 'windows' where each window contains one or more time points. For each window, the average of the time points within is recorded and stored in a vector. The approach is simple yet is shown to rival DFT and DWT [34, 26]; Adaptive Piecewise Constant Approximation (APCA) [25] is algorithmically similar to PAA with extensions that further compress the representation using a technique common to data compression known as run-length encoding (RLE). This extension is the addition of a second number beside the mean value of the window indicating the length of the segment; and Symbolic Aggregate Approximation (SAX) [34, 35] which builds on PAA to take the average values calculated for each window then assigns a letter to that window based on where the mean value falls under a Gaussian curve. The result is a symbolic representation of a time series that lends itself to many computational manipulations such as hashing or storage in data structures such as suffix trees for fast pattern searching. Suffix trees have been used by Lin *et al* [34], to build a motif discovery tool for time series data, capable of highlighting anomalous data. A weakness of the symbolic approach however is the need to define a window and alphabet size ahead of time - these will differ depending on the domain due to differences in periodicity, variance for example.

Similarity. Following these steps, comparison can be performed on the outputs of the approximation which will give an idea of how close a time series is to another. Starting with the most simple of methods, there is Euclidean distance, where for two time series (t_1, t_2) the Euclidean distance would be calculated using $\sqrt{\sum_{k=1}^{n}(t_{1_k} - t_{2_k})}$ where t_{n_k} represents a point in time series n. This metric is fast to compute due to its simplicity, however it carries the caveat that the two sequences must be of the same length. When sequences are not of the same length, there are more complicated algorithms largely based on dynamic programming methodologies such as: Dynamic Time Warping [4] that allow for comparison between sequences of varying lengths and also support shifts in the series; Edit distance with Real Penalty (ERP) allows for gaps in a time series that may be penalised using some configurable value; Longest Common Subsequence (LCSS) [43] introduces a threshold value allowing a degree of mismatch between sequences; and Edit Distance on Real sequences (EDR) [9] merges the concept of gaps and mismatch thresholds presented in ERP and LCSS respectively. Due to the use of dynamic programming techniques, DTW, ERP, LCSS and EDR have limitations in that there are many comparisons made between sequences that have no relevance whatsoever. To address these limitations, there is the Fast Time Series Evaluation [37] algorithm that introduced a more efficient way of building up the comparison matrix meaning that non-related sequences were never compared.

2.2 Time Series Visualization

Time series visualizations, given their use across every possible field have been published about heavily in the past two decades. More complete surveys of time series visualization, can be found in [2, 13]. Here we highlight some of the work in the visualization domain that aims to ease the task of time series analysis and provide a more effective means for users to navigate their data. Such work includes: StackZooming [21] which provides a hierarchical zooming interface for time series data; a spectral visualisation system for visualizing overviews of financial data [23]; the use of 'lenses' to focus on particular areas of a time-series [30, 48]; LiveRAC [36] for computer/information system management; Horizon charts[19] that attempt to aid comparison of many time series in one height fixed display; TimeSearcher [20] and VizTree [33] that allow for exploration of time series via a visual query interface; spiral visualizations to allow for trend discovery in datasets [45, 11]; importance-driven layouts for time series data [16]; multiresolution techniques for exploration of time-series data [17]; and the visual exploration of frequent patterns in time series data [18].

Glyphs have been used for visualization in time series for a number of tasks. Many such uses have been in order to summarise a series [29, 15, 44, 47]. They have also been used to represent uncertainty in time series data [3].

3 A Visual Analytics Approach

Fig. 2 shows an overall pipeline for detecting FLPs in a time corpus, and for compressing a time series with glyphs. Steps 1, 2, 5 and 6 represent the pipelines traditionally used in the literature. Steps 3 and 4 represent the newly added visual analytics steps for model testing, visualization, analysis and editing.

Similar to the visual analytics system by Legg *et al* [32] for sports video search refinement, this system will allow users to refine FLP models resulting from a conventional FLP algorithm so that users may filter the FLPs most appropriate for their use case.

This pipeline is detailed as follows:

1. a **Time Series Corpus** as input to the system - this is usually a very large collection of time series.

2. a **Frequent Long Pattern (FLP) detection subsystem** - this detects FLPs in the time series, the output of which is a list of different FLP candidates. A user can choose a specific FLP as an individualized FLP model. The model will detect only patterns that match this FLP;

3. a **Visual Analytics Platform** consisting of a number of components to aid model output view, edit and test and refinement operations. The interface that realizes this platform is shown in Fig. 2 A,B,C and D. It is divided in to four linked panels serving a number of complementary functionalities. These are:

   - **A) Model Debugging Panel**: results from the FLP model are presented in this overview area where results of the model can be approved or rejected. The network view presents each motif as a glyph representing the approximation amongst its feature space. These glyphs can be arranged in a number of different topologies:

     (a) by distance between the symbolic representation of each motif - this is calculated on motifs with equal lengths through their euclidean distance. This is defined by the equation below from Lin *et al* [35]

$$MINDIST(S_X, S_Y) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (dist)(S_{X_i}, S_{Y_i})^2} \quad (1)$$

     where the *dist* function indicates the use of a lookup table to determine the distance between two letters, say A and D in the symbolic approximation. Since A and D are further away from each other than A and B for each, $dist(A,D)$ would return a larger value than $dist(A,B)$

     (b) by whether motifs have been accepted or not; or

     (c) by a hierarchy representing parent-child relationships where parent motifs (e.g., abbaca) have a sequence that contains N child motifs (e.g., abb, bb, bba, or baca)

     This view also provides users with a way to select a glyph and view more information about the motif it represents. A popup shown in Fig. 3A allows users to mark motifs as accepted or rejected, view their feature space in more detail, obtain their context in the collection of time series being analysed (see Fig. 2 B) and view the original time series.

   - **B) Time Series Overview Panel**: provides a summary view of the time series being used by the motif finding algorithm. This is also used to highlight where motifs occur within the time series to provide further contextual information to users.

   - **C) Model and Parameter Editing Panel**: split in to two sections: the *model* area - this area allows users to create new models, changing the window ω and alphabet size α and combine the results of these models; and the *refinement*, providing an interface for users to edit the feature ranges required for acceptance or rejection of a motif. Users can recalculate at any point and see the results update in Fig. 2 A. See Section 6.

