Time Series Novelty Detection with Application to Production Sensor Systems

by

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

May 2011

St. John's Newfoundland

To Lisa and Georgia

Abstract

Modern fiber manufacturing plants rely heavily on the use of automation. Automated facilities use sensors to measure fiber state and react to data patterns, which correspond to physical events. Many patterns can be predefined either by careful analysis or by domain experts. Instances of these patterns can then be discovered through techniques such as pattern recognition. However, pattern recognition will fail to detect events that have not been predefined, potentially causing expensive production errors. A solution to this dilemma, novelty detection, allows for the identification of interesting data patterns embedded in otherwise normal data. In this thesis we investigate some of the aspects of implementing novelty detection in a fiber manufacturing system. Specifically, we empirically evaluate the effectiveness of currently available feature extraction and novelty detection techniques on data from a real fiber manufacturing system.

Our results show that piecewise linear approximation (PLA) methods produce the highest quality features for fiber property datasets. Motivated by this fact, we introduced a new PLA algorithm called improved bottom up segmentation (IBUS). This new algorithm produced the highest quality features and considerably more data reduction than all currently available feature extraction techniques for our application.

Further empirical results from several leading time series novelty detection techniques revealed two conclusions. A simple Euclidean distance based technique is the best overall when no feature extraction is used. However, when feature extraction is used the Tarzan technique performs best.

Acknowledgements

I would first like to thank Dr. Dennis Peters for his supervision during my Masters program. His expert insight, guidance and especially patience during my multi-year absence from the program is appreciated. I would also like to thank Chris Dawson, mainly for convincing me to take on this research while I was working at INSTRUMAR Limited. Chris also provided crucial feedback during the work relating to the fiber manufacturing domain - something only an expert of the domain could provide.

I would also like to acknowledge the funding support provided by INSTRUMAR Limited, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Atlantic Canada Opportunities Agency (ACOA) through the Atlantic Innovation Fund (AIF).

It should also be mentioned here that portions of this thesis have been already been published. Parts of chapters 1 and 2 were included in a paper in the *Proceedings of the 15th Annual Newfoundland Electrical and Computer Engineering Conference* (NECEC 2005) [3]. In addition, chapter 3 is an expanded version of a paper included in the *Proceedings of the Fourth IASTED International Conference on Signal Processing, Pattern Recognition, and Applications* (SPPRA 2007) [4].

Contents

List of Figures

Notation

Acronym	Description
APCA	Adaptive Piecewise Constant Approximation
BUS	Bottom-Up Segmentation
DFT	Discrete Fourier Transform
DSL	Domain Specific Language
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
FTF	Filter Template File
HWT	Haar Wavelet Transform
IBUS	Improved Bottom-Up Segmentation
PAA	Piecewise Aggregate Approximation
PLA	Piecewise Linear Approximation
SAX	Symbolic Aggregate approXimation
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SVR	Support Vector Regression
TRF	Test Runner File

Symbol	Definition
$\vec{x}$	A time ordered series of real values.
n	The length of $\vec{x}$.
x_i, $x[i]$	The i^{th} value of $\vec{x}$ where $0 \leq i < n$.
$\vec{X}$	A vector of features.
N	The length of $\vec{X}$ plus any bookkeeping data.
C_{ratio}	The compression ratio, $C_{ratio} = \frac{N}{n}$.
$maxError$	The maximum distance between an approximating line and the corresponding points in $\vec{x}$.
`merge(s1, s2)`	A function that returns a new segment consisting of the first point of segment `s1` and the last point of segment `s2`.
`error(s, t)`	A function that returns the distance between an approximating segment `s` and the original time series `t`. The calculation used is called the *time aligned Euclidean distance* and is described in Section 3.2.1.
train	The data set that the novelty detection algorithm is trained on.
test	The data set that is input into the novelty detection algorithm for analysis. The output of the algorithm is *actAnomaly*.
actAnomaly	The data set that represents what the novelty detection algorithm considers novel. Any value above zero in this data set is considered to be a novelty. An algorithm associates a higher degree of novelty with a higher value in *actAnomaly*.

$expAnomaly$	The data set is derived from an anomaly definition file. Any anomalies defined in the file are represented as a value of one in the $expAnomaly$ data set. All other areas have a value of zero.
C_t	The total number of data points in the $actAnomaly$ data set.
C_e	The number of $actAnomaly$ data points that fall within the anomaly regions defined in the $expAnomaly$ data set.
C_a	The number of $actAnomaly$ data points that are greater than zero.
C_d	The number of $actAnomaly$ data points that fall within the anomaly regions defined in the $expAnomaly$ data set and are greater than zero.
R_{Det}	the rate of detection, which is defined to be $\frac{C_d}{C_e}$
R_{FP}	the rate of false positives, which is defined to be $\frac{C_a - C_d}{C_t - C_e}$

Chapter 1

Introduction

The demand for improved efficiency and quality in the manufacturing industry has led to an increased use of automation. The success of automated production depends largely on control systems that monitor material state and take appropriate action. One method of monitoring uses sensors to detect changes in physical properties of the material being produced. In some cases, this measurement data has been interpreted manually by qualified human operators. This has worked well for many years in part because the amount of data has been small and manageable. Advances in data capture technology and the availability of inexpensive storage are making it possible to record and store vast amounts of measurement data. As a result, human operators are becoming less able to detect all material defects in a timely and cost effective fashion. There is a growing realization that for such monitoring systems to be effective, the data analysis must be automated.

Automating data analysis for the detection of faults is not a new concept in the manufacturing industry. For years computational intelligence techniques have been

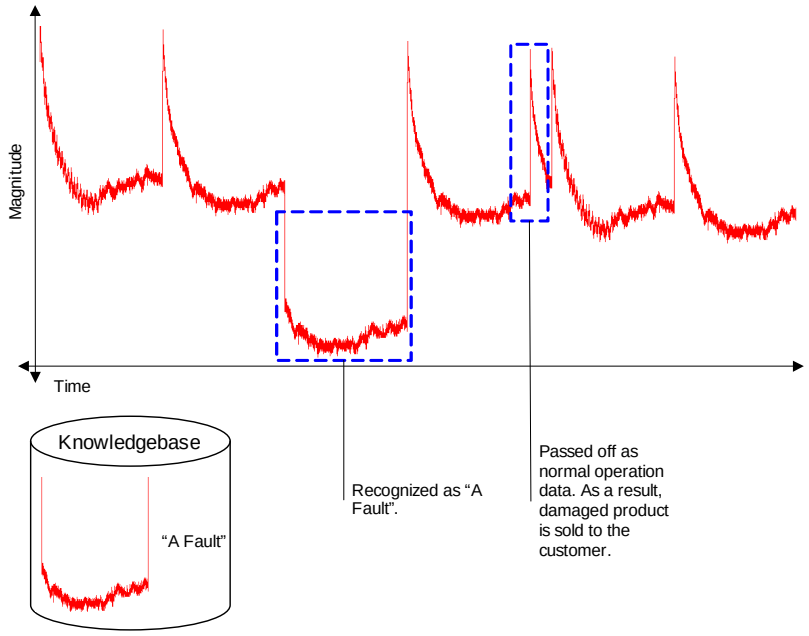

Figure 1.1: An incomplete knowledge base will ultimately cause interesting events to be missed.

used to detect tool breakage, product defects, etc. [18, 36, 15]. On the forefront of industrial monitoring is INSTRUMAR Limited, a company based in Newfoundland and Labrador. This company has developed a sophisticated on-line synthetic polymer fiber monitoring system called the Attalus^{TM} Fiber System [6, 5, 7, 8, 9]. Synthetic polymer fiber in this application is used to make carpet and other industrial fibers.

The basis for most of these techniques is to match the data to known patterns, which correspond to physical events. The matching process can be implemented with techniques such as pattern recognition, similarity search, or by other specialized means [15, 2]. Such systems are fine for detecting faults that have previously occurred and been detected. The procedure for doing so involves two steps: define data signatures for all possible faults, and then search for them continually. This works quite well if

nothing unexpected happens. In this case, faults are expected patterns since we are searching for them. As shown in Figure 1.1, the obvious problem with this system is made evident when an unidentified event occurs. The system would fail to recognize the event and would therefore pass it off as normal operation data. In the case of a production line, this oversight could lead to substantial financial losses in off-specification product. In other words, sometimes the most interesting fault being searched for is not known. It may be unknown because it has never occurred before (as described above) or computing a data signature for it is too expensive [22].

Clearly, enumerating all known fault patterns cannot detect all events. A more robust approach, called novelty detection, can be defined as "... the automatic identification of unforeseen or abnormal phenomena embedded in a large amount of normal data." [40] Much research has been devoted to this approach in general. Until recently however, novelty detection on time series data has received much less attention than novelty detection in other types of data [30]. This is in part due to the fact that time series databases are usually very large and the notion of similarity can be subjective. Similarity may depend on the user, the domain, and the task at hand [28].

The occurrence of false positives is a problem for many novelty detection methods. A false positive occurs when a particular method marks something as novel when it is not. As a result, novelty detection systems should not be solely responsible for monitoring of a product. An ideal monitoring system could take advantage of both approaches. A pattern matching approach is very capable of finding known data patterns and novelty detection allows the system to expand its knowledge base either autonomously or by a supervised means.

In this research we investigate some of the aspects of implementing novelty de-

tection in a fiber manufacturing system. Specifically, we evaluate the effectiveness of currently available feature extraction and novelty detection techniques on data from a real fiber manufacturing system.

1.1 Background

The manufacture of fiber and fiber based products represents an enormous global industry. In general, fiber can be grouped into two categories: natural and man made. Natural fiber comes from several different sources in nature: animal (e.g., silk and wool), mineral (e.g., asbestos), or vegetable (e.g., cotton) [39]. Man made fiber is created from fiber forming substances and does not occur naturally. Man made fiber can be divided further into two groups: cellulosics and synthetics [16]. Cellulosic fiber is formed from plants that naturally produce cellulose. Synthetic fiber is created entirely from chemical compounds (i.e., polymers) made from petroleum or natural gas. In this work our data sets are all based on synthetic fiber, so we will not consider other types of fiber.

1.1.1 Synthetic Fiber Manufacturing

Manufacturing synthetic fiber is conceptually simple. As illustrated in Figure 1.2, free-flowing polymer is extruded through the small holes of a spinnerette and is then solidified to produce fiber filaments [16]. The viscosity of a polymer can be lowered to allow extrusion by heating or dissolving into a solution. The individual polymer fiber filaments are solidified by cool air or immersion in a suitable fluid. Filaments of fiber are usually grouped together into a tow of fiber. A tow is a continuous strand

of many fiber filaments grouped together.

After the fiber is solidified it can undergo a number of post processing stages. These stages are used to modify the properties of the fiber to meet various requirements. We will focus on the processes of interlacing/noding, and lubricating/finishing [16, 6]. Noding, as illustrated in Figure 1.3, is the process of compressing a tow of fiber at regular intervals, creating nodes that hold the fiber together. Liquid finish is applied to fiber in order to create desirable properties such as smoothness, luster, water repellency, flame retardancy, etc. [39].

To ensure fiber quality, many properties of the produced fiber are measured. We will focus on the following physical properties:

- **Denier**: the weight in grams per 9000 meters of the final product [39].
- **Node Count**: the number of nodes in the fiber that occur during a sampling period [6].
- **Node Quality**: the compactness of a node [6].
- **Amount of Finish**: the amount of finish on the final product [6].

1.1.2 Synthetic Fiber Monitoring

The basic structure of a monitoring and control system is depicted in Figure 1.4. The manufacturing process in this case is where the synthetic fiber gets produced. As described in the previous section, a number of physical properties need to be measured to ensure that the fiber meets the required specification. INSTRUMAR's Attalus^{TM} Fiber System determines these properties by using a novel electromagnetic

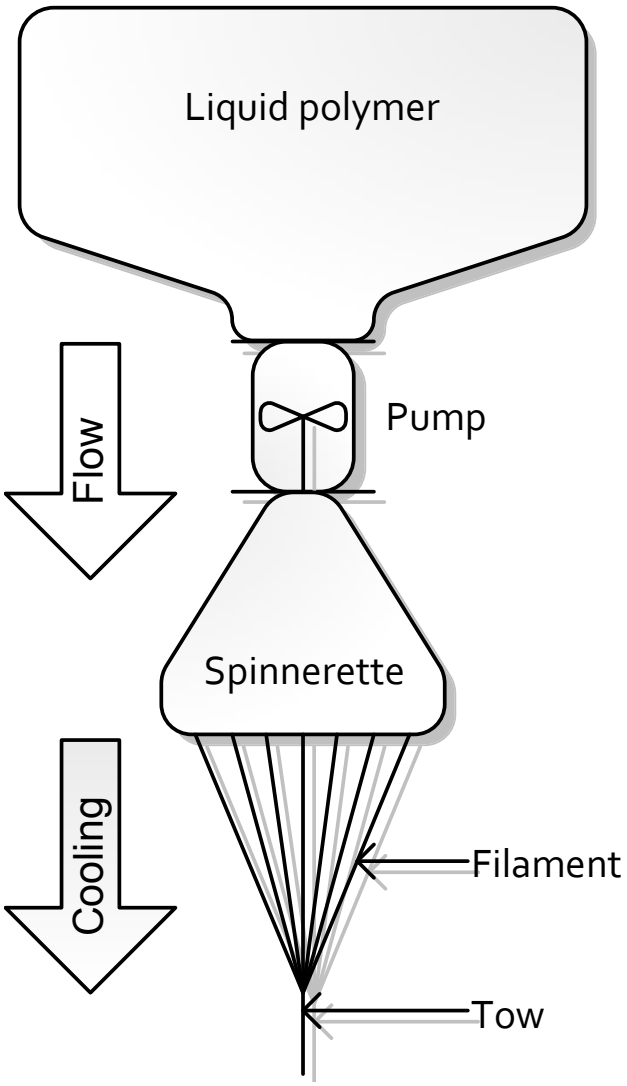

Figure 1.2: Steps in creating synthetic polymer fiber.

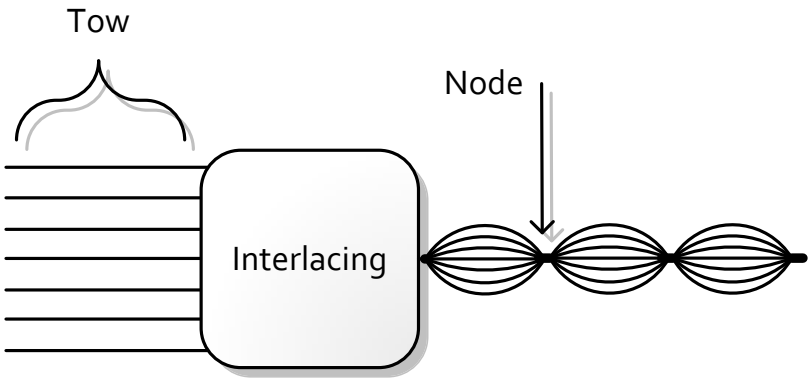

Figure 1.3: Interlacing a tow of synthetic polymer fiber.

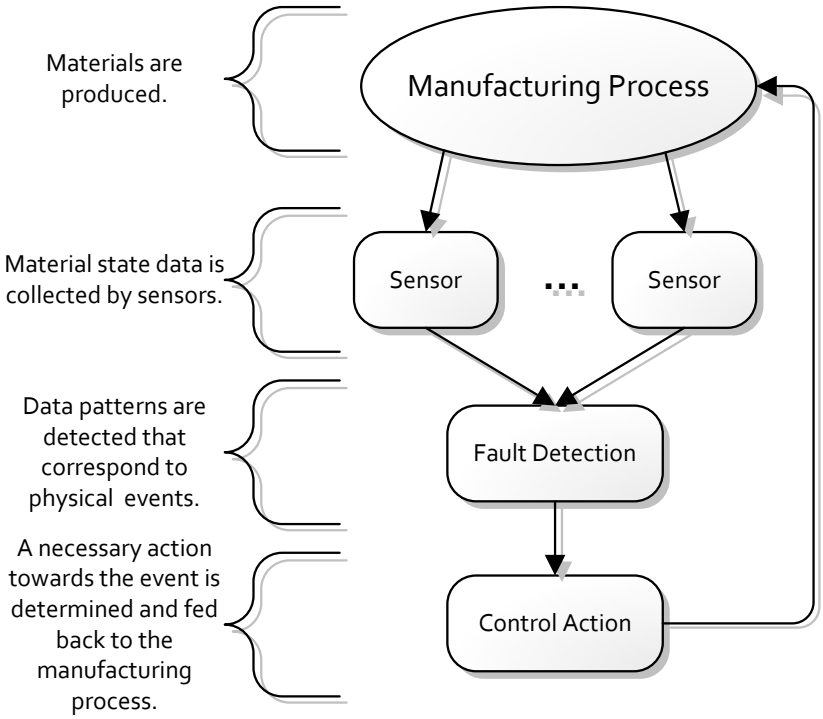

Figure 1.4: Implementing automation with a monitoring and control system.

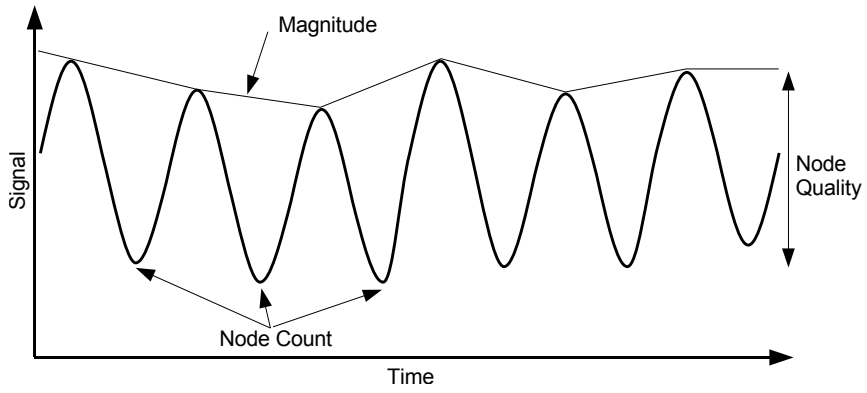

Figure 1.5: How several Attalus fiber properties are calculated from the electrical impedance response signal.

sensor technology. The Attalus sensor measures the electrical impedance of fiber as it passes through an electromagnetic field. This measurement is sensitive to the geometry and electrical properties of both the fiber and finish [6].