   - **D) Feature Space Panel**: to visualize the feature space of the motifs, we use parallel coordinate plots. Users can brush a combination of axes to find the 'best' parameters that yield the motifs they really want. This is aided by a refinement window shown in 3 B and a link with the network view which filters out nodes that have parameters outside the brushed region(s) (see 2A and D). See Section 5.

4. a **Refinement Loop** which allows re-runs of the FLP detection model(s) given user-defined refinements to the conditions required for motif acceptance;

5. a collection of **Accepted Motifs**; and

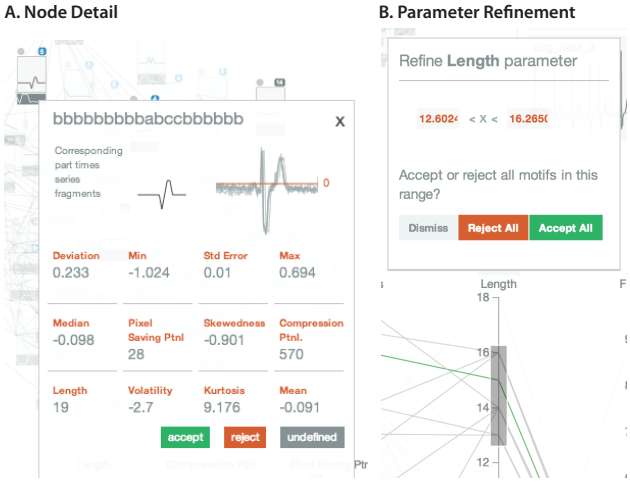

Fig. 3. A) Node detail view showing the feature space of the nodes, the approximation and the corresponding parts of the time series that this motif is associated with. Additionally, the context is shown for the motif in the time series panel on hovering over a node. B) On brushing an axis in the parallel coordinate plot, a parameter refinement window appears to allow the user to either remove results based on particular parameters or to accept them

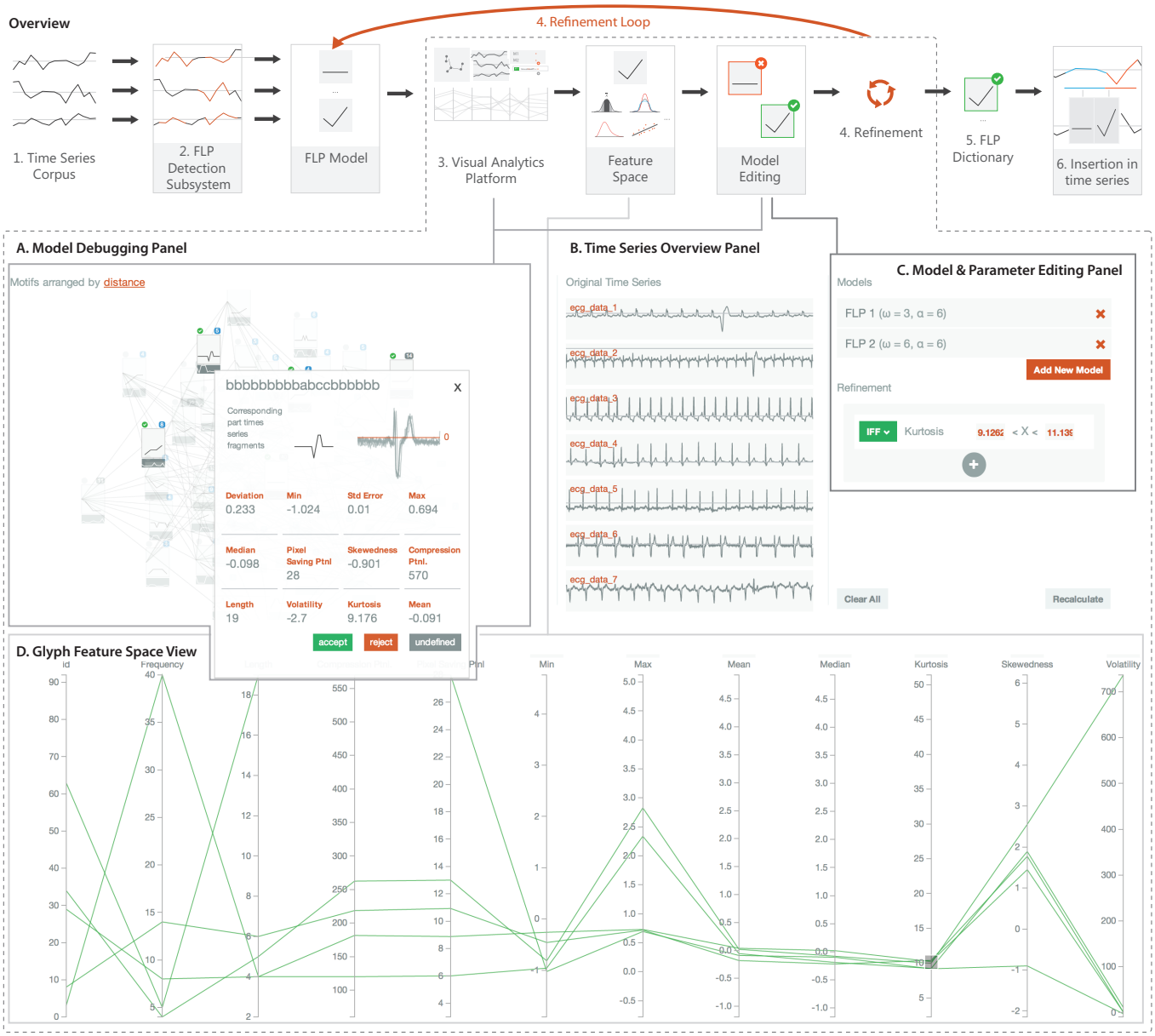

Fig. 2. The overview of the process followed in this paper and a screenshot of the visual analytics platform built to facilitate this process. The visual analytics platform interface has four interlinked panels to support: A) visual model debugging; B) overview of the larger time series to support context views for each motif (where it occurs in the actual time series); C) model and parameter editing; and D) navigation and refinement of the feature space for each motif.

6. a **Time Series Compression** step which inserts the motifs as glyphs in to time series for compression and/or anomaly detection (see Section 7).