Figure 1.5 shows how several fiber properties recorded by Attalus are calculated from the electrical impedance response signal. The *magnitude* reading relates to the measurement of denier. That is, magnitude does not equal denier, but changes in magnitude indicate similar changes in denier. When a fiber node passes over the sensor, the signal drops. From this observation, both *node count* and *node quality* can be calculated. Node count is simply the number of drops in the signal and node quality is the depth of the drop. Another property not illustrated in Figure 1.5 is the *phase* property. Phase is defined as the time delay between the signal waveform and a reference waveform [6]. The phase reading is sensitive to the amount of finish applied to the fiber. This is due to the finish having a higher electrical conductivity to that of the fiber. In this work, these four measurements produced by Attalus are

referred to as *fiber properties*.

In this research, we are interested in enhancing the fault detection module of the monitoring and control system depicted in Figure 1.4. Considerations for the manufacturing process, sensor system, and control action subsystem are not in the scope of this work.

1.2 Problem Statement

To our knowledge, no time series novelty detection algorithms have been applied to data originating from synthetic fiber manufacturing processes. Due to the success of such methods in other fields, it is interesting to determine whether they would apply well to fiber applications. The primary objective of this study is to empirically evaluate several of the most successful time series novelty detection techniques to fiber data, in order to determine which is the most effective. A secondary objective is to empirically evaluate several leading feature extraction techniques on fiber data, again to determine the most effective. Feature extraction is a preprocessing stage of a novelty detection application. Its goal is to reduce data size while keeping important data features intact.

1.3 Methodology

To complete the primary objective we will first review all time series novelty detection techniques presented in the literature. Two of the best techniques will then be selected for further analysis by qualitative comparison. Techniques will be eliminated

based on time complexity, scalability, unsupported issues from the fiber domain, lack of follow up work, nondeterministic properties, flaws in the approach cited in other papers, etc. The techniques will then be implemented to perform tests on real fiber data. In addition to the techniques from the literature, two additional novelty detection techniques will also be implemented. One is based on the distance from the training data to the test data. This serves as a simple baseline on which to compare the other techniques. The second additional technique implements a simple voting scheme on the outputs from all other techniques. These four techniques are then empirically evaluated on data from a real fiber monitoring application. Recommendations as to which technique is the most effective will be based on results from the experimentation.

The secondary objective is accomplished in much the same way as the primary. That is, we first review techniques in the literature, implement them, compare them empirically, and then derive conclusions from the results.

1.4 Main Contributions

Our main contributions in this work are:

- We evaluate several leading feature extraction techniques on fiber property data. It is shown that piecewise linear approximation (a.k.a. segmentation) techniques produce the highest quality features.

- We introduce a new feature extraction technique based on the existing bottom-up segmentation approach. This new technique is shown to produce the highest

quality features for fiber property data sets. It is also general enough to be applied to other domains.

- We perform the first evaluation of time series novelty detection as applied to fiber property data sets. It is shown that a simple Euclidean distance based technique is the best overall when no data compression is used. When compression is used the Tarzan technique [30] performs best.

- We design a framework for constructing, running, and visualizing time series algorithms.

- We develop two novelty detection techniques to support comparison with the techniques in the literature. One is based on the *time aligned Euclidean distance* from the training data to the test data. The second additional technique implements a simple voting scheme on the outputs from all other techniques.

1.5 Outline of This Thesis

Chapter 2 gives an overview of the time series novelty detection techniques presented in the literature. A rationale for the selection of novelty detection techniques to be evaluated empirically in Chapter 5 is also given.

In Chapter 3 we present a detailed empirical comparison of the leading feature extraction techniques. We also present a new feature extraction technique, which is shown to produce higher quality features than all other techniques tested.

Chapter 4 presents the implementation details of the novelty detection algorithms. This includes two sections:

- The design and implementation of a framework used to construct and run time series algorithms.
- The design and implementation of each novelty detection technique using this framework. Two additional novelty detection algorithms not described in the literature are also described here.

In Chapter 5 we perform a detailed empirical comparison of the novelty detection techniques described in Chapter 4. Recommendations are also made on which novelty detection techniques are best for fiber property data sets.

We conclude in Chapter 6 with a summary of our work and its key contributions. We also describe possible future work related to this research.

Chapter 2

Related Work

Before we can evaluate novelty detection on fiber property data, we must review all of the available techniques presented in the literature. Many approaches to novelty detection on time series data have been proposed [3]. Some mimic biological systems such as the human immune system [12] and the human brain [43]. Others apply classic statistical theory to the problem [27]. Still others approach the problem in a more unique way [30, 40, 49, 48].

Out of all the reviewed techniques, two of the best will be selected by qualitative comparison for further analysis in Chapter 5. Techniques will be eliminated based on time complexity, scalability, unsupported issues from the fiber domain, lack of follow up work, nondeterministic properties, flaws in the approach cited in other papers, etc. These two techniques are described in greater detail in Chapter 4 and as a result will only be briefly mentioned here.

2.1 Artificial Neural Networks

Artificial neural networks have been used for many years in the manufacturing industry for monitoring and control [52]. This is mainly because of their ability to learn patterns in data from experience — not from explicit mathematical models of the data. Neural networks are applied in cases where the underlying mathematical models are too complex or too costly to determine by traditional means. They have been used successfully for novelty detection in time series data as well [43]. They can learn and classify the normal data behaviour and therefore distinguish normal from abnormal. For small problems neural networks work quite well. However, they do not scale well to massive datasets [30] and therefore will not be considered further.

2.2 Negative Selection

Dasgupta and Forrest [12] propose a novelty detection technique based on the negative selective mechanism of the human immune system. The human immune system is able to distinguish foreign (i.e., not seen before) cells from normal body cells. It does this by first generating a set of T-cells via a pseudo-random genetic rearrangement process. These T-cells are then exposed to the body's own cells in a training area. T-cells that bond to the body's cells are destroyed. The T-cells that are left match cells that are foreign to the body. This process is called negative selection [13].

The basic idea in applying this to data sets is to somehow generate a set of candidate pattern detectors and then discard those detectors that match the training data. The pattern detectors left over from this process are then used to match novel or

abnormal patterns in the data. One major drawback of this method occurs when the normal data set becomes increasingly diverse. As a data set becomes more diverse, the number of possible patterns increases as well. As a result, more candidate detectors match the normal data and are destroyed. In the worst case, all pattern detectors are destroyed and the algorithm fails [30, 40].

The non-deterministic nature and high complexity [50] of negative selection are not desirable. As a result we will not consider negative selection in our evaluation.

2.3 Outlier Based

A novel occurrence can be thought of as being an outlier of normal occurrence data. Outlier analysis is well established in statistics as well as in the more recent data mining area.

The authors of [27] introduce a scheme whereby optimal histograms are used to identify outliers in time series data. Under this scheme, outliers are defined as "...points with values that differ greatly from that of surrounding points" [27]. In a fault monitoring system a single data point does not give much information about the system state. System state is defined by multiple consecutive data points — a data sequence or pattern. We are most interested in data patterns that deviate from the normal operational data.

As described in [30], sometimes large changes in magnitude at regular intervals are part of the normal operational data. For instance, in a fiber production facility, the process of ending a package of fiber causes a large drop in the signal at regular intervals. In this case the absence of a large drop in signal is a novel event. Outlier

based techniques would not catch this event and are therefore not suitable for fiber property novelty detection.

2.4 Wavelet Based

The authors of [49] take advantage of the multi-resolution property of the wavelet transform to find novel events at several levels of abstraction. For example, a high level of abstraction would ignore short novel events. Their approach uses a wavelet-based TSA-Tree (Trend and Surprise Abstractions Tree) structure to improve the performance of searching for novel events at multiple levels of abstraction. This method defines novelties as having a large difference between two consecutive averages [56]. This limits the detected novelties to being dramatic shifts in the signal. Because of this, short novelties that lie within the normal data pattern cannot be detected [30, 40]. Also, since the definition of surprise gives the same weight to positive and negative magnitude changes, there is no distinction made between them. Obviously for some systems, a high spike is not the same as a low spike. Again, the novelty definition imposes another flaw on this method. As described in Section 2.3, in a fiber production facility the process of ending a package of fiber causes a large drop in the signal at regular intervals. The TSA-Tree algorithm would fail in this case, and therefore it will not be considered for evaluation.

2.5 Frequency Based

Keogh et al. [30] have developed a novelty detection algorithm based on data pattern frequencies. They define a novel pattern as having a frequency of occurrence different from that of normal data. Frequencies of all the normal data patterns are encoded in a suffix tree data structure. To utilize this structure, the data is first discretized into a textual representation. A Markov model is then used to estimate the expected frequency of any new data pattern. A data pattern with a greater frequency than what was expected is considered a novel event. As with any discretization technique, meaningful data may be lost in the transformation [40]. As a result, the discretization technique must be chosen carefully. An interesting property of this technique is its time/space complexity. Once the suffix tree is constructed from the normal data, the process of detecting all novel patterns in a dataset is linear in the size of the dataset. In 2003, a National Aeronautics and Space Administration (NASA) study singled out Keogh's approach as having "*great promise in the long term*" [26]. It is currently being tested by NASA for space shuttle launch monitoring [38]. The success of this technique in studies by other authors [17, 24] and the lack of any major flaws, make it a good candidate for evaluation. It is discussed further in Chapter 4.

2.6 Support Vector Machines

Ma et al. [40] propose a novelty detection method based on support vector machines (SVMs). SVMs are a relatively new machine learning technique applicable to both classification and regression. In this case, support vector regression (SVR) is applied.

The idea behind SVR is to first map the input data via a nonlinear map to a higher dimensional feature space. By applying a simple linear algorithm in the feature space, a nonlinear fit is created in the input space. The use of kernel functions allows all computations to be carried out in the input space, with no explicit computations in the higher dimensional space. This makes SVR as powerful as a nonlinear method (such as neural networks) but much more computationally efficient [21].

Once the input data has been processed by the SVR method, all outliers of the model are considered to be novel events. An interesting feature of this method is its ability to associate a measure of confidence with each novel event. However, the authors admit that the technique has many open issues which require further investigation. Due to the small amount of empirical results [40] on this technique, as well as the absence of any follow up work, we will not consider this technique in our evaluation.

2.7 Learning States that Define Normal Data

Another take on the novelty detection problem is to learn rules from the data that define the possible normal sequences [48]. Any sequence of data deviating from these rules would be considered a novelty. This approach involves three main steps as illustrated in Figure 2.1 (adapted from [48]). The first step is to separate the raw data into clusters or segments. Next, rules that best describe each segment are generated. In the illustration, signal magnitude and slope are the defining features of each segment; other measures may be used however. From these rules, a state machine can be constructed that will only accept normal data sequences. Any sequence that is

not accepted by the state machine is marked as novel. This method was designed to automate the process of populating an expert system's knowledge base. As a result, the rules generated by this method are human readable - a contrast to the other novelty detection methods described above. Having the ability to clearly explain and tweak the underlying behaviour is of great value.

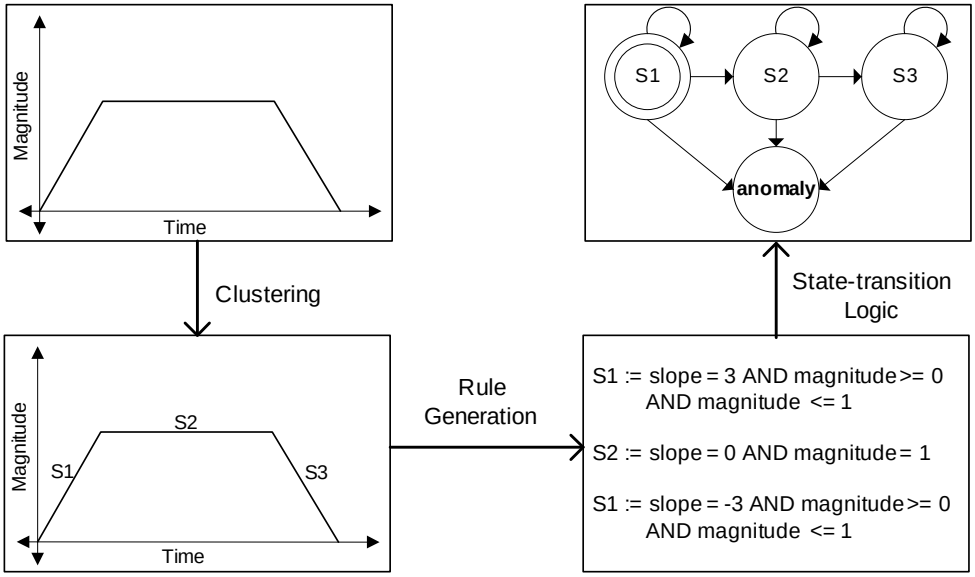

Figure 2.1: Learning rules to uncover novelties

2.7.1 Path Based

Another technique proposed by Mahoney et al. [10], is shown to produce results superior to that of the above technique. It models normal data as a sequence of minimal bounding boxes containing all of the training paths; anomalies are considered data patterns that deviate from any bounding box [11]. We will consider this technique in our evaluation and it will be discussed further in Chapter 4.

2.8 Summary

In this chapter we have reviewed all of the available time series novelty detection techniques presented in the literature. Most of these techniques were eliminated from further analysis based on time complexity, scalability, unsupported issues from the fiber domain, lack of follow up work, nondeterministic properties, flaws in the approach cited in other papers, etc. The frequency [30] and path based [10] techniques appeared to be best suited and so will be evaluated further in Chapter 5.

Chapter 3

Evaluation of Feature Extraction Techniques

The field of time series data mining has seen an explosion of interest in recent years. Researchers in this field are typically faced with two related problems: many data mining algorithms have a high time complexity and time series databases are often very large. Together, these problems make it difficult to use data mining technologies such as novelty detection in time critical systems. The solution is to either reduce algorithm complexity or reduce data size. Much work has been devoted to both problems in the data mining community. In this chapter we focus on the aspect of reducing data size through feature extraction techniques.

Feature extraction is the process of identifying important data features while removing unimportant ones. Features could be actual data points, statistics on several data points, lines, clusters, or even coefficients of functions. The goal is to end up with fewer features than original data points so that data mining algorithms can run

in a reduced amount of time.

Many feature extraction techniques have been proposed for time series data mining, including Fourier/Wavelet transforms [1, 46], Singular Value Decomposition (SVD) [53], Adaptive Piecewise Constant Approximation (APCA) [31], Symbolic Mappings [37, 45], Piecewise Aggregate Approximation (PAA) [32, 55], and Piecewise Linear Approximation (PLA) [33, 34, 44, 25]. In this chapter, we will evaluate the four most popular techniques using fiber property datasets. Our motivation for evaluating these techniques here is to select the best technique for data preprocessing in the novelty detection evaluations in Chapter 5.

The remainder of the chapter is organized as follows. In Section 3.1, we describe the major feature extraction techniques and the optimizations used. In Section 3.2 we empirically evaluate the techniques and show that PLA methods produce the highest quality features. In Section 3.3 we detail our new PLA algorithm and validate its results [4]. We conclude in Section 3.4 with a summary.

3.1 Techniques

Many feature extraction algorithms have been presented in the literature. The purpose of these algorithms is to reduce data size while keeping only the important details intact (i.e., the data features). In this section, we discuss the four most popular feature extraction techniques that have been used for time series data mining. Figure 3.1 shows these techniques in action. We refer the interested reader to works by Keogh et al. [29, 37] for a more extensive survey of feature extraction techniques.

For the purposes of our experimentation, all techniques will need to reduce a time

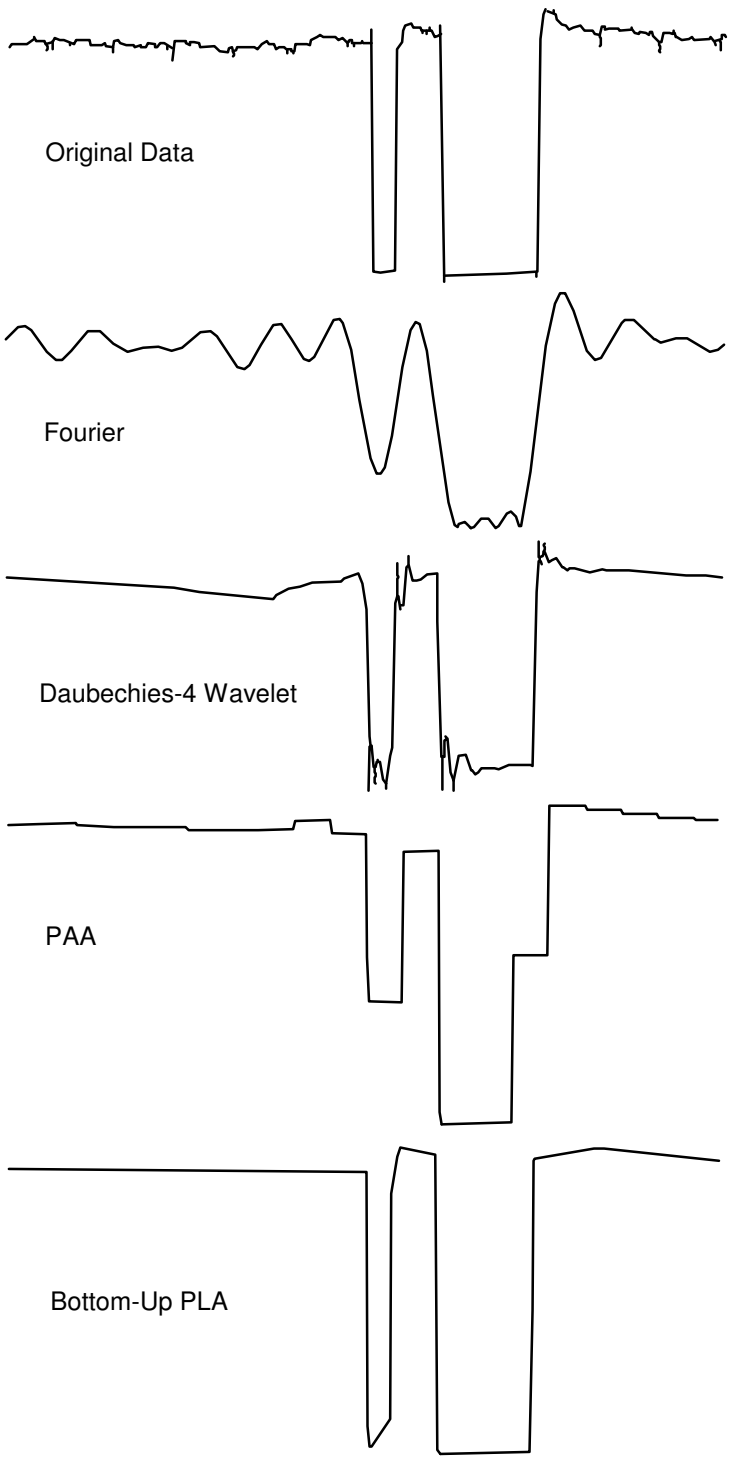

Figure 3.1: Major feature extraction techniques reducing a 512 point sample to 20 points.

ordered series of real values $\vec{x}$ of length n into a vector of features $\vec{X}$ of length N. Data points include anything required to store the compressed representation; this includes actual values, indices, or other bookkeeping data.