4 FLP Detection Subsystem

In data compression, *dictionary encoding* is a family of methods that search a text for strings from a dictionary, and replace the matching strings with the corresponding codewords defined in the dictionary. The same dictionary is maintained by the encoder and the decoder, facilitating faithful translation from the original text to the compressed representation, and back to the original text. The primary goal of such a compression strategy is to reduce storage requirements. In a less algorithmic form, we apply a similar approach in writing by using abbreviations (e.g., UN for United Nations and symbols (e.g., § and ©)). Such a 'compression' method can be based on a general 'dictionary' applicable in a wide context (e.g., commonly-used currency signs), a 'dictionary' that is meaningful only to a specific group of people or in a specific context (e.g., mathematical symbols, abbreviations for measurement units), or an ad hoc 'dictionary' to address a very specific requirement (e.g., we introduced FLP as an abbreviation for *Frequent Long Pattern*). The primary goal is to reduce the time and cognitive load required for reading long and frequently-occurring text strings, while it also facilitates space saving in many cases.

The condition of 'long and frequently occurring' reflects the principle of *entropy encoding* in information theory [10]. The most well-known entropy encoding technique is Huffman coding. In everyday life, we can observe many intuitive entropy encoding schemes. For example, traffic signs are designed to speed up readings of restrictions and instructions, which otherwise would require a few words or sentences. For less commonly used restrictions and instructions, we still have text-based traffic sign boards. The balance between icon- or glyph-based and text-based visual representations in traffic signage features considerations of frequency of usage, length and complexity of the original text, importance, and various human factors (e.g., learnability, recognition, and memorization).

This has inspired us to apply the concepts of entropy encoding and

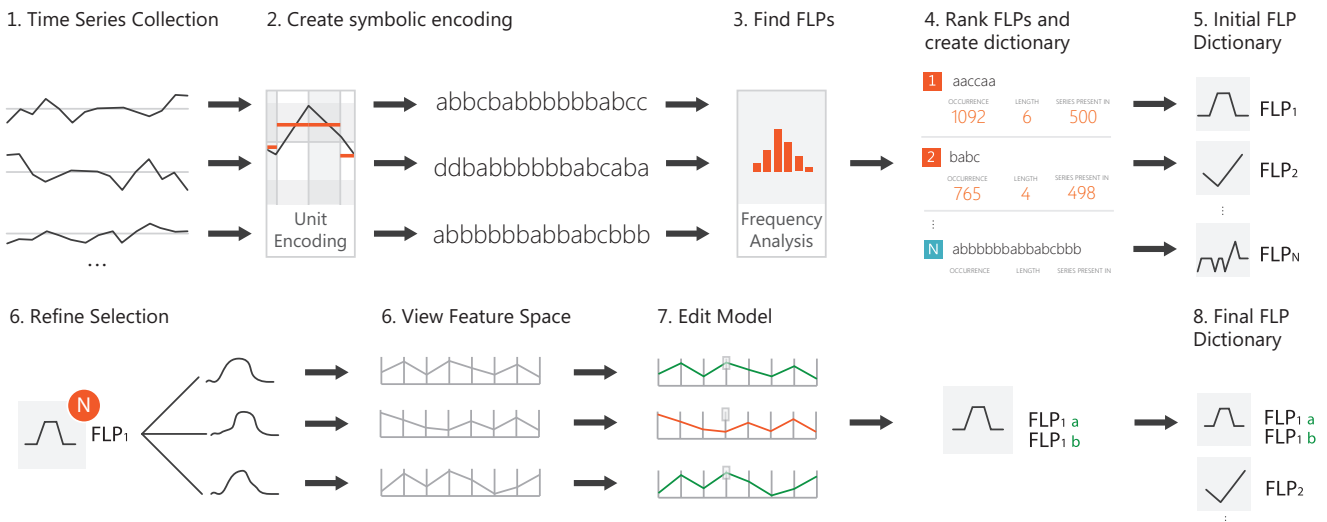

Fig. 4. Dictionary compilation steps – given a number of time series datasets $(T_1, T_2, \ldots, T_n)$ (1), each is approximated by a symbolic representation (2). A list of Frequent Long Patterns (FLPs) are found (3) from analysis of the symbolic encodings and these FLPs are ranked by their 'compression potential' (4). Highly-ranked FLPs are added to a dictionary. Finally, glyph representations are created for each FLP.

dictionary encoding to visualization, since they are everyday phenomena and underpinned by well-established mathematical theories. We can draw a parallel between a time series and a piece of text, and between a frequent long pattern in a collection of historical time series and a common long string in a text corpus. This analogy suggests that we can adopt a computational process to compile a dictionary, since there have been many previous works on symbolic processing of time series data in the literature (e.g., [33]). As glyphs have been used for time series visualization (e.g., [38]), we thus focus on a visual representation that supports dictionary-based compression while maintaining overview and selective detail views.

Fig. 4 illustrates the algorithmic steps for creating the FLP dictionary for a corpus of times series data. This dictionary creation algorithm is designed to process a collection of time series in order to establish a dictionary consisting of a set of FLPs in a specific context and can consists of the following steps:

1. **Time Series Collection.** It is necessary to collect a set of time series in order to generate good statistical indications about common patterns and outliers.

2. **Unit Encoding.** Given a collection of time series, the next step in *Dictionary Compilation* is to translate them to symbolic representations. These symbolic representations collectively form a 'text corpus'.

3. **Find FLPs.** In this step, an efficient algorithm based on a suffix-tree structure is deployed to detect a list of FLPs.

4. **Ranking FLPs.** For each detected FLP, we compute its potential compression capacity alongside other parameters in the feature space (see Section 5). We then choose the high capacity patterns and/or those with features matching any user-defined requirements defined in the visual analytics platform.

5. **Creating Glyphs.** The glyphs can be created automatically based on various attributes of the time series they represent.

Consider that we have N_p FLPs in a time series, and each FLP is of a length of $M_i (i = 1, 2, \ldots, N_p)$ samples. If each glyph requires a display space with a fixed width ω_g (in pixels), the compression ratio in terms of horizontal visual space requirement is:

$$C_{FLPs} = \frac{\omega_s \sum_{i=1}^{k} M_i}{\omega_g N_p} \tag{2}$$

where ω_s is the number pixels per sample, including interval space, along the t-axis in a conventional line graph. The overall compression ratio for the time series is

$$C_{timeseries} = \frac{\omega_s N_t}{\omega_s (n - \sum_{i=1}^{k} M_i) + \omega_g N_p} \tag{3}$$

Spatial compression is merely an indicator of the potential benefits of a frequency-based encoding strategy. In general, representing parts of a time-series as appropriate glyphs can benefit some common visualization tasks, such as identifying familiar FLPs, focusing attention on potential anomalies, and performing visual searches. The main cost is the loss of accuracy of those glyph-encoded parts of the time series. The trade-off is a universal mechanism at the heart of visualization. Many visualization techniques, such as zooming, glyph-based techniques and metro-maps, feature such a trade-off. In fact, the conventional line graphs for time-series visualization usually incur a significant loss of accuracy in comparison with the raw data, since the screen resolution is usually much lower than the data resolution. With interactive visualization, the loss of accuracy in overview can be recovered from details on demand [40], for example, using zoom or pop-up windows.