3.1.1 Discrete Fourier Transform

The first technique presented to the time series data mining community for data reduction was the Discrete Fourier Transform (DFT) [1]. The DFT breaks down a time series of length n into n sine/cosine waves, which when composed together form the original signal. Each wave is represented by a single complex number, known as a Fourier coefficient.

The DFT of a time series $\vec{x} = [t = 0, ..., n-1|x_t]$ is defined as a sequence of complex coefficients $\vec{X} = [f = 0, ..., n-1|X_f]$, represented by

$$X_f = \frac{1}{\sqrt{n}} \sum_{t=1}^{n-1} x_t \exp\left(\frac{-i2\pi ft}{n}\right)$$

where i is the imaginary unit $i = \sqrt{-1}$ and $f = 0, ..., n-1$. Computing X_f directly is an $O(n^2)$ operation. Fortunately, X_f can be computed in $O(n \log n)$ time by the Fast Fourier Transform (FFT) algorithm [1].

Referring back to Figure 3.1, the Fourier result shown is not a direct plot of $\vec{X}$. All plots shown are time domain signals reconstructed from the feature vector $\vec{X}$ (which, in the case of the Fourier transform, is in the frequency domain).

One may notice that the resulting data size after the DFT is $2n$. This is because each complex number needs two values for storage: one real and one imaginary. By taking advantage of the symmetric property of the DFT [32], half of the resultant coefficients can be discarded. This still leaves us with the same data size as we started

with. The ability to compress data comes from the fact that for most data sets, many coefficients contribute little to the reconstructed signal and can be discarded. Another useful property of the DFT is that the Euclidean distance between two time series in the frequency domain is the same as in the time domain [1]. This allows all data mining operations to be performed in the much smaller frequency domain.

Since the DFT essentially measures the global frequency content of the signal, it does not preserve localized time domain events. That is, elements that occur at a specific time and do not repeat are lost in the transformation. As a result, processing large time series with the DFT is expected to produce poor approximations.

3.1.1.1 Comparison of Compression Schemes

Agrawal [1] describes the process of discarding Fourier coefficients as simply removing all but the first few. However, the lower frequencies that are kept in this scheme may not always be the frequencies that contribute most to the signal. A technique proposed by Mörchen [42] describes keeping only the largest coefficients; this results in the most energy preservation from the original signal. Using this scheme, the resulting coefficient array will most likely be sparse and value indices will need to be stored. This means that fewer coefficients can be stored to make room for indices.

To determine which compression scheme retains a closer fit to the original data, we performed experiments similar to those in Section 3.2 of this chapter. The results of these experiments are shown in Figure 3.2. Clearly, for our fiber property data sets, keeping the first few coefficients retains higher quality features than the technique proposed by Mörchen [42]. This is perhaps due to the higher number of coefficients retained in Agrawal's approach. As a result, we will use Agrawal's approach

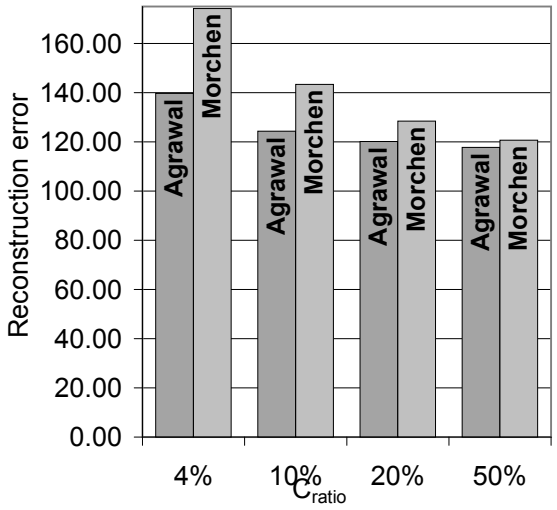

Figure 3.2: Comparison of two compression schemes for the DFT. C_{ratio} and the reconstruction error are defined in Section 3.2.1.

of removing all but the first few coefficients.

3.1.2 Discrete Wavelet Transform

Like the DFT, the Discrete Wavelet Transform (DWT) converts a time domain signal into a frequency domain signal, represented by a set of coefficients. Instead of using sine/cosine waves to reconstruct a signal, the DWT uses many scaled and/or shifted versions of a function called the mother wavelet [42]. Low frequency versions of the mother wavelet model the global contributions of a signal while high frequency versions model local events in a signal [19]. In this way, data features that are localized to a specific time can be represented as wavelet coefficients. This is a contrast to the DFT, which can only model global contributions to the signal. This fact, along with its low time complexity of $O(n)$ [41], has made the DWT a choice feature extraction

technique for many data mining applications [35].

The DWT of a time series $\vec{x}$ results in a sequence of real coefficients $\vec{X}$. Like the DFT, many of these coefficients contribute little to the overall signal and can be removed. Therefore, the compression scheme used for the DFT is also used for all DWT experiments in this paper.

There are many mother wavelet functions to choose from, with each emphasizing different aspects of a signal. The most commonly used wavelet function for the purposes of data mining is the Haar wavelet [54, 46]. Popivanov et al. also note that wavelet functions such as Daubechies [14] may perform better for certain datasets [47]. As a result, in this paper we will use two types of wavelet functions: the Haar and Daubechies-4 wavelets. The Haar transform will be referenced as HWT (the **H**aar **W**avelet **T**ransform) and the Daubechies transform will be referenced as DWT (the **D**aubechies **W**avelet **T**ransform).

3.1.3 Piecewise Aggregate Approximation

The simple, yet powerful, Piecewise Aggregate Approximation (PAA) technique was first applied to time series data mining by Keogh et al. [32] and Yi et al. [55]. It has been used before in other fields under names such as span-based averaging or simply averaging. The basic idea of this approach is to first divide the input sequence $\vec{x}$ of length n into N equal sized segments. N is determined by the amount of data compression needed. The mean of each segment is then used as a data feature in the resultant series $\vec{X}$, which is calculated using the following equation

$$\vec{X} = [mean(\vec{s}_1), ..., mean(\vec{s}_N)]$$

where $\vec{s}$ is the input sequence $\vec{x}$ divided into N equal sized segments and $\vec{s}_i$ is the ith segment. Due to the simplicity of this method, it has a time complexity of $O(n)$. This makes it particularly attractive for large data sets.

3.1.4 Piecewise Linear Approximation

Keogh et al. make the suggestion that Piecewise Linear Approximation (PLA) is perhaps the most used feature extraction technique in time series data mining [33]. The basic idea of this approach is to approximate the input sequence using a desired number of straight lines, which we call segments. Since the number of segments is typically much smaller than n, a high level of compression can be achieved.

PLA techniques can be generally classified into three categories:

- **Sliding Window**: A window is grown until a specified error threshold is reached. Even though this method produces relatively poor results, it is heavily used for its online capability. Online algorithms are able to process data in a piece-by-piece fashion, without having the entire dataset available initially. This is contrasted to offline or batch algorithms which need the entire dataset initially.

- **Top-Down**: Starts initially by approximating the entire time series with one segment. It then recursively partitions the segment until all segments fall within a specified error threshold. Without any modifications, this algorithm processes data in an offline fashion.

- **Bottom-Up**: Starts initially with the finest grain approximation possible (i.e., essentially the original data). It then iteratively merges segments until some error threshold is met. This algorithm also processes data offline.

For all techniques, approximating lines can be calculated in a variety of ways. To keep computational complexity low, in this work, lines are calculated using linear interpolation (i.e., use the first and last points of the segment as the approximating line).

As identified by Keogh et al., no PLA technique is best for all data sets [29]. However, the Bottom-Up approach is generally considered the best overall and is the technique that will be used for all experiments in this paper. It has a time complexity of $O\left(n\frac{n}{N-1}\right)$, where $N-1$ is the number of segments [33]. Pseudo code for the generic **B**ottom-**U**p **S**egmentation (BUS) algorithm is shown in Listing 3.1 (adapted from [33]). In this code listing, $\vec{X}$ is essentially the return value of the `BottomUp` routine (i.e., $\vec{X}$ = `BottomUp(`$\vec{x}$, $maxError$`)`). The `merge(s1, s2)` function returns a new segment consisting of the first point of segment `s1` and the last point of segment `s2`. The `error(s, t)` function returns the distance between an approximating segment `s` and the original time series `t`.

With little modification, the BUS algorithm can accept N as a parameter, instead of $maxError$. While this is desirable, we will use $maxError$ to better support comparison with our new technique in Section 3.3.

Combinations of the three PLA techniques have been proposed before as well. For instance, Keogh et al. have combined the Bottom-Up approach with the Sliding Window approach [33]. This novel technique produced features similar in quality to

Listing 3.1: The Bottom-Up Segmentation (BUS) algorithm

```
TimeSeries BottomUp(TimeSeries T, double maxError)
{
   Segment[] segments;
   double[]  mergeCost;
   int       i;

   for (i = 0; i < T.Length; i += 2)
      segments[i / 2] = new Segment(T[i], T[i + 1]);

   for (i = 0; i < segments.Length; ++i)
   {
      Segment approxSegment = merge(segments[i], segments[i+1]);
      mergeCost[i] = error(approxSegment, T);
   }

   while (min(mergeCost) < maxError)
   {
      i = indexOfMin(mergeCost);
      segments[i] = merge(segments[i], segments[i+1]);
      remove(segments[i+1]);
      mergeCost[i]= error(merge(segments[i], segments[i+1]), T);
      mergeCost[i-1]= error(merge(segments[i-1], segments[i]), T);
   }

   return new TimeSeries(segments);
}
```

Bottom-Up with the addition of online support. Since it still did not outperform Bottom-Up, we will not include it here.

3.2 Empirical comparison of major feature extraction techniques

In this section we will present a detailed empirical comparison of the major feature extraction techniques described in Section 3.1. As stated before, the experimentation is done on data from an industrial synthetic polymer fiber monitoring system; namely, the Attalus^{TM} Fiber System [5]. We will focus on the Magnitude fiber property which corresponds closely with fiber denier. Eleven datasets will be tested ranging in size from 256 to 1024 points. All datasets are normalized to produce a mean of zero and standard deviation of one. These datasets were not chosen arbitrarily; they all contain anomalous patterns of fiber denier. Retaining such anomalous patterns in the reduced representation is a critical goal of this work for the intended novelty detection evaluation. Figure 3.3 illustrates the eleven datasets used.

3.2.1 Methodology

To evaluate the suitability of each feature extraction technique, we will adopt a procedure similar to the one used by Keogh et al. [29]. This procedure essentially measures the reconstruction error for a fixed N, where lower reconstruction error implies higher feature quality. Keogh et al. use the simple Euclidean distance function as a measure of the reconstruction error

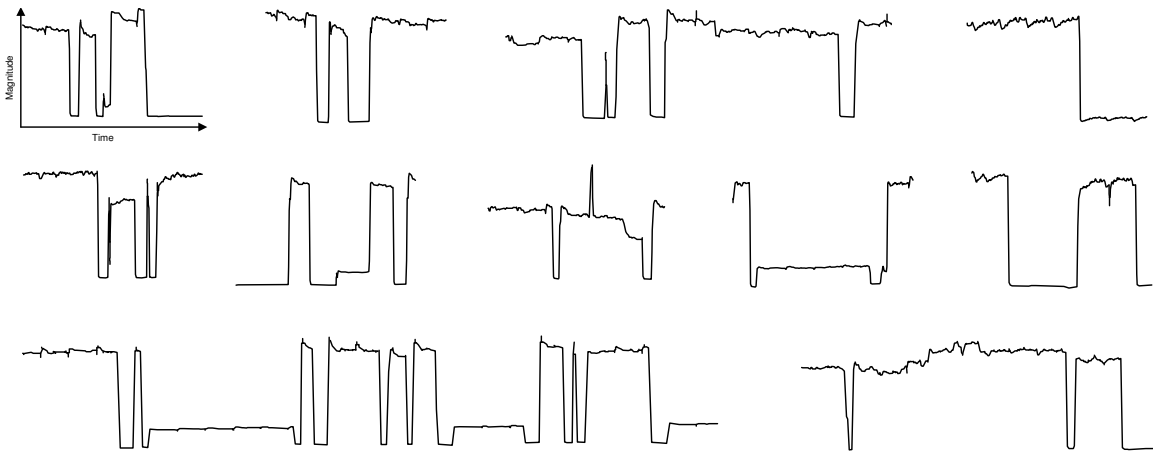

Figure 3.3: The eleven datasets used in the experiments. Each dataset is a recording of the Magnitude data property over time.

$$error(\vec{x}, \vec{X}) = \sqrt{\sum_{i=0}^{n} (x_i - X_i)^2}$$

where $n = N$. However, in all cases when $C_{ratio} > 0$, where $C_{ratio} = \frac{N}{n}$, n is greater than N so this formula will not work. In addition, points in the original and reconstructed time series will not necessarily 'line up' along the time axis. In our scheme, interpolation is used to find the distance between a point and a line in the reconstructed time series. We call this distance metric the *time aligned Euclidean distance*. This is illustrated in Figure 3.4. Note that in this work, the time that a data event occurs is important.

For each value of C_{ratio} tested, a technique must produce N data points. For example, for $C_{ratio} = 20\%$ a $n = 1000$ point sample would be reduced to $N = 200$ points. The various techniques are tuned to produce N resultant data points from a specified C_{ratio} as follows:

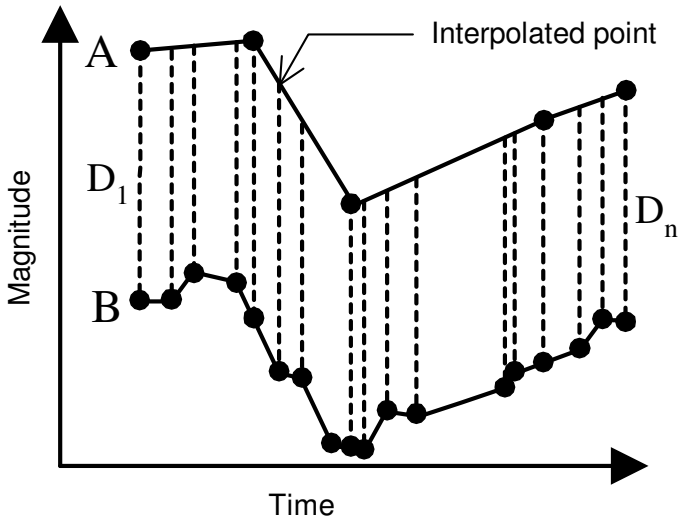

Figure 3.4: The *time aligned Euclidean distance* in the above case would be $\sqrt{D_1^2 + ... + D_n^2}$, where B is the input sequence and A is the reduced representation.

- **DFT**: $N = \frac{C_{ratio} * n}{2}$, the first N coefficients are used.
- **DWT, HWT**: $N = C_{ratio} * n$, like the DFT, the first N coefficients are used.
- **PAA**: $N = C_{ratio} * n$, $\vec{x}$ is divided into N segments of length $\frac{n}{N}$.
- **BUS**: N is the number of resulting data points (i.e., $N - 1$ segments) from the BUS algorithm on a pre-calculated maximum error where $maxError$ is defined as the maximum time aligned Euclidean distance allowed between a segment and the original data. The $maxError$ value that produces N data points is calculated by a separate utility that searches $maxError$ values in increments of 0.0001, starting from zero.

We note that the method of calculating $maxError$ values for the BUS algorithm is computationally expensive. This method is only used to facilitate uniform comparison

with the other feature extraction techniques. Not all techniques support $maxError$ as a parameter, therefore we must resort to such a procedure. In practice, $maxError$ would be set to a fixed value and N would vary accordingly.

Since we are only concerned with the relative quality of a particular technique, we normalize all results by dividing by the worst technique. The technique with the lowest quality for a particular C_{ratio} has the highest reconstruction error.

3.2.2 Results

The results of the experiments are summarized in Figure 3.5. One can readily see that the DFT performs very poorly at all levels of compression. This is most likely due to its inability to preserve the high number of local events in fiber property data.

PAA, DWT, and HWT perform similarly at all compression levels and were generally much better than DFT. None performed better than BUS at any point.

The main result here is that the BUS technique produced the highest quality features at all levels of compression.

3.3 Enhancement to the BUS technique

The results from the previous section clearly show that BUS produced the highest quality features in fiber property data. It also has a reasonable time complexity which allows it to be applied to much larger datasets. This motivated us to find further improvements to the BUS technique.

Upon examination of the features produced by BUS, we noticed that many data points appeared to be redundant. That is, many data points lay on a nearly straight

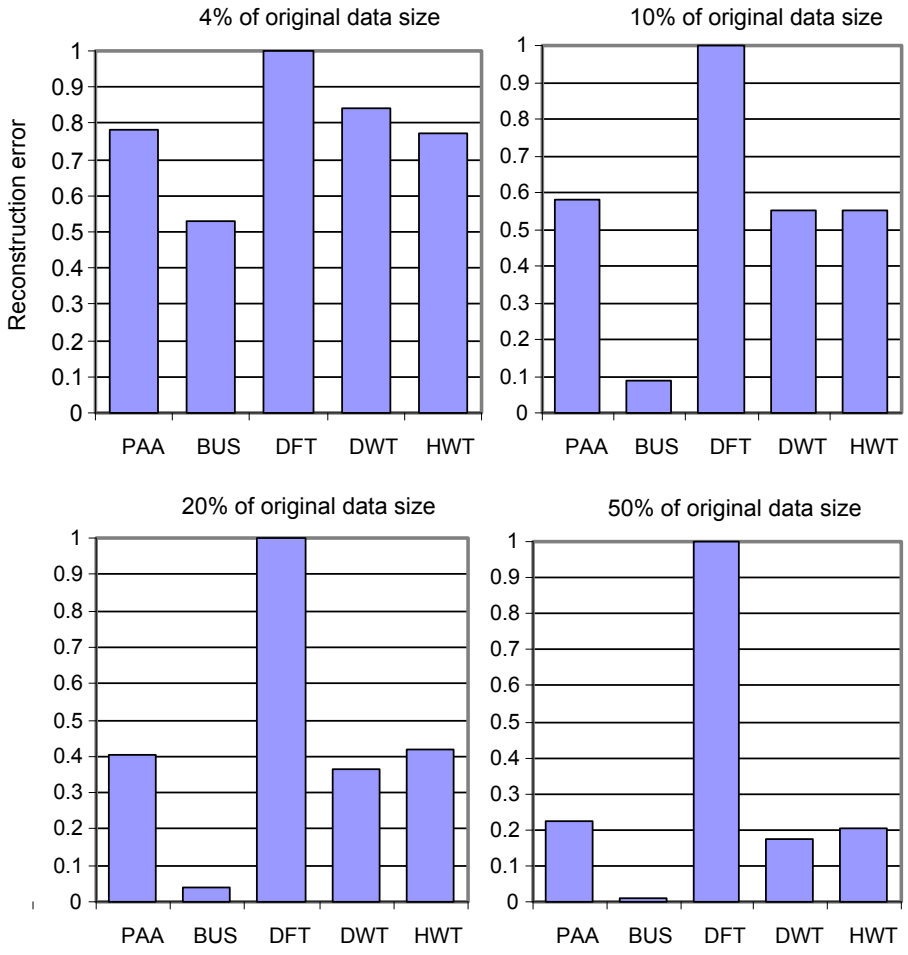

Figure 3.5: A comparison of the major feature extraction techniques on eleven fiber property datasets. Each histogram bar represents the sum of the reconstruction errors for each dataset.

line and thus did not add much value. These points exist because of the BUS algorithm's initial segmentation. Recall that initially, each segment has a length of two and so all merged segments will have an even length. However, a segment may be best represented by an odd number of data points. This was identified as a possible area for improvement by Keogh et al. [33]. To our knowledge no such improvement has been made.

To improve upon this situation we introduce a new algorithm based on BUS that removes these redundant points. We call our new algorithm IBUS (**I**mproved **B**ottom-**U**p **S**egmentation) [4].

3.3.1 The IBUS algorithm

The IBUS algorithm essentially adds a post-processing step to the BUS algorithm. Thus, it first calls the BUS algorithm to determine an initial data segmentation. It then scrolls through the BUS data, discarding points that when removed, keep the error below $maxError$. This can also be described as removing points that lie on a straight line.

This process is implemented by looking at 3 points per iteration, and evaluating whether the approximation formed by the 1^{st} and 3^{rd} points have less error than $maxError$. If the approximation is below the $maxError$ threshold, then the 2^{nd} point is removed from the BUS data. The pseudo code for this procedure is shown in Listing 3.2.

Scrolling through the data takes just $O(N)$ time and if the underlying time series is implemented as a heap, removals take just $O(\log N)$ time [33]. Calculation of the

reconstruction error is constant at each iteration. In the worst case, if every iteration produces a removal, the complexity would be $O(N \log N)$. This case is highly unlikely (if not impossible) because the BUS algorithm would need to produce a straight line with all data points lying on it. It is easily seen that the BUS algorithm cannot produce such an approximation in any non-trivial case. So the actual complexity is expected to be closer to $O(N)$. In addition, since N is much less than n, even in the worst case the additional complexity of IBUS is negligible.

3.3.2 Comparison

We repeated all experiments in Section 3.2, now including the new IBUS algorithm. Since BUS performed best last time, we will compare IBUS relatively to BUS. The $maxError$ for IBUS is calculated in the same way as BUS. So both algorithms may have different $maxError$ values. As shown in Figure 3.6, for our application IBUS produced higher quality features than BUS at every level of compression. These results are normalized values obtained by dividing by the worst technique. The largest difference between the two methods occurred at 4% of the original data size (i.e., the highest compression level). At this level, IBUS had only half of the reconstruction error that BUS had. As the compression level is decreased, the benefits of using IBUS also decreased. It is suspected that as the compression ratio approaches 100% (i.e., no compression), BUS and IBUS will produce identical results.

In further experiments, we compared how much data reduction IBUS achieves over BUS for the same $maxError$. This is necessary to rule out any benefits derived from the $maxError$ searching utility described in Section 3.2.1. This also allows us to show

Listing 3.2: The Improved Bottom-Up segmentation algorithm

```
TimeSeries ImprovedBottomUp(TimeSeries T, double maxError)
{
   TimeSeries resultSeries = BottomUp(T, maxError);

   for (int i = 1; i < resultSeries.Length - 1; ++i)
   {
      Segment approx = new Segment(resultSeries[i - 1], resultSeries
          [i + 1]);

      if (error(approx, T) <= maxError)
      {
         remove(resultSeries[i]);
         i--;
      }
   }

   return resultSeries;
}
```

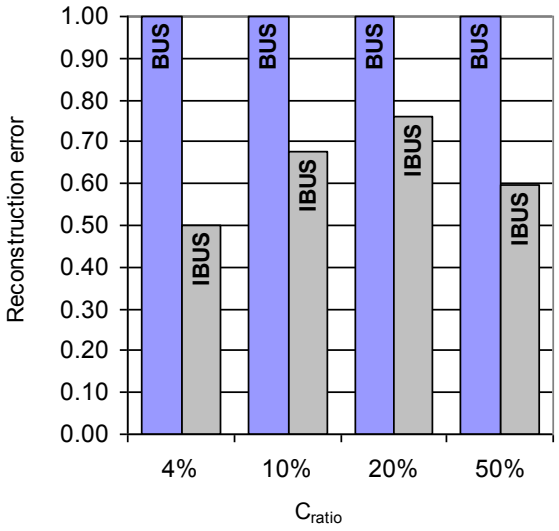

Figure 3.6: A comparison of the relative reconstruction errors of the BUS and IBUS algorithms.

the percent improvement for replacing an already deployed BUS solution with IBUS. The BUS $maxError$ values from the previous experiments are used here for both BUS and IBUS. Figure 3.7 shows the results of these experiments. At each C_{ratio}, IBUS reduces the data size considerably more than BUS. IBUS data size ranged from 61-68% of the BUS data size.

We reiterate that these claims are for fiber property datasets only. However, since no domain knowledge was used in the development of IBUS, it should be applicable to other domains as well.

While it is obvious how IBUS can lower data size by removing redundant points, it may be unclear as to how feature quality can be improved by this process. The improved feature quality comes from that fact that IBUS starts out with a higher quality BUS segmentation (i.e., lower $maxError$) and removes only redundant points to achieve a data size of N. This is contrasted to BUS which needs a higher $maxError$

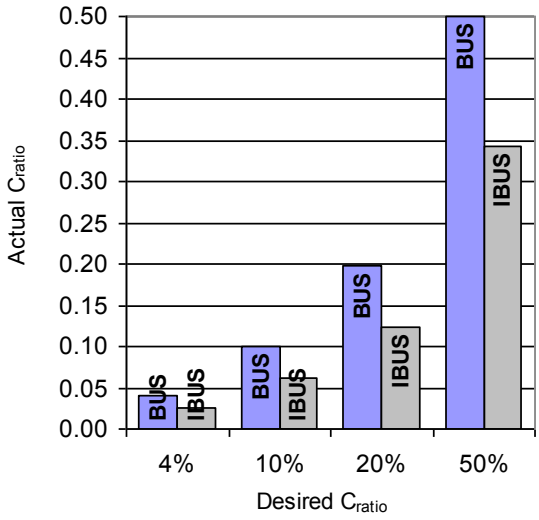

Figure 3.7: A comparison of the BUS and IBUS algorithms with a fixed maxError. In every case IBUS reduces the data size without reducing quality by removing redundant points.

to get the desired N points. The result is that IBUS has similar quality to BUS with a larger N.

3.4 Summary

In this chapter we have shown that PLA methods produce the highest quality features for fiber property datasets. We also introduced a new PLA algorithm (IBUS) which, for our application produced the highest quality features and considerably more data reduction than all currently available feature extraction techniques. As a result we will use IBUS as the feature extraction method for all novelty detection experimentation in Chapter 5.

Chapter 4

Implementation of Novelty Detection Techniques

To evaluate a novelty detection technique some may say you only need to implement the core algorithm. You would quickly realize however that much of the functionality needed to evaluate it is unavailable. You may ask:

- How is the data loaded?
- How can I change the feature extraction method without making a code change?
- How do I visualize the result?
- How do I measure, record, and analyze the results?
- How do I test the many combinations of datasets, novelty detection techniques, and algorithm parameters?
- How well can this be reused in an existing enterprise software system?

While all of these questions could be answered by manual manipulation of the core algorithm's code base, it would take too long with all of the combinations of techniques and parameters. The process of quantitative evaluation needs to be automated.

This chapter encompasses all of the concerns related to implementing the novelty detection techniques in a way that supports quantitative evaluation. In Section 4.1 we introduce a generic time series algorithm testing framework to handle the complex process of evaluation. This framework includes functionality for specifying algorithms, loading data, visualizing results, and recording the necessary statistics for evaluation. Section 4.2 describes how the four novelty detection methods are implemented. Section 4.3 summarizes the contents of this chapter.

4.1 Time Series Algorithm Testing Framework

Before any work was done on the framework its many goals had to be specified. This was done upfront to minimize more time consuming rework later in the process. Section 4.1.1 describes this in detail. Architectural and design decisions that support the goals are described in Section 4.1.2. Most of this framework was orginal work done as part of this research; notable exceptions being the .NET framework libraries and the charting components.

4.1.1 Goals

The major goals of the framework are as follows:

- **Abstract Data Source:** A time series algorithm should not be concerned with where it's data is coming from. To achieve this, data sources should be

abstracted. Abstracting the source of data is also necessary in the case that this work would be used with another existing system (e.g. INSTRUMAR Limited's Attalus Fiber System).

- **Reusable and Composable Processing Steps:** The novelty detection techniques share several processing steps. The framework should support the ability to reuse these processing steps.

- **Low Coupling Between Steps:** To enhance testability, among other things, coupling between the processing steps of an algorithm should be reduced to a minimum. Testability is essential so that the correctness of each processing step can be verified.

- **Dynamic Composition:** During experimentation it is very likely that an algorithm will need to be tweaked to improve results. Several characteristics of an algorithm may need to be changed: the order of processing steps, a single processing step may be swapped for another, or the parameters of a processing step may need to be modified. All of these changes should be possible without any code change.

- **Algorithm Specification Language:** In order to modify an algorithm without changing the implementation, an algorithm specification must be used. This specification file must be human readable and modifiable.

- **Visualization Support:** In many cases analyzing an algorithm's result is very difficult without a visual representation of the results. Results and intermediate steps of an algorithm should be visualized.

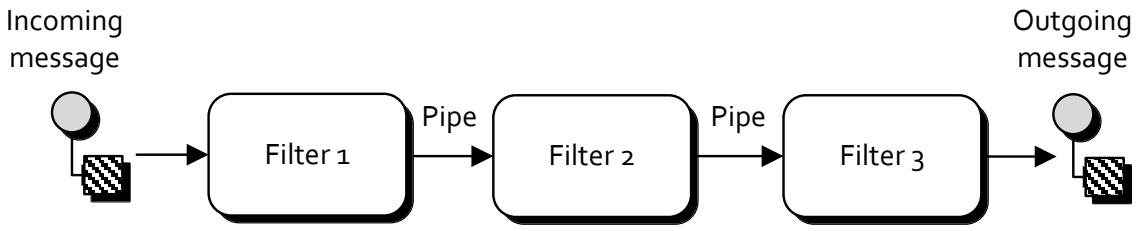

Figure 4.1: The pipes and filters pattern [23, 51].

- **Statistical Analysis Support:** Statistics that will eventually be used to compare the various novelty detection methods will need to be calculated in an automatic fashion. This is due to the sheer volume of possible experiment configurations that will be run.

- **Matrix Run Support:** There must be a way to specify many combinations of datasets, novelty detection techniques, and algorithm parameters.

4.1.2 Architecture

4.1.2.1 Core

To support reusablity, composability, and to reduce coupling between steps a common architectural pattern comes to mind right away; that is, of course, the pipes and filters pattern [23, 51]. In a pipes and filters style architecture, the desired algorithm is broken into series of steps called filters. These filters accept a message, apply some operation to the message content, and pass the modified message to the next filter in the chain. This is illustrated in Figure 4.1.

Implementing this pattern is conceptually simple. Figure 4.2 shows that there are only a handful of entities required. We use an `IFilter` interface to represent

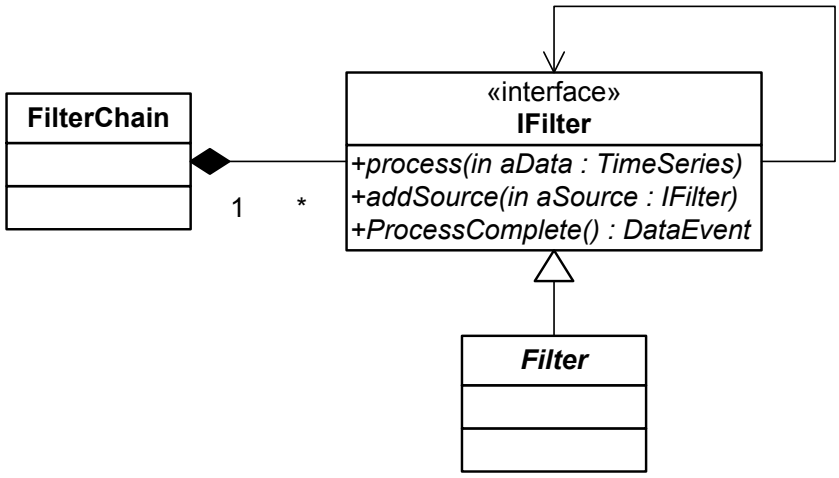

Figure 4.2: The pipes and filters pattern implementation.

our filter. The `process` method accepts a `TimeSeries` and does processing on it. Of course, if this filter is not the only one in the chain, it will need to pass on the resultant `TimeSeries` to the next `IFilter` via a pipe. To keep things simple and fast the implementation uses a C# event, called `ProcessComplete` to act as the pipe between filters. To make an `IFilter` consume from another `IFilter`, the `addSource` method is used. The `FilterChain` class basically has the job of assembling the chain by repeatedly calling `addSource`.

Another very important class mentioned in Figure 4.2 is the `TimeSeries`. As implied by the name, a `TimeSeries` is an ordered collection of time/value pairs.

4.1.2.2 Package Structure

The core classes mentioned in Section 4.1.2.1 are part of the Pipes And Filters Framework (PAFF) package, along with many other supporting classes. Other parts of the testing framework are divided up into separate packages (a.k.a. .NET assemblies).

Figure 4.3 shows all packages and their inter-dependencies. A brief description of each package is as follows:

- **CLoDS (Class Library of Data Sources):** Assembly used to read and write from data sources.
- **DaVinCI (Data Visualization Component Identities):** Assembly containing visualization components.
- **FiTE (Filter Test Executor):** Console application used as a front end for FRy and TRAy.
- **FRy (Filter Runner assembly):** Assembly used to run filter templates against a set of parameters and data and then collect the results.
- **LiFE (Library for Feature Extraction):** Assembly that contains all generic feature extraction filters.
- **PAFF (Pipes And Filters Framework):** Main assembly for the framework. Contains base classes used in most other assemblies.
- **PAFFUI (Pipes And Filters Framework User Interface):** The graphical front-end for the PAFF. Contains basic functionality for opening, saving, editing, and running filter template files. Filter output can be visualized via charts or data grids.
- **PATH:** Assembly implementing the PATH novelty detection algorithm [10].
- **RADS (Reuseable Assembly of Data Structures):** Assembly that contains reusable data structures.

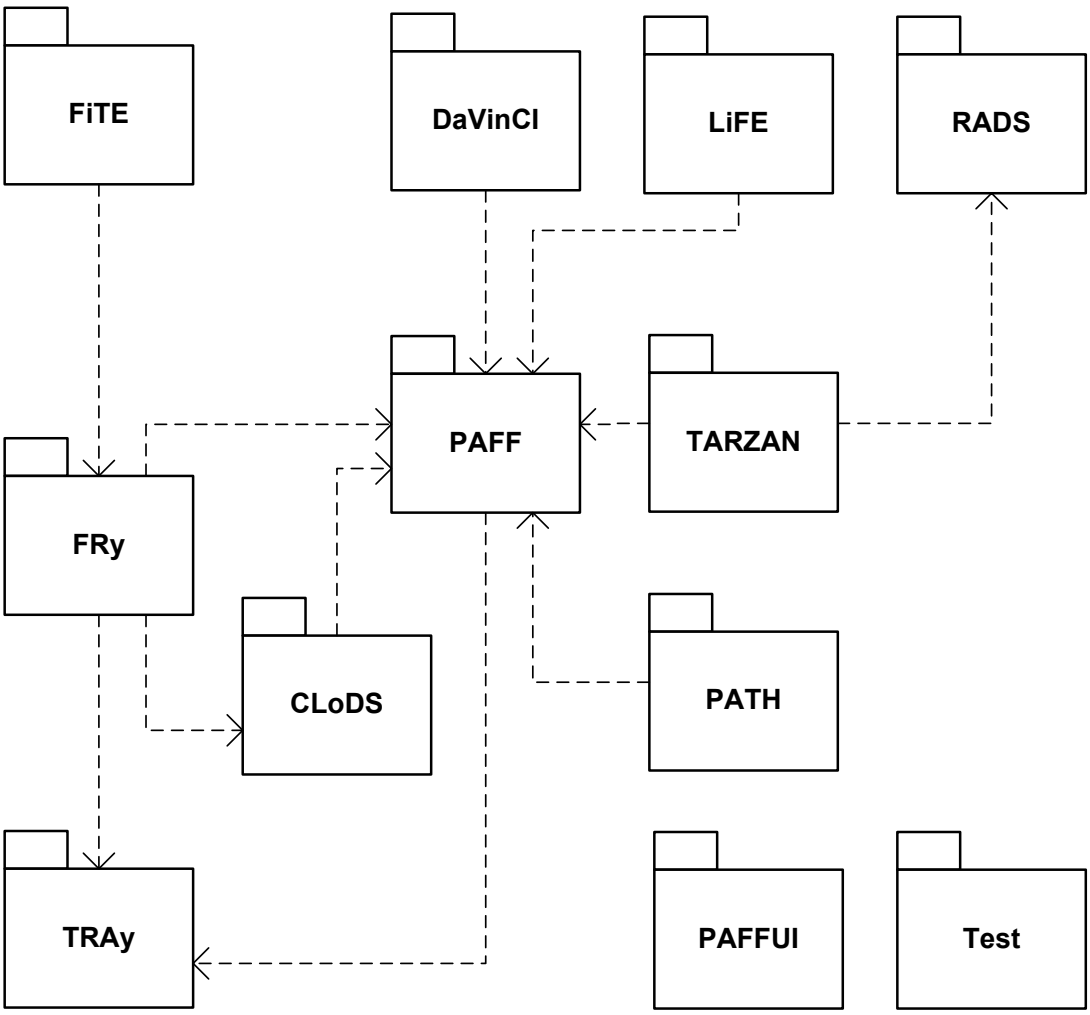

Figure 4.3: Package diagram of the framework.

- **TARZAN:** Assembly implementing the TARZAN novelty detection algorithm [30].
- **Test:** Assembly containing unit tests and supporting mock objects for all other assemblies. Data driven system tests are also contained in this package.
- **TRAy (Template Runner Assembly):** Assembly used to run templates with varying parameters.