4.1 Unit Encoding

Let T be one of such a time series with N_t samples, we first divide it into N_{unit} *time units*, each of which consists of a fixed number of samples. In other words, there are N_{spu} samples per unit, such that $N_{unit} = \lceil N_t / N_{spu} \rceil$. In the literature, various terms have been used to denote such a time unit, e.g., 'time primitive' and 'chronon' in [5] and 'window' in [33].

An encoding scheme is then selected to map each unit to a symbol in an alphabet A. In this paper, we denote such symbols with lower case letters, such as a, b, and so on. Such an encoding scheme usually facilities an initial lossy compression in numerical representations. The potential compression ratio is influenced primarily by the size of the alphabet $|A|$, the number of samples per unit, N_{spu}, and the number of bits per sample value. However, some parts of the time series will not be encoded as glyphs and will be displayed as conventional time-series plots instead since their points do not fall within the realms of the frequently occurring patternss.

Being able to index and compare time series is of importance in this work due to the necessity to determine whether or not particular patterns in a time series are different from what has already been

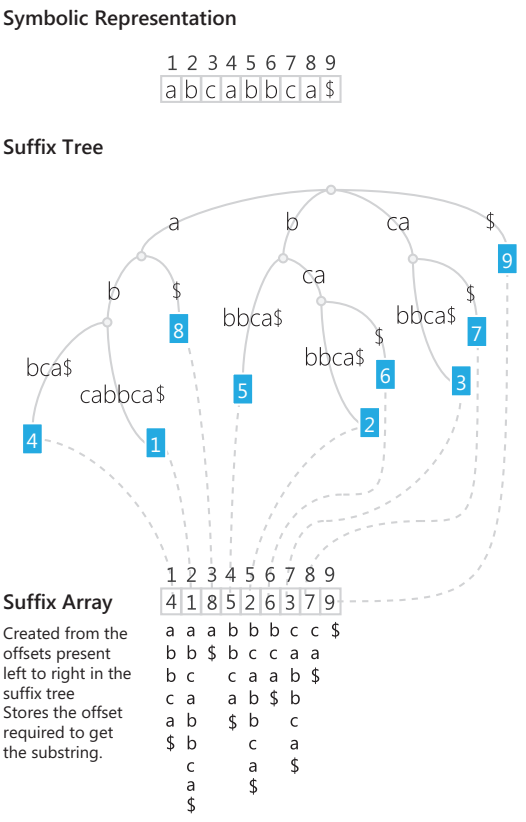

Fig. 5. Suffix tree and array representations, adapted from [22].

seen. In this work, we chose to base our algorithm on the symbolic approximation approach because it: 1) is widely used by the time series community; 2) is performant with O(n) complexity; and 3) has output ideally formed for indexing, hashing and storage in data structures such as suffix trees as performed in [33].

4.2 Frequent Long Pattern (FLP) Detection Algorithm

From the symbolic representation calculated in Section 4.1, we need to extract the most common, long substrings from available set of sequences with the expectation that selection of the most common strings will lead to greater compression.

The FLP detection model has three principle components: an 'enhanced' suffix array (with longest common (LCP) prefix array) to provide a fast, space efficient platform for fast string searching; an algorithm to compute the common elements in the string and their frequency; and a filter to fulfil any user-defined refinements on the features that motifs should have - by default, the model satisfies the requirement of finding those patterns matching the FLP properties of both 'long and frequent'.

4.2.1 Finding the FLP Candidates

A suffix tree [46, 42], or position tree is a data structure used to represent all suffixes of a string [42]. Fig. 5 is an example of such a tree and its corresponding suffix array adapted from Kasai *et al* [22]. Suffix trees thrive in their ability to speed up previously computationally intensive string operations such as: finding the longest common substring; finding repeats; finding maximal palindromes; and inexact string matching (the Weeder algorithm).

Although suffix trees are fast to query and build, they are costly to store. An alternative to the suffix tree is the suffix array, a sorted array of all suffixes that are able to perform all tasks that a suffix tree can perform in the same time when 'enhanced' with a Longest Common Prefix (LCP) array [1]. The LCP array gives for each adjacent pair of suffixes, the longest common prefix shared between them. For example, in Fig. 5, the longest common prefix between positions 6 and 7 in the suffix array, or '*bca*' and '*cabbca*' is 2, referring to the length of '*ca*' common to both suffixes. From this LCP array in combination with the suffix array, [1] showed that all suffix tree traversal types could be supported.

The algorithm, defined below takes a collection of time series approximations S_{approx}, finds all the common patterns, then returns a filtered list of FLPs matching the criteria defined in Section 4.2.2.

```
1:  procedure DETECT_FLPS(S_approx)
2:      Set_FLPs ← {}
3:      for all S_i ∈ S_approx do
4:          Array_suffix ← sort(createSuffixArray(S_i))
5:          Array_lcp ← createLCPArray(Array_suffix)
6:          pos_1 ← 0
7:          index ← 0
8:          while index < Array_lcp.length do
9:              pos_2 ← Array_suffix[index + 1]
10:             length ← Array_lcp[index]
11:             end_index ← max(pos1, pos2) + length
12:             FLP_candidate ← getSuffix(S_i, index, end_index)
13:             if FLP_candidate ∃ Set_FLPs then
14:                 get(Set_FLPs, FLP_candidate).occurrence++
15:             else
16:                 FLP_candidate.occurrence ← 1
17:                 Set_FLPs ← FLP_candidate
18:             end if
19:             index ← index + 1
20:             pos1 ← pos2
21:         end while
22:     end for
23:     return Filter_FLPS(Set_FLPs)                ▷ see 4.2.2
24: end procedure
```

4.2.2 Selecting the 'Best' FLPs

Let Ξ be an FLP found using the detection algorithm in Section 4.2.1, which can be written as a series of letters $\Xi = \alpha_1\alpha_2\ldots\alpha_m$. L_Ξ denotes its length (i.e., the number of letters) and F_Ξ denotes its frequency of occurrence obtained by the detection algorithm. F_Ξ can be the number of occurrence of Ξ in the data repository, or be moderated by a predefined maximum number. The compression ratio for Ξ is

$$C(\Xi) = \frac{\omega_s \times N_{spu} \times L_\Xi}{\omega_g} \tag{4}$$

In Eq. (4), ω_s is the minimal width along the t-axis required to adequately distinguish a data point in a time series (e.g., 2 pixels), ω_g be the screen width of a glyph, and N_{spu} is the window size for the symbolic approximation (i.e., the number of samples per letter). As the actual realization of this compression ratio depends on the probability of its occurrence in an input string, we thus moderate $C(\Xi)$ with F_Ξ as

$$P_C(\Xi) = F_\Xi \times C(\Xi) \tag{5}$$

We refer to $P_C(\Xi)$ as the *potential compression power* of Ξ. For example, consider a 20-letter string and Ξ, $N_{spu} = 10$. This FLP has 200 samples. If ω_s = 2 pixels, and ω_g = 40 pixels, we have $C(\Xi) = \frac{2\times10\times20}{40} = 10$. This implies that the glyph replacement will take up 10% of the original space requirement for the 200 samples. Assuming an unmoderated $F_\Xi = 50$, we have $P_C(\Xi) = 500$.

The key balance to be found in the approach is finding the optimal number for N_{spu} for any given dataset so that the system is able to compress optimally and intelligently. A larger number for N_{spu} will lead to a coarser representation of the time series, however it will probabilistically lead to further rates of compression. Conversely, a small number for N_{spu} will result in a more fine grained approximation for all time series but lead to less compression due to a generally higher amount of entropy in the approximation.

The selection of N_{glyphs} 'best' FLPs relies mainly on the P_C scores. This may depend on a predefined limit of N_{glyphs}, as too many glyphs would incur extra difficulties in learning, recognition and memorization. This may alternatively depend on a preset threshold for P_C. In the simplest case, the process can stop here and the results can be compiled into a 'dictionary' of FLPs $\mathscr{D}$. The resulting dictionary can then be used to compress a time series (see Section 7).

In some cases however, an FLP may classify what can be deemed an anomaly along with normal patterns due to subtle differences in the series even though the overall shape of the distribution is similar. This is where the visual analytics platform comes in to play whereby users can view the feature space of an FLP (mean, standard deviation, kurtosis, etc.) can sub classify an FLP and its corresponding time series with more fine-grained control.

5 Feature Space

As mentioned earlier, The FLP detection subsystem in Section 4 focuses on noise tolerance. Each of the selected FLPs determines an individualized model, Ξ, which is defined in the resolution of the unit encoding. This represents a significant approximation in terms of both temporal range and attribute value range. A selected FLP can easily encompass many similar patterns of the time series. To be able to distinguish between those time series segments that are 'valid' versus those that are not, we need the visual analytics platform to refine the model. First, we apply the individualized model, Ξ, to all or a subset of time series in the corpus. This would result in a collection of results, $\mathscr{R} = \{R_1, R_2, \ldots, R_m\}$, detected by using Ξ. We then construct a feature space for $\mathscr{R}$.

There are a variety of computable attributes for time series data used in the literature (e.g., 23 considered by Tam *et al* [41]). We consider a number of these variables to characterize each motif and the times series data they represent.

Alongside more trivial metrics such as *maximum* and *minimum* values, *average*, *standard deviation* and *variance*, the software also calculates the following:

1. *Correlation/error*: calculated as the Pearson product-moment correlation coefficient between the approximation values and the original time series values as a way of identifying how close the approximation is to the reality. This coefficient is defined in Equation 6 where X and Y are two arrays representing the approximated series and the time series respectively, and $\bar{X}, \bar{Y}$ represent the mean of both arrays.

$$\rho = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{6}$$

2. *Kurtosis*: kurtosis, as illustrated in Fig. 6 gives a measure of the 'peakedness' of a distribution. This is calculated using the equation defined in Equation 7 where X is the original time series T.

$$\kappa = \frac{1}{n\sigma_X^4}\sum_{i=1}^{n}(X_i - \bar{X})^4 \tag{7}$$

3. *Skewness*: the skewness, illustrated in Fig. 6 gives a measure of 'curve asymmetry' [41] and is defined in Equation 8.

$$\gamma = \frac{1}{n\sigma_X^3}\sum_{i=1}^{n}(X_i - \bar{X})^3 \tag{8}$$

4. *Burstiness*: also called local variance, indicates how quick adjacent values rise or fall. The equation below has been adapted from [39].

$$L_V = \frac{1}{n-1}\sum_{i=1}^{n-1}\frac{3(X_i - X_{i+1})^2}{(X_i + X_{i+1})^2} \tag{9}$$

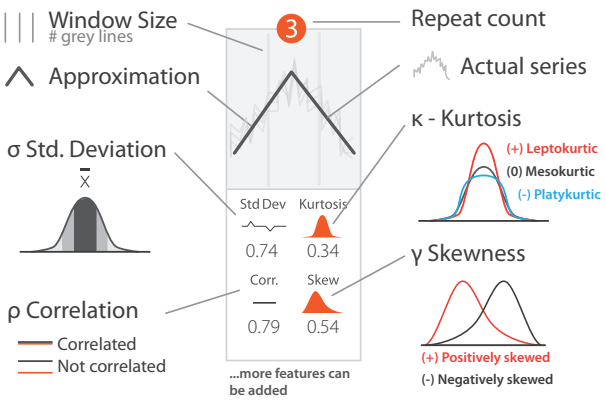

Fig. 6. A) The compression power of a particular FLP and the part of the time series compressed by a glyph is given by an overview bar. B) Details are available on demand when users click on a glyph in an implementation inspired by StackZooming [21].

Additionally, each motif has a compression potential calculated as part of the FLP algorithm discussed in Section 4.2.

Similar to Tam *et al* [41], we visualize these parameters using parallel coordinate plots (see Fig. 2D) and provide interaction to allow users to edit the model. Additionally, these features can be incorporated in to glyphs as illustrated in Fig. 6 and visualized for the user.

6 Model Editing

The process of model editing is enabled through many components within the visual analytics platform shown in Fig. 7 A and B. Users can click on nodes in the network view and accept or reject individual motifs whilst seeing the effect of such refinements immediately. The model editing window, shown in more detail in Fig. 7 A2 and B2 can be split in to two sections: 1) the model panel; and 2) the refinement panel.