4.1.2.3 Data Sources

Abstraction of data sources is somewhat already built into a pipes and filters architecture. A filter doesn't care where its data is coming from, as long as it is a `TimeSeries`. However, to make things a bit easier for dealing with different data sources, the CLoDS package introduces some helper concepts.

Using CLoDS, the start of every filter chain is a `DataSource`. Once started, a `DataSource` reads a chunk of time series data from the actual data source at specified intervals. Some of the concrete `DataSource` implementations include:

- **Synthetic** - Procedurally generated data sources like random and monotonically increasing.
- **File based** - Sources that read data from various file formats.
- **Proprietary INSTRUMAR** - These can't be discussed here but involve connecting to proprietary Attalus data services.

The relationships of these are shown in Figure 4.4.

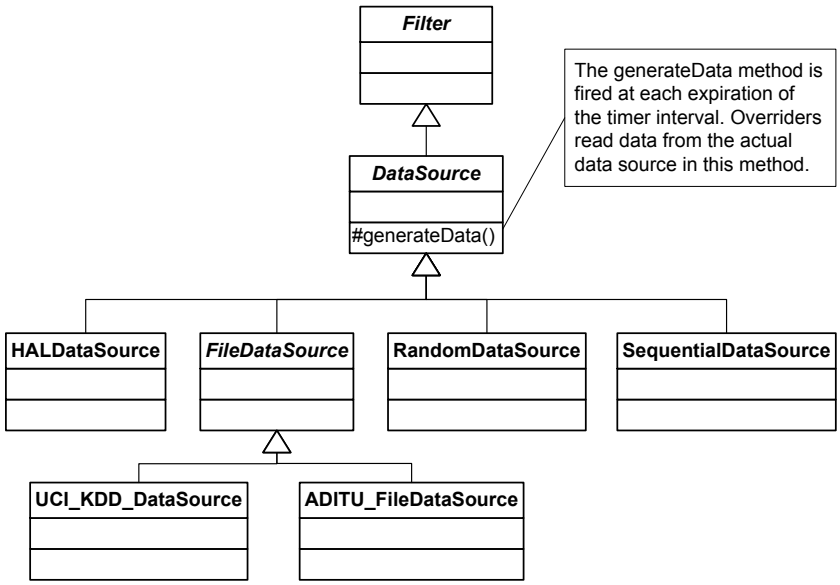

Figure 4.4: Diagram of the CLoDS package.

4.1.2.4 Dynamic Composition

We can create complex algorithms by combining filters that perform specific tasks. But how do we describe how these filters are configured and how their inputs and outputs are connected? Also these mappings need to be specified in a way that is human readable and requires no code change. Rather than coming up with our own Domain Specific Language (DSL) we'll be using a specific XML dialect; we call this the Filter Template File (FTF) format. XML is the obvious choice for the FTF format. XML provides human readability, parsing support in many languages, built in composability, and structure defining languages like DTD and XML Schema. A complete description of the FTF format in provided in appendix B.1.

Composing filters together automatically links the inputs and outputs. In Listing 4.1, filter A receives input from the output of filter B, filter B receives input from the

Listing 4.1: In this example of composing filters, messages would flow from filter D to filter A.

```
<?xml version="1.0" encoding="utf-8" ?>
<filter id="A" assembly="..." type="...">
  <config />
  <filter id="B" assembly="..." type="...">
    <config />
    <filter id="C" assembly="..." type="...">
      <config />
      <filter id="D" assembly="..." type="...">
        <config />
      </filter>
    </filter>
  </filter>
</filter>
```

output of filter C, and so on. Filter D has no composed filter so it is inferred to be a `DataSource`.

The PAFFUI application provides a way to load, edit and run an FTF. Figure 4.5 shows PAFFUI loaded with a file. Any modifications made to the filters in the settings editor can be loaded by clicking the save button.

4.1.2.5 Visualization

The most effective way to visualize time series data is in an X-Y chart. Visualizing the output of a particular filter is as simple as specifying the "vis=Plot" attribute on a particular filter FTF element. Figure 4.6 shows an example chart in PAFFUI.

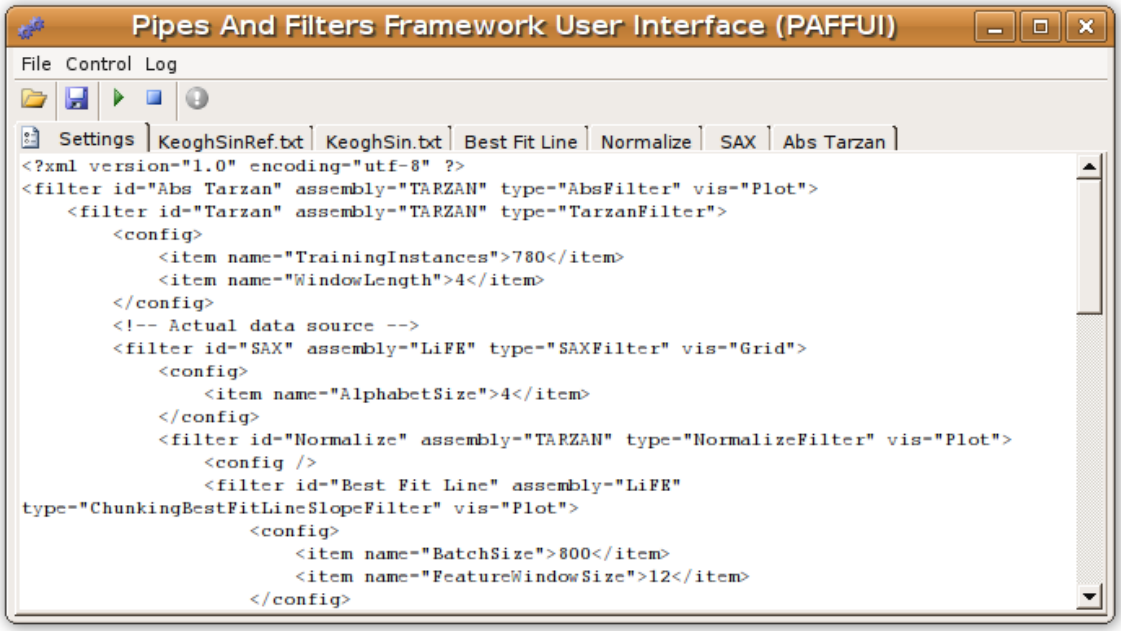

Figure 4.5: FTF configurations can be edited directly in PAFFUI.

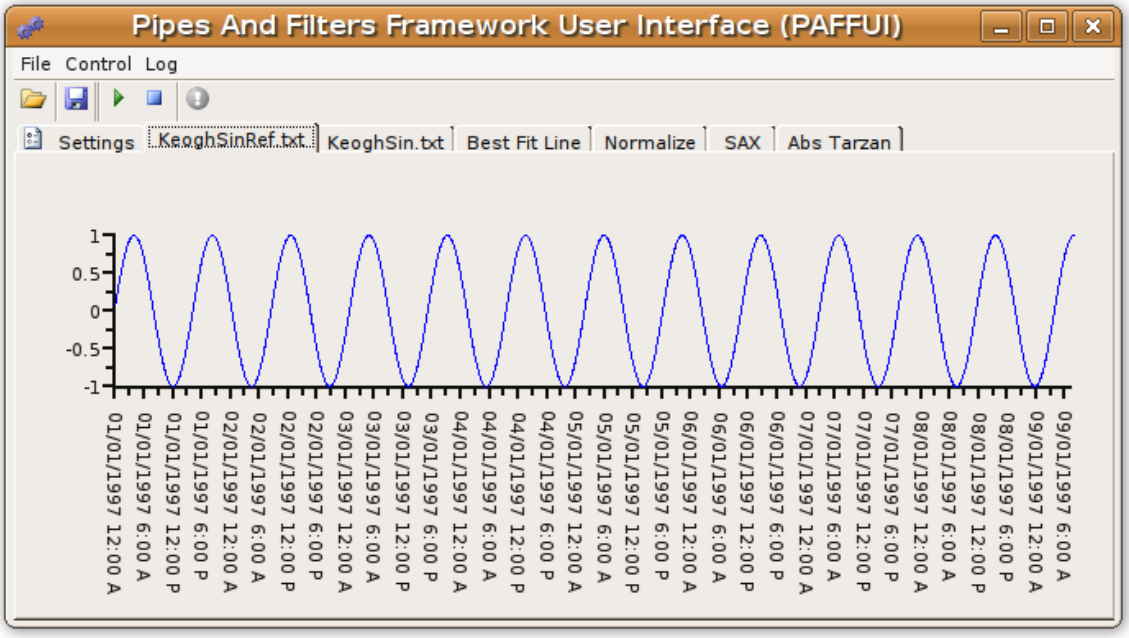

Figure 4.6: A chart of the data at a point in the filter chain.

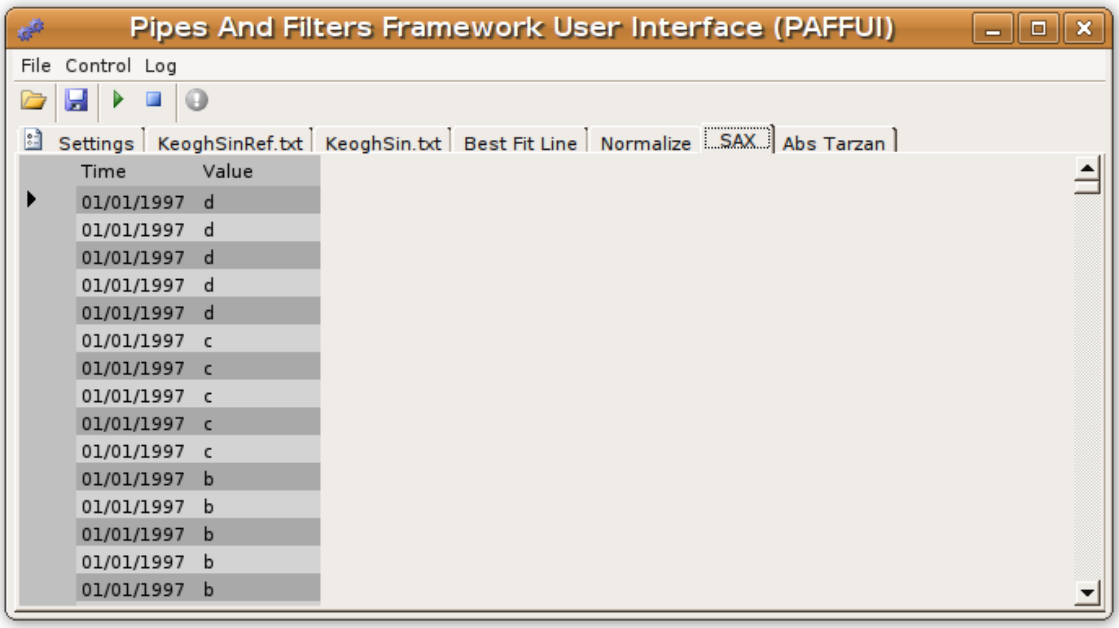

Figure 4.7: A table of the data at a point in the filter chain.

Certain types of filters output non numeric data so a grid visualization component is enabled by specifying "vis=Grid". An example grid is shown in Figure 4.7. This is useful for displaying filters that have a non-numeric output data type.

4.1.2.6 Handling Multiple Consumers

Handling multiple input data streams (i.e. multiple consumers) is a common pattern in novelty detection algorithms. As seen in Figure 4.2, there is no restriction on how many filters one filter can consume from. So the pipes and filters architecture already supports this requirement. The most obvious application of multiple data streams in novelty detection algorithms would be a filter that first learns from training data; one data stream being the training data, the other being the data which we want to test for novelty. The PAFF package provides the `LearningFilter` for this type

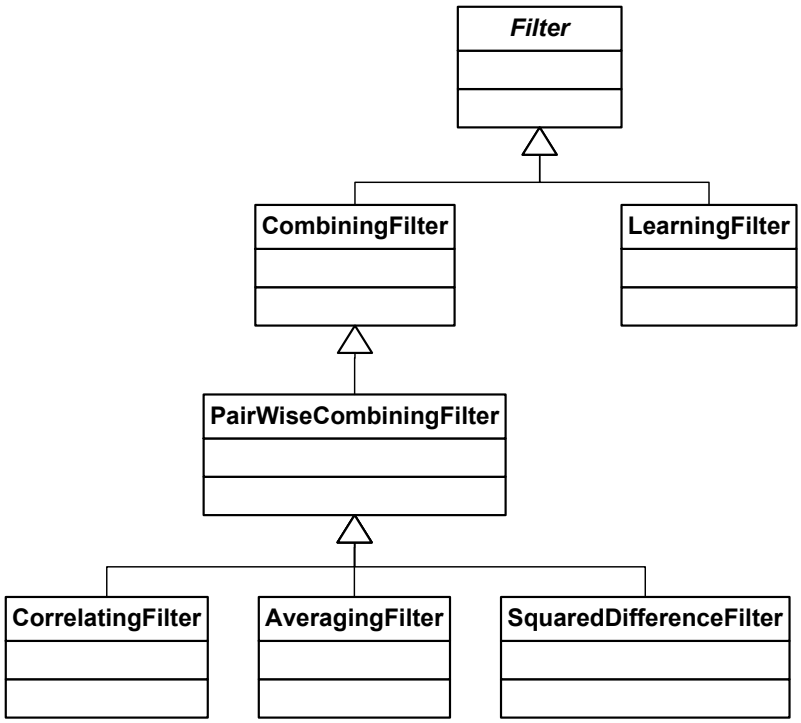

Figure 4.8: Diagram of filters used to combine multiple data streams.

of algorithm. A `LearningFilter` operates on the two data streams in a sequential fashion. It first processes the training data; any incoming test data is queued. It will only process the test data once it has processed enough training data.

It is also useful in some cases to combine multiple data streams concurrently and produce an output stream. The PAFF package provides the `CombiningFilter` for this type of combination.

4.1.2.7 Result Analysis

Due to the volume of experimental configurations used in evaluating the novelty detection algorithms, two goals were devoted to the automation of results analysis:

statistical analysis and matrix run support. The FiTE, FRy, and TRAy packages were added mainly to support these two goals.

It starts with FiTE, a console application that accepts an FTF and a test runner file (TRF). The TRF contains a list of test configurations. Essentially for each test a template FTF is reconfigured for new data, algorithm properties, and expected anomaly regions corresponding to the data. Further detail on this format is given in appendix Section B.2.

To support statistical analysis the FRy package injects filters into each FTF loaded into the framework. These new filters accumulate statistics during operation and write the results out to files after completion. This process also writes out images of each step, fully resolved FTF files, and the original data used for the test run. This gives a complete picture of what happened during the algorithm's execution.

4.2 Novelty Detection Techniques

Chapter 2 provided a broad survey of time series novelty detection techniques from the literature. These techniques were qualitatively analysed and were eliminated based on time complexity, scalability, unsupported issues from the fiber domain, lack of follow up work, nondeterministic properties, flaws in the approach cited in other papers, etc. In this section we will describe in greater detail two techniques that were not eliminated in this analysis and additionally two simple techniques that are not discussed in the liternature.

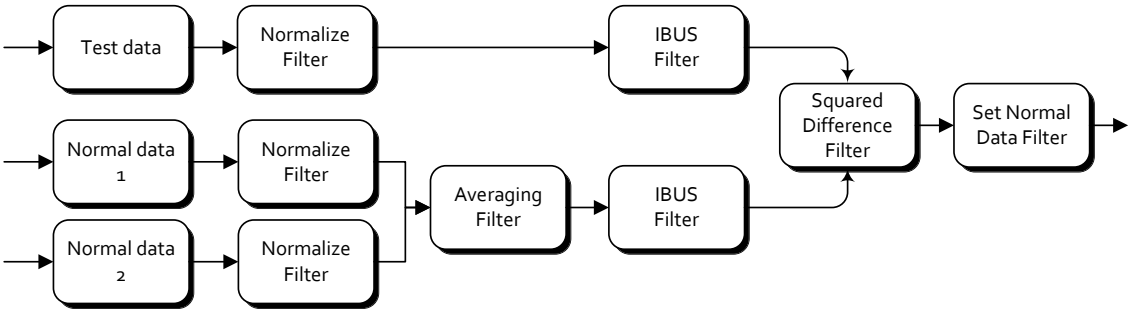

Figure 4.9: Filter composition for the time aligned euclidean difference algorithm.

4.2.1 Time Aligned Euclidean Difference

The time aligned euclidean difference mentioned in Section 3.2.1 can also be used as a simple measure of novelty. In this case, the distance between a normal data stream and a test data stream becomes the measure of novelty. The main reason for introducing this technique is to provide a simple baseline from which to compare the other more complex methods. The filters required for this algorithm are shown in Figure 4.9.

The fiber property data streams have differing offsets and amplitudes, resulting from environmental conditions and fiber composition. These differences are of little concern for our detection algorithm - they are expected characteristics of the data. It is also well known that comparing time series data with different offsets and amplitudes is meaningless [29]. To get around this, we normalize all incoming data to have a mean of zero and standard deviation of one.

While not strictly necessary for the difference algorithm, it was found that the average of two normal data streams was a better model of normal than just a single data stream. As we discovered in the previous chapter, IBUS is the best feature

extraction technique for fiber property data. As a result we will use it to extract features from all incoming data streams.

Points in the time series will not necessarily line up along the time axis, which is required to find the difference. To get around this we use interpolation to find the distance between a point and a line in the reconstructed time series. This is illustrated in Figure 3.4 from Chapter 3.

By looking at the output stream of the difference algorithm we can easily see what areas are possible anomalies. However, to automatically flag these areas as anomalous without having to view the output ourselves, we use the `SetNormalDataFilter`. This filter only keeps data points that differ from the mean by more than a configurable number of standard deviations; any other data points are set to some configurable value.

4.2.2 Frequency Based

As mentioned in Section 2.5, Keogh et al. have developed a novelty detection algorithm based on data pattern frequencies [30]. As illustrated in Figure 4.10, their algorithm flags a pattern as novel if it occurs more or less frequently than the norm. This approach seems simple enough, except for two problems that would need to be solved to make it feasible:

1. How do we compute pattern frequencies in a time efficient manner?

2. How do we calculate the expected frequency of any pattern?

To compute pattern frequencies in a time efficient manner, the authors borrow a concept from string processing; that is, they use a suffix tree data structure. The

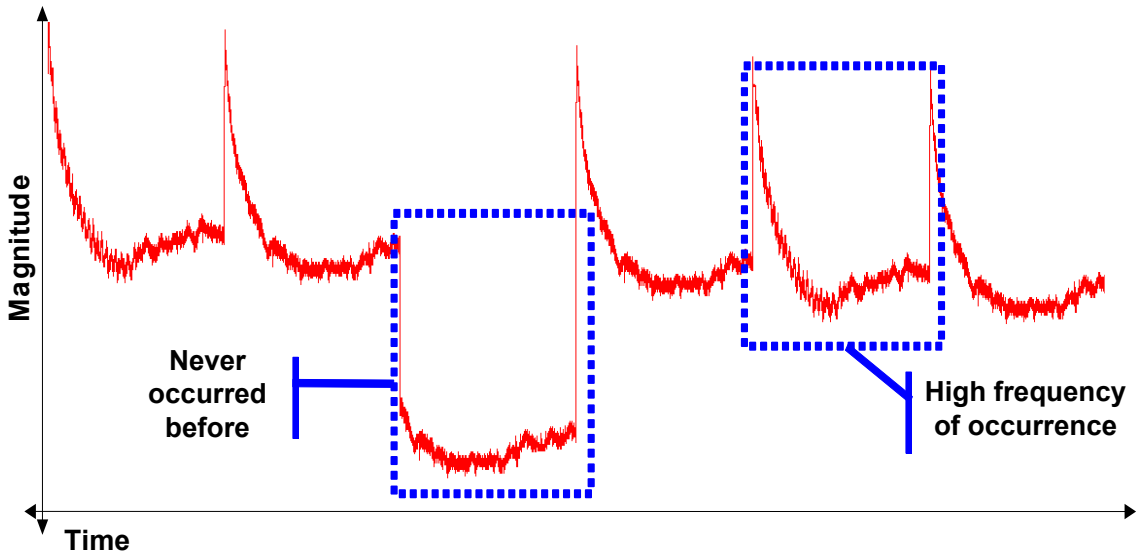

Figure 4.10: A pattern is novel if it occurs more or less frequently than the norm.

suffix tree data structure allows one to count the frequencies of all patterns in a string in $O(n)$ time, and allow for retrieval of that count in $O(1)$ time [20].

But in order to utilize the suffix tree data structure the data has to be converted into a string with a finite alphabet. Real time series values are mapped to this alphabet such that each letter has equal probability. This process of converting a stream of real values into discrete values is called discretization. In our case the discretization algorithm used is called SAX (Symbolic Aggregate approXimation) [37]. An example of how a SAX composed time series is loaded into a suffix tree is shown in Figure 4.11 (adapted from [37]).

In order to calculate the expected frequency of any pattern the authors utilize Markov chain theory. Essentially, a Markov model allows you to calculate the probability of encountering a pattern when you only know the frequency of its sub-patterns. Now, calculating these Markov probabilities is a very expensive operation. For a given

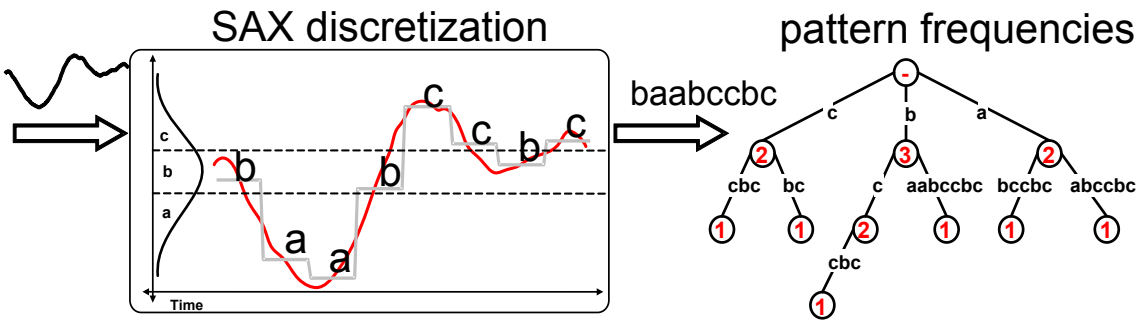

Figure 4.11: We first discretize the normal data into symbols using SAX [37]. Then, a suffix tree data structure can be used to count the frequency of any data pattern.

sequence, you need to count the frequencies of each substring. Since we are loading all pattern frequencies into a suffix tree, we can still complete this process very quickly.

Using these concepts we can form a high level view of how the algorithm calculates surprise (a.k.a. novelty) for a given time series x:

$$surprise(x) = |freq_{test}(x) - E(x)|$$

where $freq_{test}(x)$ is the frequency of time series x in the *test* data set (not the reference dataset that is considered normal). This frequency is retrieved from a suffix tree created from the *test* data set. $E(x)$ or the expected frequency of x is defined as:

$$E(x) = \begin{cases} freq_{ref}(x) & \text{, if } x \text{ occurred in the } ref \text{ data set} \\ E_{Markov}(x) & \text{, otherwise} \end{cases}$$

where $freq_{ref}(x)$ is the frequency of time series x in the ref data set. The ref data set contains the data that is considered normal. Like the test data, this frequency is retrieved from a suffix tree created from the ref data set. $E_{Markov}(x)$ is the expected frequency of occurrence of the pattern x using a Markov model and is defined as:

$$E_{Markov}(x) = \frac{\prod_{i=1}^{m-M} freq_{ref}(x_{[i,i+M]})}{\prod_{i=2}^{m-M} freq_{ref}(x_{[i,i+M-1]})}$$

where m is the length of x and $x_{[i,i+M]}$ is a substring of x starting at position i and ending at position $i + M$. M is defined as the order of the Markov chain, which in our case means the largest value of M constrained by $1 < M < m - 1$ such that all subsequences of x of length M exist in the reference data set (i.e. $freq_{ref}(x_{j,j+M}) > 0$). The full derivation of the E_{Markov} calculation can be found in [30].

From this point the algorithm is very simple, it just needs to scroll through the data using a sliding window approach, and calculate a surprise value for each pattern x using the $surprise(x)$ function. Keogh et al. refer to this as the Tarzan algorithm [30].

4.2.2.1 Tarzan Algorithm Composition Using PAFF

As shown in Figure 4.12, the filter layout looks very similar to the time aligned euclidean distance technique. We make use of the same input filter chain of normalizing filters, averaging reference data sets and extracting features with IBUS. Data from IBUS is then fed into the `SAXFilter` which discretizes the incoming data into a finite alphabet such that each letter has equal probability. Next this character stream is fed into the `TarzanFilter` which comprises the Tarzan algorithm we discussed in the previous section. Finally, like the distance technique, the `SetNormalDataFilter` is used to keep data points that differ from the mean by more than a configurable number of standard deviations; this way, we will only see the surprising values in the output stream.

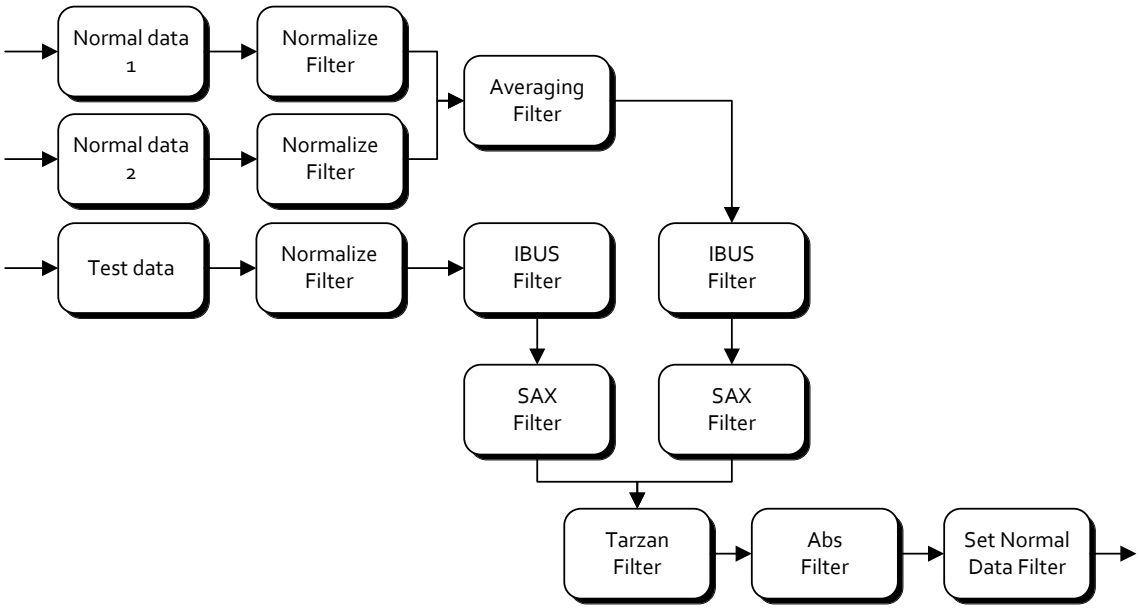

Figure 4.12: Filter composition for the TARZAN novelty detection algorithm.

4.2.3 Path Based

Another technique that we evaluated is one demonstrated by Mahoney et al. [10]; we briefly mentioned this technique in Section 2.7.1 previously. This technique models training data as a set of trajectories or paths and defines data patterns that deviate from any bounding box contained within these paths as novel. To be clear, the paths are essentially just PLA representations of the orginal data.

The technique described by Mahoney et al. [10] gives a method for reducing an input training data set into a PLA representation. To smooth out the input data it also employs a low pass filter prior to segmenting. We do not use these functions as described in Mahoney's work; we will be using IBUS as described in chapter 2 since it was shown to produce the most accurate representation of fiber property data sets.

Once the data has been reduced into a PLA representation, the algorithm does

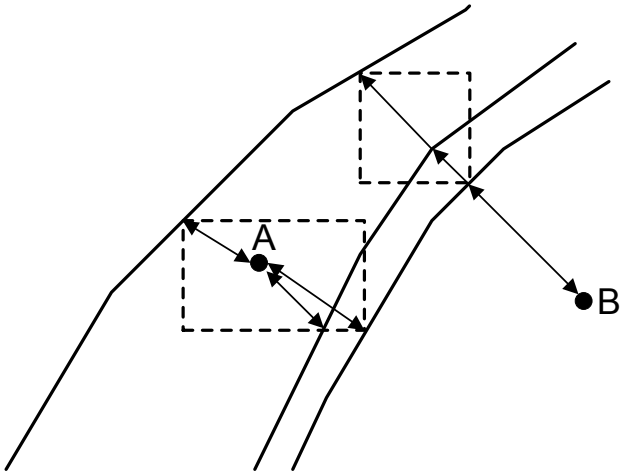

Figure 4.13: Point A is within the bounding box defined by the 3 points in the training paths closest to it so its novelty is zero. Point B is outside the bounding box formed by the 3 closest points so its novelty is square of the Euclidean distance to the bounding box.

an additional step by evaluating $M-1$ derivatives at each data point. So say $M=3$ and x_Y is the Yth derivative of x, x_0 would be the input value, x_1 would be the slope at the point x, and x_2 would be the curvature of the line at point x. These data features will be used in the comparisons to determine novelty. All these data features are also scaled to fit a unit cube, such that each value will range from 0 to 1. By taking derivatives in this fashion, there is essentially a new set of training paths created for each Yth derivative.

After the training paths are saved in a PLA representation and the derivatives are taken, each test data point needs to be compared to the training data. To do this comparison, a bounding box is created that encloses the nearest point on each path to the test data point. If the test data point is within this bounding box, then the

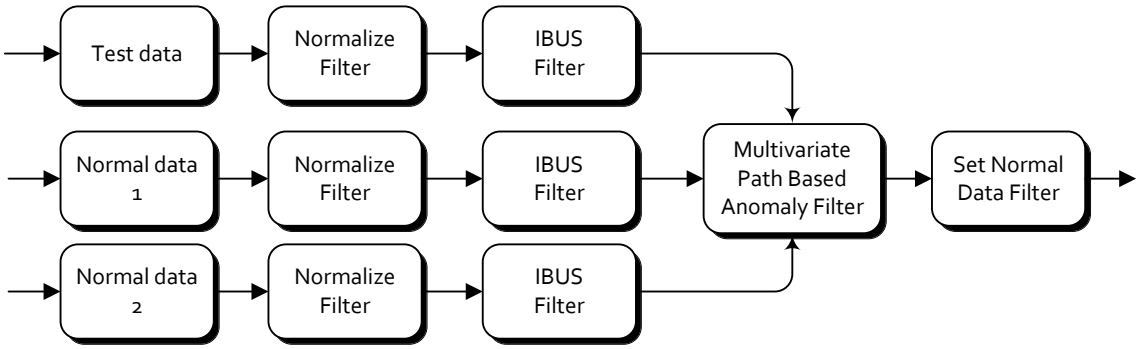

Figure 4.14: Filter composition for the Path based novelty detection algorithm.

novelty is zero. Otherwise, the novelty is the square of the Euclidean distance to the bounding box. This is illustrated in Figure 4.13 (adapted from [10]). This is repeated for each derivative. So, the training paths in effect define the normal level, slope, and curvature of each point.

4.2.3.1 Path Algorithm Composition Using PAFF

The Path algorithm filter layout is the similar to the previous two techniques discussed. The main difference to note is that in the Path algorithm we do not average multiple training data sets into one single set - we feed both training sets directly into the `MultivariatePathBasedAnomalyFilter`, which comprises the actual Path algorithm. We did this because the Path algorithm relies on having multiple training sets to form bounding boxes from. If there was only one training set, the results would be very similar to the plain Euclidean distance.

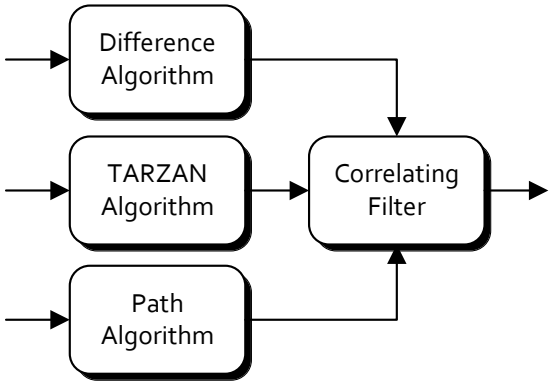

Figure 4.15: Filter composition for the Hybrid novelty detection algorithm.

4.2.4 Hybrid Approach

In addition to the three other techniques, we designed a simple voting algorithm that combines the outputs of all other techniques. This was done to overcome the deficiencies of any single novelty detection technique and to increase the likelihood that a particular detection is correct. The filter composition of this algorithm is shown in Figure 4.15.

The `CorrelatingFilter` inherits the same interpolation ability described in Section 4.2.1 so the differing outputs from each technique are no problem. The filter will output a value of 1 (i.e. anomaly detected) if a specified number of techniques also report the segment as anomalous. The `CorrelatingFilter` uses a configurable threshold to determine if a technique is reporting an anomaly. All of our techniques are using the `SetNormalDataFilter` described in Section 4.2.1 so a single threshold can be used for all inputs.

4.3 Summary

In this chapter we have described the evaluation requirements that needed to be met for the novelty detection experimentation in Chapter 5. We have also briefly shown the testing framework that was created to fill these requirements. Finally, we have described how each of the novelty detection techniques is built up from concepts in this testing framework.

Chapter 5

Evaluation of Time Series Novelty Detection Techniques

Qualitative comparison of a set of algorithms is useful to quickly eliminate those that would not work for a particular domain. Such a comparison is given in Chapter 2 to determine which time series novelty detection methods from the literature is applicable to the fiber domain. Two techniques from the literature stand out as possible solutions based on this comparison. This chapter presents a detailed quantitative comparison of these two techniques, as well as others presented in Chapter 4.

In order to do a fair quantitative comparison of the novelty detection techniques, we will focus on a single class of anomalies from the fiber domain. This is not the same as looking for a particular pattern as in pattern recognition or similarity search; we are merely selecting a particular class of anomalies to run experiments on.

The rest of this chapter is organized as follows. Section 5.1 describes the class of anomalies that we will be testing for. Section 5.2 describes the data sets used,

the statistics recorded, and provides some experimental descriptions. Results from these experiments are presented in Section 5.3. Section 5.4 concludes the chapter with recommendations on the techniques best suited to fiber applications.

5.1 Type of Novelties

In a fiber manufacturing plant there are several levels of fiber grouping. The lowest level, as described in Section 1.1.1, is that of fiber filaments being grouped (conceptually as well as physically) into a tow of fiber. Monitoring generally occurs at the tow level, where each tow is interrogated by a sensor after the post processing steps are complete. There are a set of data streams produced from each monitored tow.

A second level of grouping occurs when several tows (three in our case) are produced on the same machine. As a result, they have similar environmental properties and generally produce similar data streams. We will focus on the anomaly resulting from one tow's data stream deviating from the rest. A large deviation indicates that something has gone wrong with the monitored tow. In this way, tows other than the one being tested are considered to be the normal operational data. This is of course not always the case since many anomalies occur over all monitored tows. For all experiments in this chapter, we will focus on the class of anomalies where one tow's data stream deviates from the rest.

5.2 Experimental Setup

Great care must be taken during an empirical comparison to prevent the introduction of data, implementation or other bias [29]. Data bias can be defined as the process of using only data which supports the desired outcome. This may be done intentionally or without the researcher even knowing. In fact, Keogh et al. [29] shows an example where three techniques are tested and all are shown to be the best by choosing data sets that benefit a particular method. So, when deriving results from specific data sets, great care has to be taken to not claim too much. Since this research is applied to a specific application, the results will be for this application area only. Different data sets would certainly produce different results.

Implementation bias occurs when the quality of code in two techniques under comparison varies greatly. This can also occur intentionally or unintentionally to produce a desired outcome. For example, if an individual proposes a new $O(n(\log n)^2)$ feature extraction algorithm and compares it against the $O(n^2)$ implementation of the DFT, it would not be a fair comparison. The claim of the technique being faster would be incorrect because there is a $O(n \log n)$ implementation of the DFT. In the case of the following novelty detection experiments, we will only empirically compare the results of algorithms as described in Section 5.2.2. The running time will not be compared. This is done to eliminate the possibility of implementation bias. That said, since the end result of this work is to be applied to a real time system, time complexity is very important. All novelty detection techniques under study perform in $O(n \log n)$ time in the worst case, some are much better than this.

Another difficulty in testing novelty detection algorithms is defining what novel is.

Novelty is highly subjective, one person may consider a pattern as novel while another may not. In the novelty detection literature, one method of defining novelties is by asking a domain expert to classify a data set by hand [48]. In this way the results are based on the assumption that the domain expert is the best that any technique can get. This is actually not true because humans can and do miss important events. With that said, we will still adopt the testing procedure as used in other novelty detection work so the results have a useful scale of 0 to 100%.

Each experiment involves several steps:

1. Train the technique on two training data sets representing normal data.
2. Run the technique on a third data set which contains a novel pattern. In our case, this is the data stream deviating from the other two.
3. Calculate statistics based on the result obtained from the technique and what was expected.