The model panel is there to support the generation of motifs with differing window and alphabet sizes for the SAX algorithm used for FLP detection. These models are combined together and the combination of model outputs provides the total number of motifs available for selection.

The refinement panel supports the acceptance or rejection of motifs based on one or more of the parameters in the feature space. Users can combine these operations with a number of logical operations such as union (OR), intersection (AND) and subtract (SUB). Users can also specify whether values should lay within a range or be less than, less than or equal to, greater than or greater than or equal to some limit value.

The combination of simple logic statements provides a way for users to visually program the model so that it outputs only the desired results.

Fig. 7 shows an overview of the visual analytics workflow applied for the engineering example depicting the solenoid current measurements on a Marotta valve used to control fuel on a space shuttle. In Fig. 7 A, a network view displays the motifs found by the FLP detection algorithm as glyphs. Hovering over a glyph highlights the area that glyph represents in the time series corpus. This shows that one FLP represents all of patterns in the time series corpus, however an expert (derived from annotations in the original data set) marked the FLP highlighted in the black outline as anomalous. The next step will involve filtering this pattern out by looking in more detail at the FLPs and the time series patterns they represent.

Fig. 7 B shows a view of all 16 occurrences of the FLP of interest from the first step. Each occurrence has its own feature space representing the actual series represented. We have clicked on the motif of interest and can see in the popup that the profile matches the anomaly we wish to exclude.

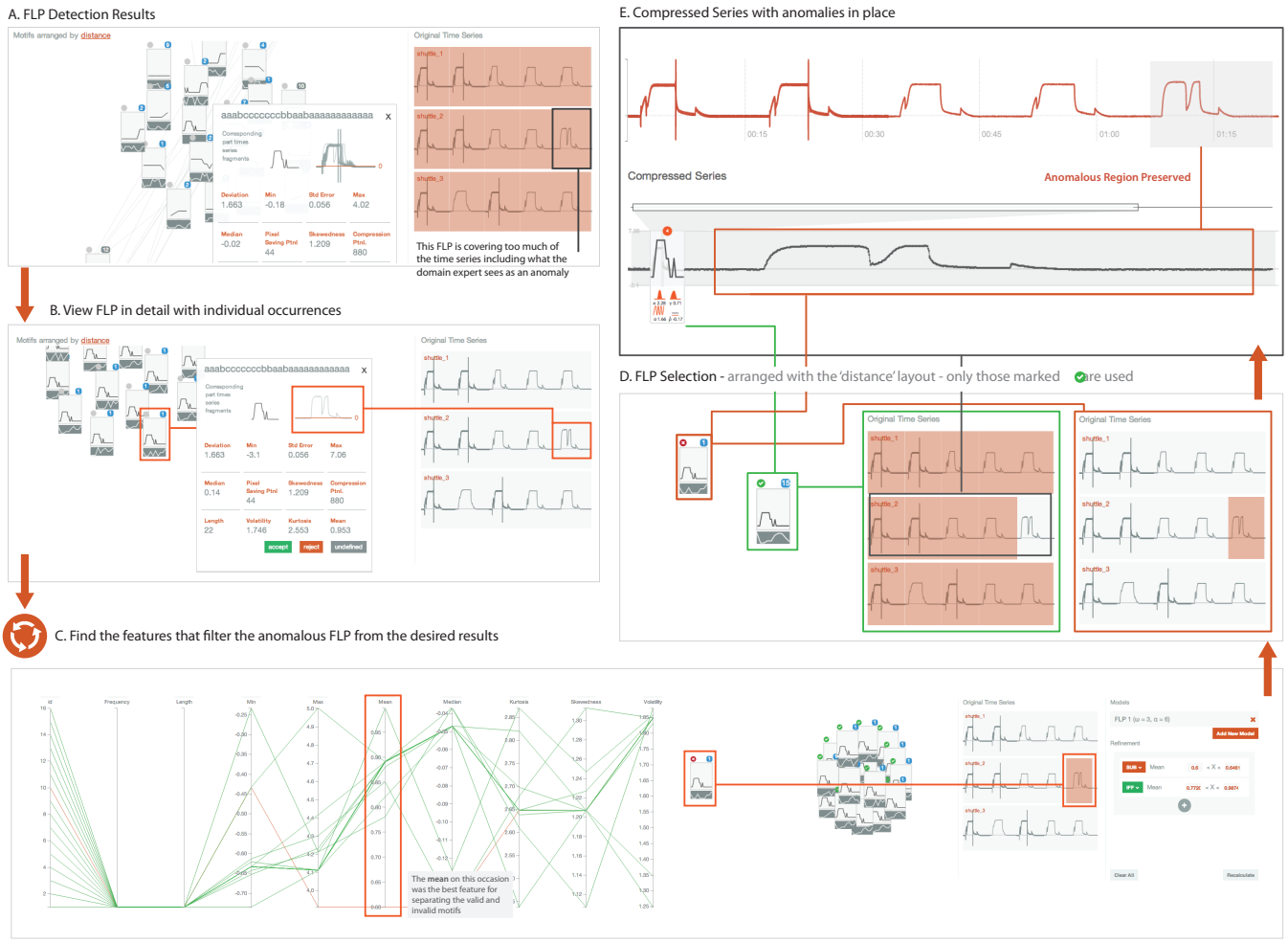

Fig. 7. A visual analytics approach to model refinement on an engineering time series depicting the solenoid current measurements on a Marotta valve used to control fuel on a space shuttle [14].

Having identified the anomalous motif, Fig. 7 C shows how, through use of the parallel coordinates, we can find the features that are able to classify between valid and invalid FLPs. In this case, the mean was a good choice since the anomalous FLP had a mean much lower than the others. The network layout automatically updates to visually separate the rejected FLPs from those that were accepted. In D, the set of FLPs are then exported as a dictionary where feature refinements are also preserved for later compression. Finally, a time series can be compressed using the FLPs in the dictionary and some time series T to create a compressed representation of the time series with the anomalous region preserved. This visual compression step is described more in the next section.

7 Visualizing the Compressed Time Series

Given a dictionary of FLPs $\mathscr{D}$ obtained from the FLP Detection Subsystem (Section 4.2), the process of glyph-based time series compression, illustrated in Fig. 8, can be applied repeatedly to many time series. Using traffic signage as an analog, once functional and visual representations of different signs have been decided, we can place them in any applicable location.