5.2.1 Data

There are thirteen data tests in all. For each data test, there is a test data set which is fed into a trained anomaly detection technique. Techniques are trained on two training data sets which represent the normal behaviour. As described in Section 4.1.2.6, multiple training sets are in some cases merged into one training set. The test data sets are the same as used in Chapter 3 for the feature extraction experimentation. In that chapter, only eleven data sets were mentioned (the ones using the Magnitude fiber property). In this chapter we include data sets using the Node Quality and Node Count fiber properties as well.

There is also a separate file that defines where the anomalous regions are located. For example, the following *anomaly definition file*

```
<anomalies>
   <anomaly start="01/11/2004 10:11:30 AM" end="01/11/2004 11:05:30 AM"/>
   <anomaly start="01/11/2004 11:09:50 AM" end="01/11/2004 11:13:10 AM"/>
</anomalies>
```

defines two anomalous regions as identified by a domain expert. These anomaly definitions combined with the output of a particular algorithm allow us to calculate the measurements in Section 5.2.2 in an automated fashion.

Appendix C shows each data set with the anomalous regions highlighted.

5.2.2 Recorded Statistics

The techniques are compared based on two metrics: rate of detection and rate of false alarms. The rate of detection refers to the rate that a method successfully detects a novel data pattern during a particular time frame. The rate of false alarms refers to the rate that a method detects data patterns that are not particularly novel. In order to calculate these two properties we first need to define some data sets that will be used:

- *train*: The data set that the novelty detection algorithm is trained on.
- *test*: The data set that is input into the novelty detection algorithm for analysis. The output of the algorithm is *actAnomaly*.

- *actAnomaly*: The data set that represents what the novelty detection algorithm considers novel. Any value above zero in this data set is considered to be a novelty. An algorithm associates a higher degree of novelty with a higher value in *actAnomaly*.

- *expAnomaly*: The data set is derived from an anomaly definition file. Any anomalies defined in the file are represented as a value of one in the *expAnomaly* data set. All other areas have a value of zero.

From these data sets, a number of data point counts are calculated. These counts are used to determine the rate of detection and rate of false positives. They are illustrated in Figure 5.1 and defined as follows:

- C_t (total count): The total number of data points in the *actAnomaly* data set.

- C_e (expected anomaly count): The number of *actAnomaly* data points that fall within the anomaly regions defined in the *expAnomaly* data set.

- C_a (actual anomaly count): The number of *actAnomaly* data points that are greater than zero.

- C_d (detection count): The number of *actAnomaly* data points that fall within the anomaly regions defined in the *expAnomaly* data set and are greater than zero.

From these counts we can readily obtain values for the rate of detection and rate of false positives. The rate of detection is defined as

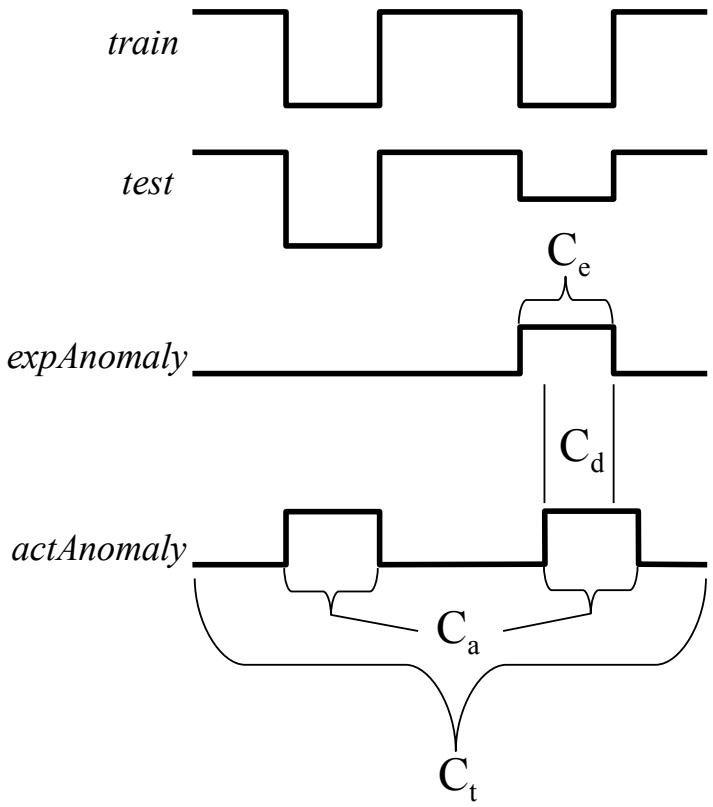

Figure 5.1: Illustration of the data point counts that are used to calculate the rate of detections and rate of false positives.

$$R_{Det} = \frac{C_d}{C_e}$$

and the rate of false positives is defined as

$$R_{FP} = \frac{C_a - C_d}{C_t - C_e}$$

These rates are also dependent on the `SetNormalDataFilter` used at the end of each technique. This filter only keeps data points that differ from the mean by more than a configurable number of standard deviations; any other data points are set to some configurable value. This means that increasing the configurable standard deviations would in effect increase both the R_{Det} and R_{FP} rates. Likewise, decreasing the configurable standard deviations would lower both the R_{Det} and R_{FP} rates. In all the following experiments we use a standard deviation of 1.

Additionally, we obtain values for R_{Det} and R_{FP} from all individual data sets and then average the results for discussion.

5.2.3 Experiment Descriptions

5.2.3.1 Novelty Detection using IBUS

In this experiment we test each of the novelty detection techniques on data reduced by the IBUS feature extraction technique. To make IBUS reduce data to the appropriate C_{ratio}, we use the same procedure described in Section 3.2.1. The difference here is that we omit experiments at $C_{ratio} = 4\%$ and include experiments for $C_{ratio} = 100\%$. The 4% compression ratio was not used because it did not retain many of the interesting features needed in the test data sets. We ran the tests with no compression

to determine if novelty detection was better with features as input rather than raw data.

5.2.3.2 Novelty Detection using IBUS and LP Filter

Many of the anomalies of interest are essentially low frequency changes in the signal. High frequency components of the signal may interfere with the detection of these anomalies. The purpose of this experiment is to determine whether removing the high frequency content of a signal improves the results by making the anomalies easier to detect. This is done as a preprocessing step to the IBUS technique. We use a simple moving average filter to perform the low pass (LP) filtering. A window size of 35 samples was determined by trial and error.

5.2.3.3 Varying the FilterTime Parameter of the Path Algorithm

In the first two experiments, the Path algorithm's parameters were set to produce the best results. One feature of the Path algorithm that was not used is its built in low pass filtering ability. This feature is enabled by setting the FilterTime parameter to something greater than one. The FilterTime parameter is mentioned in Section A.4.1. In this experiment we will test several different values of FilterTime in hopes that an optimal value will be discovered.

5.2.3.4 Varying the AlphabetSize Parameter of the Tarzan Algorithm

In order to calculate the required probabilities in the Tarzan technique, the data must first be discretized. In the previous experiments, the input signal was discretized into 4 distinct values (i.e., the AlphabetSize parameter was set to 4). In this experiment

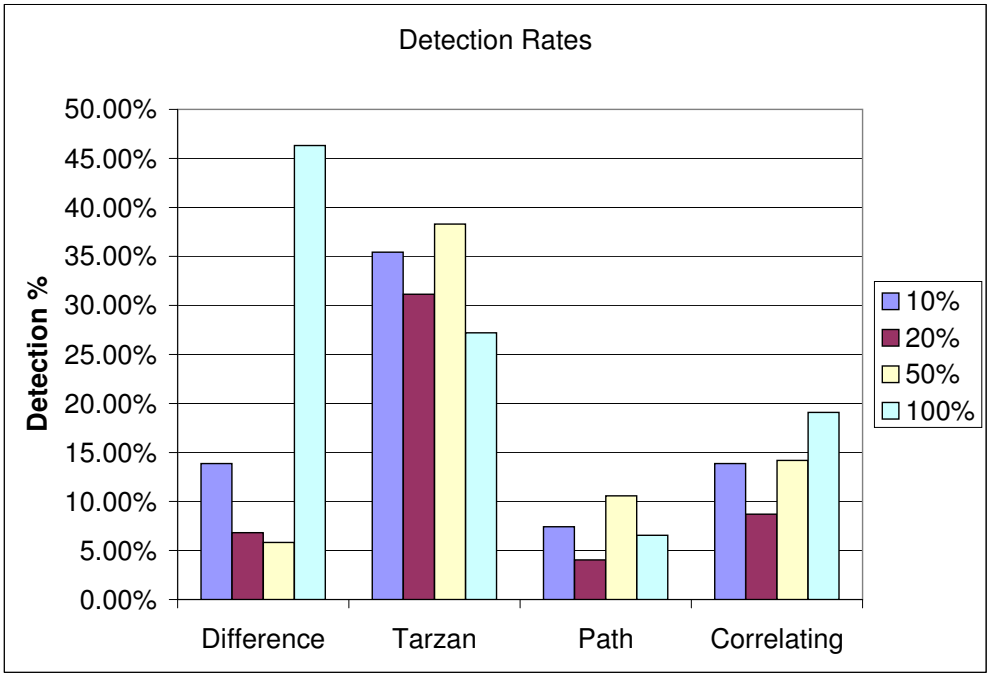

Figure 5.2: A comparison of detection rates for the novelty detection techniques using IBUS.

we test several other values of AlphabetSize to see if one is better.

5.3 Results

5.3.1 Novelty Detection using IBUS

The results obtained from the experimentation certainly did not identify a clear winner overall. As shown in Figure 5.2, the Tarzan technique produced the highest detec-

tion rates with the exception of when $C_{ratio} = 100\%$. At $C_{ratio} = 100\%$, the Difference technique had the highest detection rate out of all experiments. It had 8% more detections than the second best, which was the Tarzan technique at $C_{ratio} = 50\%$. At other compression levels, the Difference technique had very poor results. The explanation for Difference performing so much better at $C_{ratio} = 100\%$ is related to the number of points recorded during a novelty. The raw data (i.e., $C_{ratio} = 100\%$) is evenly spaced in time; every 20 seconds there is a data point. Since the Difference technique compares each data point to another, more data points in an area of novelty creates a greater chance for a detection. Conversely, a low C_{ratio} would potentially mean fewer data points that occur during a novelty and thus a lower chance of a detection hit.

The Path technique was the worst performing overall. Its detection rates ranged from 4% to 10%. These poor results, along with those of the Difference technique influenced the Correlating technique to have less than expected detection rates.

It may seem intuitive that all techniques would perform better when the data is less compressed (i.e., with more detail intact). This was not so for the Tarzan and Path techniques. Tarzan and Path performed best at $C_{ratio} = 50\%$. In addition, all techniques performed better at $C_{ratio} = 10\%$ than at $C_{ratio} = 20\%$.

As shown in Figure 5.3, the Tarzan technique also had the highest false positive rates. Tarzan's false positive rate nearly doubled at $C_{ratio} = 10\%$ from what it was at $C_{ratio} = 20\%$. Generally, for all techniques as C_{ratio} decreased, the rate of false positives increased.

It is also interesting to note that the Tarzan technique had the least amount of false positives at $C_{ratio} = 50\%$, the same compression ratio that it had its highest

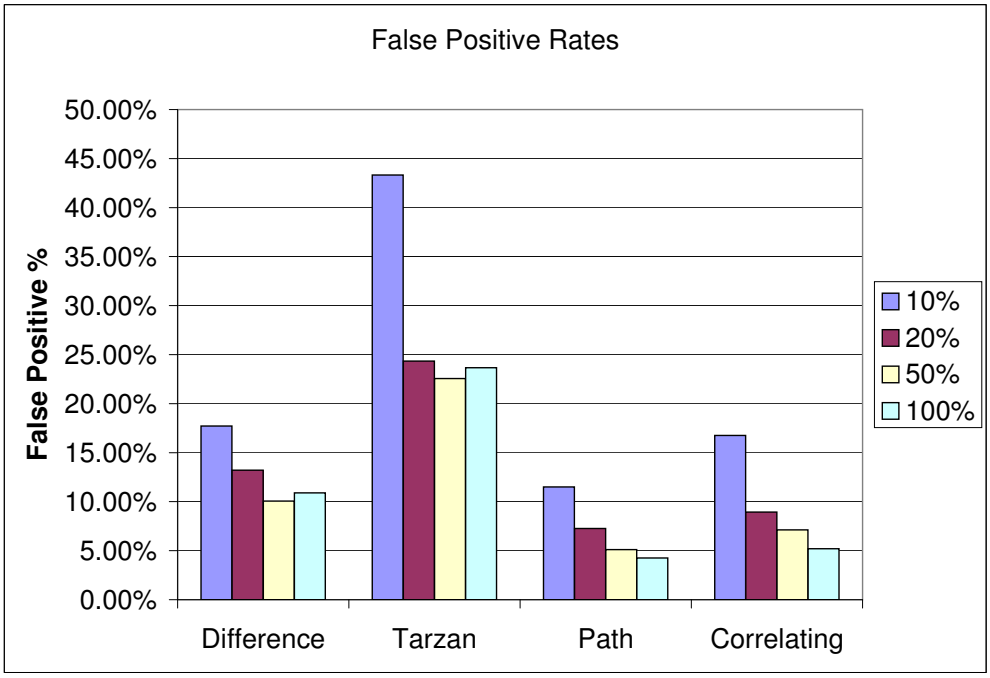

Figure 5.3: A comparison of false positive rates for the novelty detection techniques using IBUS.

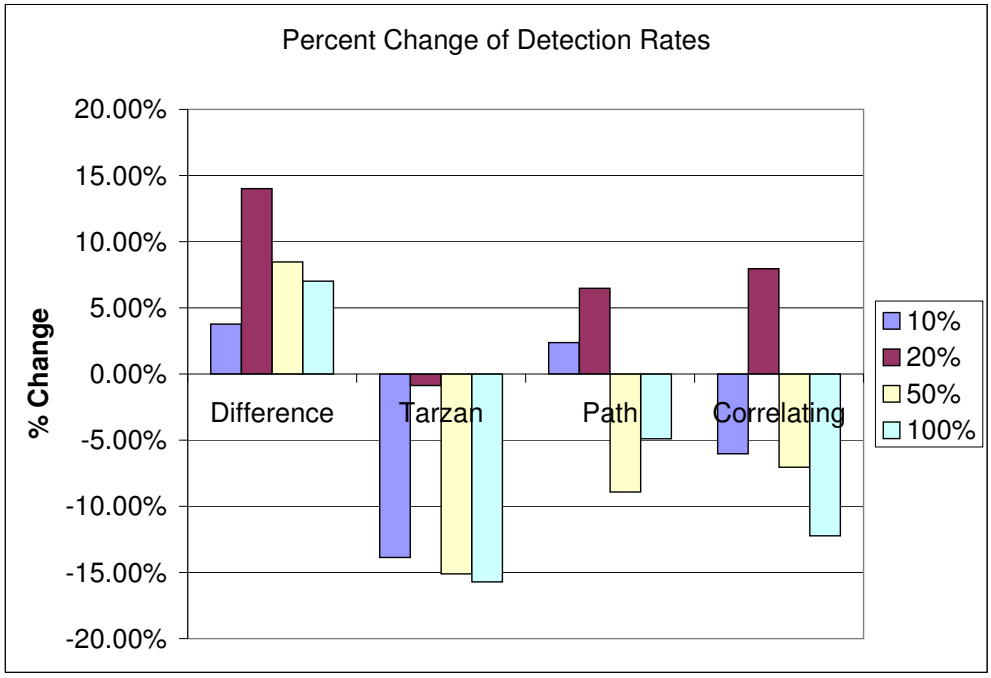

Figure 5.4: A comparison of the percent change of detection rates for the novelty detection techniques using a low pass filter in addition to IBUS.

number of detections.

5.3.2 Novelty Detection using IBUS and LP Filter

As shown in Figure 5.4, the technique that benefited the most from the low pass filter was the Difference technique. The low pass filter increased the Difference technique's detection rate between 3% and 14% for each compression ratio. The greatest benefit for all techniques was at $C_{ratio} = 20\%$.

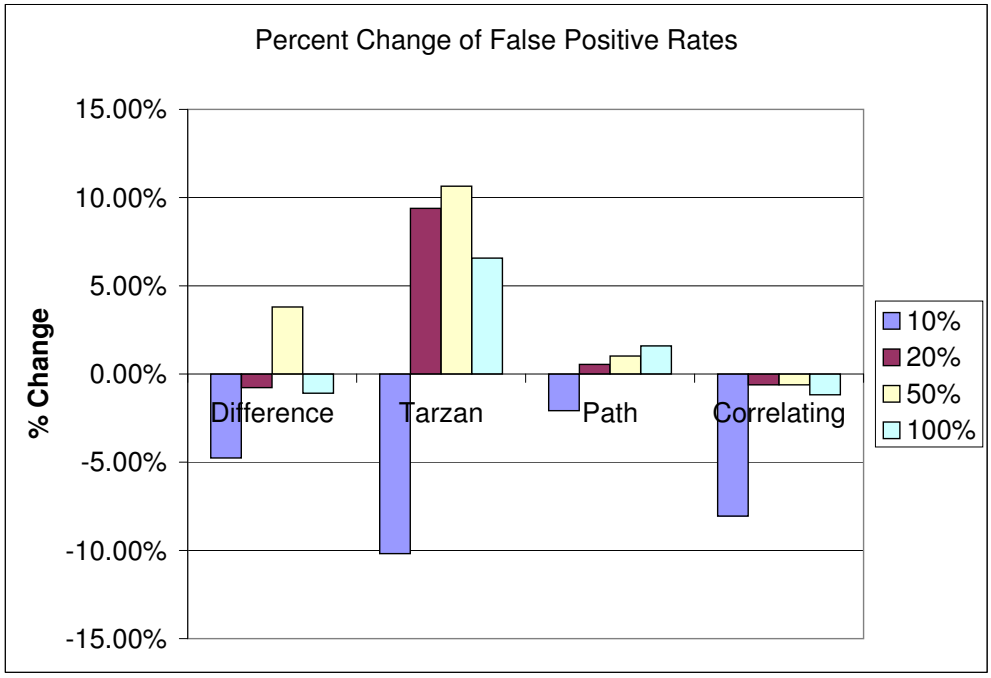

Figure 5.5: A comparison of the percent change of false positive rates for the novelty detection techniques using a low pass filter in addition to IBUS.

For the most part, the low pass filter had a negative impact on the other technique's detection rates.

As shown in Figure 5.5, the biggest improvement in false positive rates was at $C_{ratio} = 10\%$. At this compression ratio, all techniques had a substantial improvement. However, at all other compression ratios, the Tarzan technique had a large increase in the number of false positives when using a low pass filter.