Glyph-based time series compression involves two steps: 1) given a dictionary of FLPs $\mathscr{D}$ and a time series T, create a compressed representation of a time series; and 2) using this representation of the time series, render it.

For the first step, we create the enhanced suffix array representation as performed when creating $\mathscr{D}$. From this data structure, a binary search is performed for each of the FLPs defined in $\mathscr{D}$ to identify all positions (defined by the suffix array) the particular FLP appears in. From the constructed list of matching positions for each FLP, some additional processing is made for the rendering layer. The first is to merge directly adjacent matches, that is, say one of our FLPs is composed of $abcd$, then for $abcdabcdddd,\ldots$, we have a record of $\Xi_{abcd} \uplus [1,5]$ including a list of matching indexes. Here the operator $\uplus$ denotes annotation, meaning that a FLP Ξ of symbols $abcd$ is annotated with index values 1 and 5. Since the distances in the start indexes for the occurrences of Ξ_{abcd} are separated exactly by its length L_Ξ, $1 + L_\Xi = 5$, the second occurrence can be merged with the first, and a 'repeat count' for Ξ_{abcd} is increased, so $\Xi_{abcd} \uplus [1,R2]$. Finally, a further check is performed to ensure that the indexes for the matched FLPs do not overlap, that is if $\Xi_{bcd} \uplus [2,6]$ has start indexes overlapping with $\Xi_{abcd} \uplus [1,R2]$. In this case, the FLP with the lowest compression ratio (C) power will be removed from the results. At the end of this process, the software produces a map from each start index of a FLP to its symbolic representation and repeat count.

The second step, rendering the time series, involves the incorporation of the glyphs representing compressed regions in to a time series visualization, the design of which is shown in Fig. 9. This representation maintains an overview bar showing the overall length of the time series whilst showing how much of a region has been compressed by a glyph. The glyphs provide information about the approximation, the run length (how many times the motif was repeated) and a number of metrics from the feature space. Glyphs can be interacted with and expanded to provide details of the underlying compressed time series represented by the glyph. This can be seen in Fig. 7 A and B. Addi-

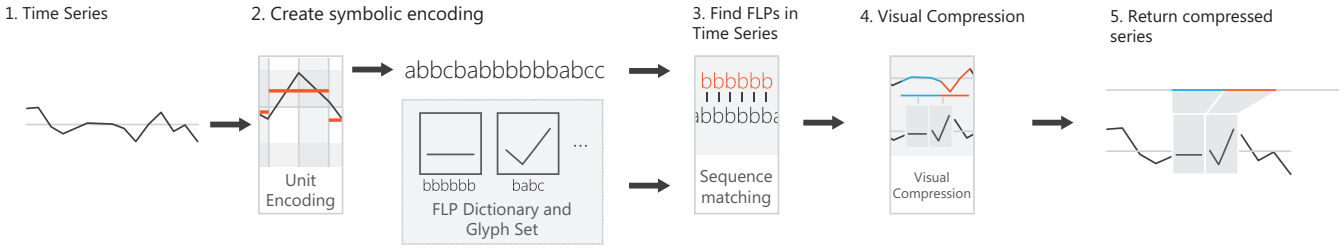

Fig. 8. Dictionary-based compression steps – Given a time series T_1, we encode it symbolically (2). Then, with the FLP dictionary created in (A), an algorithm finds the occurring FLPs in the incoming sequence (3). These found FLPs are replaced with glyphs (4) and the compressed time series is visualized.

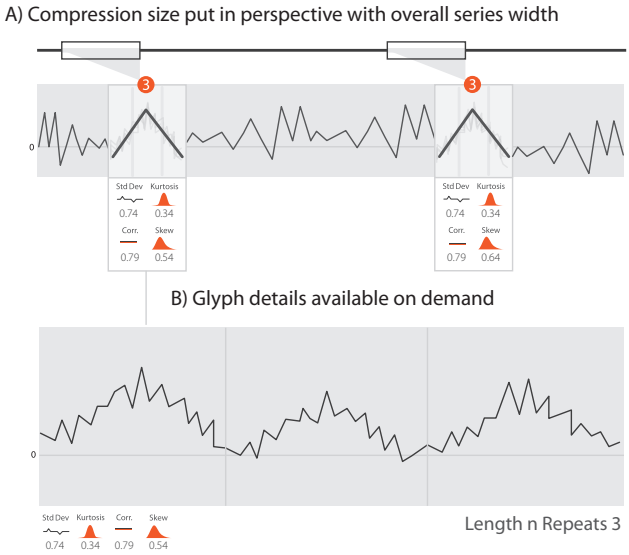

Fig. 9. A) The compression power of a particular FLP and the part of the time series compressed by a glyph is given by an overview bar. B) Details are available on demand when users click on a glyph in an implementation inspired by StackZooming [21].

tionally, when large areas are compressed, this changes the rendering of a time series somewhat since the series can spread over a much larger range. In many cases this is acceptable, but in some cases, the spacing may introduce interpretation problems. To navigate this issue, users can select to maintain the aspect ratio of the original series.

8 Software Availability

The software will be open source (on publication) and has been developed in a modular way to enable reuse at any level of the software stack. Each of the three components, the: FLP detection subsystem, implemented in Python; visual analytics platform, implemented as a thick client built with HTML5, CSS and JavaScript (with D3 and jQuery); and library to visualize the results, also built with HTML5, CSS and JavaScript (with D3 and jQuery) are available separately and can be run independently. The visual analytics platform and the compressed series output code rely on a simple JSON (JavaScript Object Notation) serialized format while the FLP detection subsystem outputs its results in the format required by the front end components.

Due to the modularity in the code base, and in particular the visual analytics platform, we envisage the software being of use in model refinement for similar problems requiring a visual analytics approach.

9 Conclusions and Future Work

In this work, we utilized the capabilities of visual analytics to address a challenging problem in time series analysis and visualization. In order to refine a model derived from the traditional FLP algorithm, visual analytics acted as a 'software development tool' by enabling users to test the model, analyze and visualize its results in a feature space, identify false positive results, identify related parameter ranges, and edit the model accordingly.

Because the feature space is computed directly with the original time series data, it can capture finer feature differences between an anomaly and its closely related FLP. Because the computation is only for a limited set of test results, it is also cost-effective. Most importantly, this allows human users to have the direct control over the tasks of debugging the results from a model and selecting features for refining a model.

In our future work, we would like to develop new visual representations of model space, which can support model visualization and editing in a semantically more meaningful manner.

References

[1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[2] W. Aigner, S. Miksch, W. Mller, H. Schumann, and C. Tominski. Visualizing time-oriented data — A systematic view. *Computers & Graphics*, 31(3):401 – 409, 2007.

[3] W. Aigner, S. Miksch, B. Thurnher, and S. Biffl. Planninglines: novel glyphs for representing temporal uncertainties and their evaluation. pages 457–463, 2005.

[4] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[5] M. Bögl, W. Aigner, P. Filzmoser, T. Lammarsch, S. Miksch, and A. Rind. Visual analytics for model selection in time series analysis. *IEEE Transactions on Visualization and Computer Graphics, Special Issue "VIS 2013"*, 19, 12/2013 2013.

[6] S. H. Cameron. Piece-wise linear approximations. Technical report, DTIC Document, 1966.

[7] K. Chan and A. W. Fu. Efficient time series matching by wavelets. pages 126–133, 1999.

[8] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133. IEEE, 1999.

[9] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[10] M. Chen and H. Jaenicke. An information-theoretic framework for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1206–1215, 2010.

[11] Y. Drocourt, R. Borgo, K. Scharrer, T. Murray, S. I. Bevan, and M. Chen. Temporal visualization of boundary-based geo-information using radial projection. *Computer Graphics Forum*, (3):981990, 2011.

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. volume 23. ACM, 1994.

[13] S. Fernandes Silva and T. Catarci. Visualization of linear time-oriented data: a survey. In *Web Information Systems Engineering, 2000. Proceedings of the First International Conference on*, volume 1, pages 310–319 vol.1, 2000.

[14] B. Ferrell and S. Santuro. Nasa shuttle valve data. http://www.cs.fit.edu/ pkc/nasa/data/ ., 2005.

[15] J. Fuchs, F. Fischer, F. Mansmann, E. Bertini, and P. Isenberg. Evaluation of alternative glyph designs for time series data in a small multiple setting. 2013.

[16] M. C. Hao, U. Dayal, D. A. Keim, and T. Schreck. Importance-driven visualization layouts for large time series data. pages 203–210, 2005.

[17] M. C. Hao, U. Dayal, D. A. Keim, and T. Schreck. Multi-resolution techniques for visual exploration of large time-series data. pages 27–34, 2007.

[18] M. C. Hao, M. Marwah, H. Janetzko, U. Dayal, D. A. Keim, D. Patnaik, N. Ramakrishnan, and R. K. Sharma. Visual exploration of frequent patterns in multivariate time series. *Information Visualization*, 11(1):71–83, 2012.

[19] J. Heer, N. Kong, and M. Agrawala. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations. In *In Proc. ACM Human Factors in Computing Systems (CHI*, pages 1303–1312, 2009.

[20] H. Hochheiser and B. Shneiderman. Interactive exploration of time series data. In *Discovery Science*, pages 441–446. Springer, 2001.

[21] W. Javed and N. Elmqvist. Stack zooming for multi-focus interaction in time-series data visualization. pages 33–40, 2010.

[22] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching*, pages 181–192. Springer, 2001.

[23] D. A. Keim, T. Nietzschmann, N. Schelwies, J. Schneidewind, T. Schreck, and H. Ziegler. A spectral visualization system for analyzing financial time series data. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC conference on Visualization*, EUROVIS'06, pages 195–202, 2006.

[24] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001.

[25] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM SIGMOD Record*, volume 30, pages 151–162. ACM, 2001.

[26] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[27] E. Keogh, J. Lin, and A. Fu. Time series datasets, http://www.cs.ucr.edu/ eamonn/discords/.

[28] E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Data mining, fifth IEEE international conference on*, pages 8–pp. IEEE, 2005.

[29] E. Keogh, L. Wei, X. Xi, S. Lonardi, J. Shieh, and S. Sirowy. Intelligent icons: Integrating lite-weight data mining and visualization into gui operating systems. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 912–916, 2006.

[30] R. Kincaid. SignalLens: Focus+Context applied to electronic time series. *IEEE transactions on visualization and computer graphics*, 16(6):900–907, 2010.

[31] W. Lang, M. Morse, and J. M. Patel. Dictionary-based compression for long time-series similarity. *Knowledge and Data Engineering, IEEE Transactions on*, 22(11):1609–1622, 2010.

[32] P. Legg, D. Chung, M. Parry, R. Bown, M. Jones, I. Griffiths, and M. Chen. Transformation of an uncertain video search pipeline to a sketch-based visual analytics loop. *IEEE transactions on visualization and computer graphics*, 19(12):2109–2118, 2013.

[33] J. Lin, E. Keogh, and S. Lonardi. Visualizing and discovering non-trivial patterns in large time series databases. *Information visualization*, 4(2):61–82, 2005.

[34] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. *DMKD*, 2003.

[35] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15, 2007.

[36] P. McLachlan, T. Munzner, E. Koutsofios, and S. North. LiveRAC: interactive visual exploration of system management time-series data. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, page 14831492, 2008.

[37] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 569–580. ACM, 2007.

[38] P. Saraiya, P. Lee, and C. North. Visualization of graphs with associated timeseries data. In *Proc. IEEE Information Visualization*, pages 225–232, 2005.

[39] S. Shinomoto, K. Shima, and J. Tanji. Differences in spiking patterns among cortical neurons. *Neural Computation*, 15(12):2823–2842, 2003.

[40] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[41] G. K. Tam, H. Fang, A. J. Aubrey, P. W. Grant, P. L. Rosin, D. Marshall, and M. Chen. Visualization of time-series data in parameter space for understanding facial dynamics. In *Computer Graphics Forum*, volume 30, pages 901–910. Wiley Online Library, 2011.

[42] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[43] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 216–225. ACM, 2003.

[44] M. O. Ward and Z. Guo. Visual exploration of time-series data with shape space projections. In *Computer Graphics Forum*, volume 30, pages 701–710. Wiley Online Library, 2011.

[45] M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. In *Infovis*, volume 1, pages 7–14, 2001.

[46] P. Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE, 1973.

[47] H. Wickham, H. Hofmann, C. Wickham, and D. Cook. Glyph-maps for visually exploring temporal patterns in climate data and models. *Environmetrics*, 23(5):382–393, 2012.

[48] J. Zhao, F. Chevalier, E. Pietriga, and R. Balakrishnan. Exploratory analysis of time-series with ChronoLenses. *IEEE transactions on visualization and computer graphics*, 17(12):2422–2431, 2011.