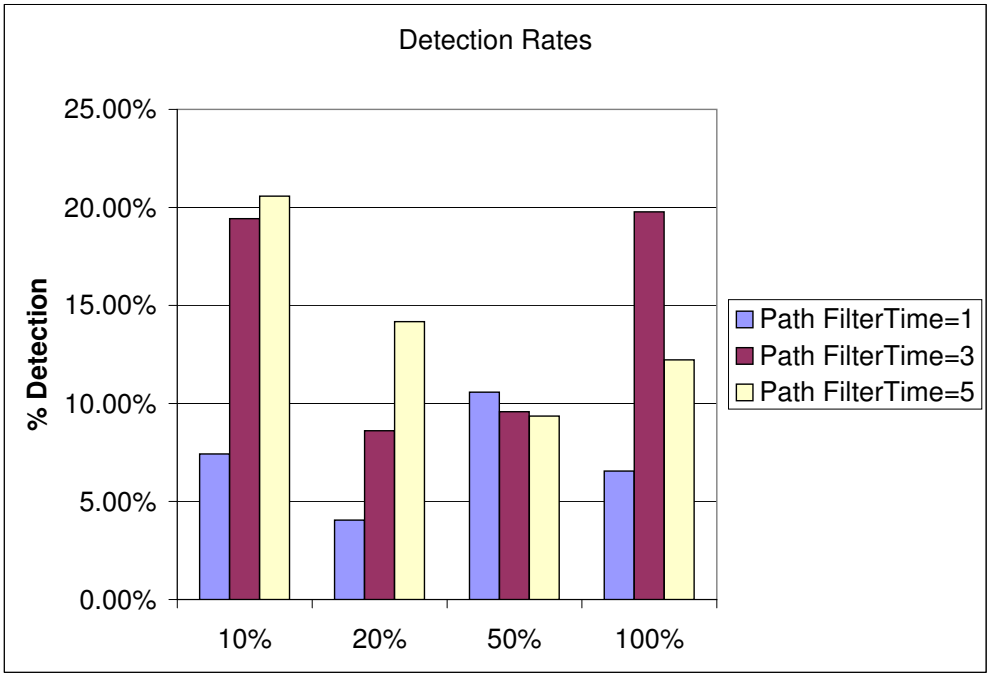

Figure 5.6: A comparison of the detection rates for the Path technique when the FilterTime parameter is varied.

5.3.3 Varying the FilterTime Parameter of the Path Algorithm

As shown in Figure 5.6, increasing the FilterTime parameter of the Path technique generally improved the detection rate. The only exceptions were for when $C_{ratio} = 50\%$ and $C_{ratio} = 100\%$, in which case the detection rate decreased. Figure 5.7 shows that increasing FilterTime also increased the false positive rate substantially.

Figure 5.7: A comparison of the false positive rates for the Path technique when the FilterTime parameter is varied.

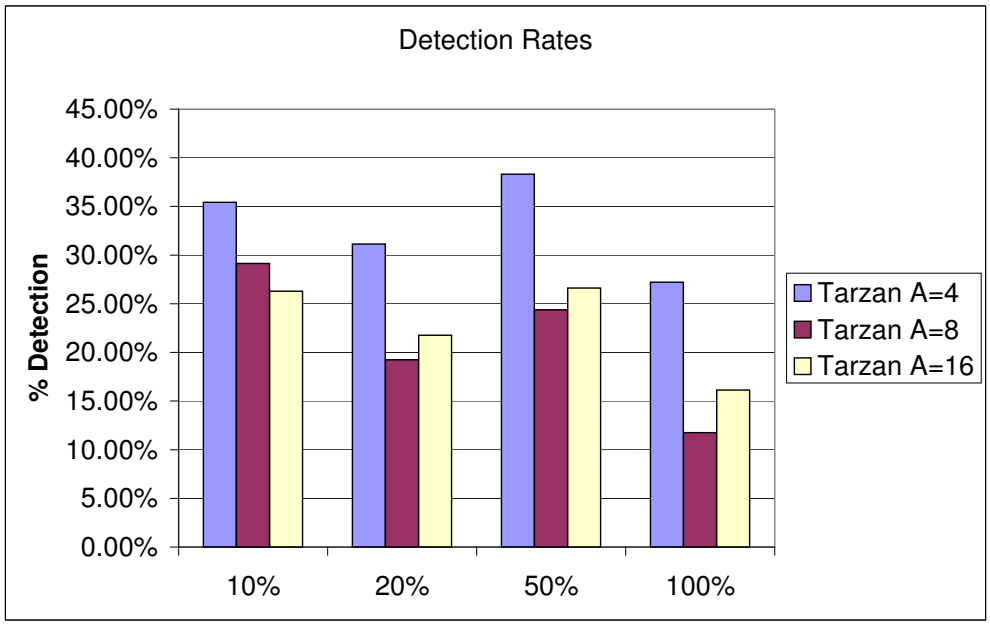

Figure 5.8: A comparison of the detection rates for the Tarzan technique when the AlphabetSize parameter is varied.

5.3.4 Varying the AlphabetSize Parameter of the Tarzan Algorithm

As shown in Figures 5.8 and 5.9, increasing the AlphabetSize parameter of the Tarzan approach had very negative effects on the results. In nearly all cases, the detection rate was decreased and the false positive rate was increased.

Figure 5.9: A comparison of the false positive rates for the Tarzan technique when the AlphabetSize parameter is varied.

5.4 Conclusion

The results show that the best technique is the Difference technique at $C_{ratio} = 100\%$. At this compression ratio $R_{Det} = 53.3\%$ and $R_{FP} = 9.8\%$. This means that the best results were obtained when no feature extraction was used. This is disappointing because no speedup can be obtained when the data is not reduced. When the data was compressed, the Difference technique performed relatively poorly. At $C_{ratio} = 50\%$ the Tarzan technique had the best results with $R_{Det} = 38.3\%$ and $R_{FP} = 22.5\%$.

The Path technique had R_{Det} values that were very low throughout the experiments. The R_{FP} values were also low, but without many detections the technique would be of little practical value in this application domain.

The mediocre results for the Correlating technique were surprising. They indicate that the other techniques are not all detecting the same portions of anomalous data. It makes sense that certain techniques are better at detecting certain data patterns than others.

In conclusion, if no compression is required, the Difference technique with a low pass filter is the best choice for applications involving fiber property data sets. If compression is required, the Tarzan technique with AlphabetSize=4 is best.

Chapter 6

Concluding Remarks

In our work we have explored several aspects of implementing time series novelty detection in a fiber manufacturing process. One aspect of realistic novelty detection is how to reduce the data so that important data features remain intact. The process of doing this is called feature extraction. We proposed a feature extraction technique that produces the highest quality reduced representation for fiber property data sets. Given that result, we implemented and evaluated several of the leading novelty detection techniques using our new feature extraction technique as a preprocessing step.

In Section 6.1 we detail the key contributions from this study. While we consider these results to be highly valuable, they are not considered final. Many other concerns exist when considering the full integration of novelty detection into a fiber monitoring system. As we discuss in Section 6.2, some of these may be considered for future work.

6.1 Key Contributions

- We evaluated several leading feature extraction techniques on fiber property data; this included the Discrete Fourier Transform (DFT), Piecewise Aggregate Approximation (PAA), Bottom-Up Segmentation (BUS), the Haar Wavelet Transform (HWT), and the Daubechies Wavelet Transform (DWT). The main result was that the BUS technique produced the highest quality features at all levels of compression.

- Upon inspection of the features produced by the BUS technique we discovered that many data points were redundant (i.e. could be removed with no effect on the quality). To take advantage of this fact, we introduced a new feature extraction technique based on the BUS approach, the Improved Bottom-Up Segmentation (IBUS) technique. After repeating the feature extraction tests, IBUS was shown to produce the highest quality features for fiber property data sets. The benefits of using IBUS were most pronounced at high levels of compression. For instance at $C_{ratio} = 4\%$ the reconstruction error for IBUS was half of what it was for BUS. Additional tests comparing the data reduction capabilities of IBUS showed that the IBUS data size ranged from 61-68% of the BUS data size - a considerable improvement.

- As a precursor to implementing the novelty detection techniques, we designed a framework for constructing, running, and visualizing time series algorithms.

- The main component of our study was to evaluate several leading time series novelty detection techniques on fiber property data sets. To our knowledge

no other study has done this. To compare the techniques we measured their rate of detection and rate of false positives for anomalies recorded by a domain expert. Results showed that a simple Euclidean distance based technique is the best overall when no data compression is used. When compression is used the Tarzan technique [30] performs best.

6.2 Limitations and Future Directions

- A fiber monitoring system, like INSTRUMAR's Attalus Fiber System, operates in a strictly online fashion. This fact was not considered in any of the current work. Considering that some batch algorithms may not be able to be converted to an online situation, this is an area of considerable future work.

- All datasets used in this study were hand-picked anomalies from a certain range of time at a plant. It would be interesting to set up a test whereby the novelty detection algorithms would process data as in comes in for a longer period of time. Results could then be compared with that of the human operators. The interest here is to see if the novelty detection techniques would pick up on anomalies that were missed by the human operators.

- In our testing of the novelty detection techniques, we focused on the anomaly resulting from one tow's data stream deviating from the rest. It would be valuable to record detection and false positive rates for all known types of fiber anomalies.

- Since this study was focused on the fiber industry, no time was allocated for

testing IBUS on other datasets. As it was mentioned previously, IBUS is general enough to be applied to any time series dataset. It would be interesting to see if IBUS would show similar improvements on more diverse datasets, like the one used in [33].

- In chapter 3 we mentioned that some segmentation algorithms utilize a combination of the three approaches: Sliding Window, Top-Down, and Bottom-Up. One such algorithm by Keogh et al. combines the Bottom-Up approach with the Sliding Window approach [33]. This algorithm claims to produce results similar to that of BUS while running in an online fashion. It would be interesting to determine whether IBUS can be adapted to use this algorithm instead of BUS.

- For a particular problem domain, it may be possible to improve results further by incorporating domain constraints. For example, there are large drops in the fiber data magnitude at regular intervals that correspond to changing out spools of fiber. An additional signal from the machine changing the spools of fiber could flag to the novelty detection algorithm that any drops during a particular interval are normal.

- It may be interesting to regenerate the results using a simpler probability calculation for R_{Det}. Instead of considering every data point, it could be defined as just number of flaws detected divided by the total number of flaws. A flaw would be a data pattern consisting of consecutive anomolous data points. This would perhaps be more in line with the sliding window approach used in the algorithms.

- In all the novelty detection experiments we obtained values for R_{Det} and R_{FP} from all individual data sets and then averaged the results for discussion. It may be interesting to closely examine the results from each data set individually to see if the techniques performed particularly well or poor depending on the data set.

Appendix A

Filter Definitions

This appendix covers the filters used during the experimentation in this text; both novelty detection and feature extraction filters are covered.

A.1 CLoDS package

A.1.1 ADITU FileDataSource

This filter reads in files generated by INSTRUMAR's Attalus Data Internet Transfer Utility (ADITU). The data in these files is delimited by spaces. The first row contains column names, and the remaining rows are the column values. An example of an ADITU file, with header row in place is as follows:

Listing A.1: Sample Attalus Data Internet Transfer Utility (ADITU) file

```
MachineName PositionName ThreadlineName ThreadlineIndex Time
   PctValidData BIT_Status Magnitude NodeCount NodeQuality Phase
   ThreadPresence WinderStatus
"Machine1" "Position1" "Thread1" "26" "01/10/2004␣05:00:02" "100" "0
   " "1989" "25" "415" "101" "1" "2"
...
```

There are several possible configuration options available:

- **FileName** - The name of the ADITU file to read.
- **TimeName** - The column name that is the time value of the time series.
- **ValueName** - The column name that is the data value of the time series.
- **Interval** - Interval in milliseconds between polls.
- **RecordsPerInterval** - Number of records to read in each poll. Default is all records available.
- **RecordStart** - Specify a starting record number. For an ADITU file, this would be a line number from which to start reading. All records before this will be ignored.
- **RecordLimit** - Specify an ending record number. Any records after this number will be ignored.

A typical configured filter looks like:

Listing A.2: Typical ADITU FileDataSource filter configuration

```
<filter assembly="CLoDS" type="File.ADITU_FileDataSource">
  <config>
    <item name="FileName">path/to/aditu.txt</item>
    <item name="TimeName">Time</item>
    <item name="ValueName">Magnitude</item>
    <item name="Interval">1</item>
  </config>
</filter>
```

A.2 LiFE package

A.2.1 BUSFilter

This filter calculates a piecewise linear representation of the incoming data using a bottom up approach as described in Section 3.1.4.

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before performing the segmentation. The default is to process each time series regardless of size (i.e. no batching).
- **MaxError** - The maximum residual error allowed between the original time series and the piecewise linear approximation.

A typical configured filter looks like:

Listing A.3: Typical BUSFilter filter configuration

```
<filter assembly="LiFE" type="BUSFilter">
    <config>
        <item name="MaxError">0.5</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.2 BestFitLineSlopeFilter

This filter first calculates a piecewise linear representation of the incoming data using a bottom up approach as described in Section 3.1.4. It then calculates the slope at each data point and outputs that as the values in the new time series. This approach was described by Keogh, et al in [33].

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before performing the segmentation. The default is to process each time series regardless of size (i.e. no batching).

- **MaxError** - The maximum residual error allowed between the original time series and the piecewise linear approximation.

A typical configured filter looks like:

Listing A.4: Typical BestFitLineSlopeFilter filter configuration

```
<filter assembly="LiFE" type="BestFitLineSlopeFilter">
    <config>
        <item name="MaxError">0.5</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.3 IBUSFilter

This filter calculates a piecewise linear representation of the incoming data using the improved bottom up approach as described in Section 3.3.

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before performing the segmentation. The default is to process each time series regardless of size (i.e. no batching).
- **MaxError** - The maximum residual error allowed between the original time series and the piecewise linear approximation.

A typical configured filter looks like:

Listing A.5: Typical IBUSFilter filter configuration

```
<filter assembly="LiFE" type="IBUSFilter">
    <config>
        <item name="MaxError">0.5</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.4 SAXFilter

This filter uses the Symbolic Aggregate approXimation (SAX) [37] approach to discretize the incoming time series. The output of this filter is not a time series of real values but rather a time series of letters from a finite alphabet. You can think of the alphabet size as defining the resolution of the data. At the lowest resolution, with an alphabet size of 1, the resultant time series is a straight line.

This filter has the requirement that incoming data be normalized first using the NormalizeFilter.

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before performing the SAX process. The default is to process each time series regardless of size (i.e. no batching).
- **AlphabetSize** - The number of letters used for values in the resultant time

series.

A typical configured filter looks like:

Listing A.6: Typical SAXFilter filter configuration

```
<filter assembly="LiFE" type="SAXFilter">
    <config>
        <item name="AlphabetSize">4</item>
    </config>
    <filter assembly="TARZAN" type="NormalizeFilter">
        <config />
        <filter ...>
            <config>
              ...
            </config>
        </filter>
    </filter>
</filter>
```

A.2.5 MovingAverageFilter

This filter performs low pass filtering on the incoming data using a configurable sized moving average.

There is only one configuration option available:

- **WindowSize** - the size in data records of the sliding window. This must be odd and greater than 1.

A typical configured filter looks like:

Listing A.7: Typical MovingAverageFilter filter configuration

```
<filter assembly="LiFE" type="MovingAverageFilter">
    <config>
        <item name="WindowSize">35</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.6 SetNormalDataFilter

This filter only keeps data points that differ from the mean by more than a configurable number of standard deviations; any other data points are set to some configurable value.

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before processing. The default is to process each time series regardless of size (i.e. no batching).
- **SetValue** - The value to set non interesting data points to.
- **AllowedDeviations** - The number of standard deviations away from the mean that a data point must be to be considered an outlier.

A typical configured filter looks like:

Listing A.8: Typical SetNormalDataFilter filter configuration

```
<filter assembly="LiFE" type="SetNormalDataFilter">
    <config>
        <item name="AllowedDeviations">1</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.7 DWTFilter

This filter processes the incoming time series with a Discrete Wavelet Transform (DWT) that uses a Daubechies-4 wavelet function.

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before processing. The default is to process each time series regardless of size (i.e. no batching). This must be a power of two for the DWT.

- **KeepPercentage** - Percentage of the resultant wavelet coefficients to keep. Typically, a signal can be represented accurately by only a subset of the possible coefficients.

- **KeepHighestCoefficients** - This flag specifies whether to use only the highest coefficients. As described in Section 3.1.1.1 sometimes a signal is best repre-

sented by keeping the highest valued coefficients rather than the first few. For fiber property data sets though, it was found that the first few coefficients produced a better fit.

A typical configured filter looks like:

Listing A.9: Typical DWTFilter filter configuration

```
<filter assembly="LiFE" type="DWTFilter">
    <config>
        <item name="KeepPercentage">0.1</item>
        <item name="KeepHighestCoefficients">false</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.8 HaarFilter

This filter processes the incoming time series with a Discrete Wavelet Transform (DWT) that uses a Haar wavelet function.

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before processing. The default is to process each time series regardless of size (i.e. no batching). This must be a power of two for the DWT.
- **KeepPercentage** - Percentage of the resultant wavelet coefficients to keep.

Typically, a signal can be represented accurately by only a subset of the possible coefficients.

- **KeepHighestCoefficients** - This flag specifies whether to use only the highest coefficients. As described in Section 3.1.1.1 sometimes a signal is best represented by keeping the highest valued coefficients rather than the first few. For fiber property data sets though, it was found that the first few coefficients produced a better fit.

A typical configured filter looks like:

Listing A.10: Typical HWTFilter filter configuration

```
<filter assembly="LiFE" type="HWTFilter">
    <config>
        <item name="KeepPercentage">0.1</item>
        <item name="KeepHighestCoefficients">false</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.9 FFTFilter

This filter processes the incoming time series with a Discrete Fourier Transform (DFT).

There are several possible configuration options available:

- **BatchSize** - Specifies the number of records in the time series to batch before processing. The default is to process each time series regardless of size (i.e. no batching). This must be a power of two for the DFT.

- **KeepPercentage** - Percentage of the resultant wavelet coefficients to keep. Typically, a signal can be represented accurately by only a subset of the possible coefficients.

- **KeepHighestCoefficients** - This flag specifies whether to use only the highest coefficients. As described in Section 3.1.1.1 sometimes a signal is best represented by keeping the highest valued coefficients rather than the first few. For fiber property data sets though, it was found that the first few coefficients produced a better fit.

A typical configured filter looks like:

Listing A.11: Typical FFTFilter filter configuration

```
<filter assembly="LiFE" type="FFTFilter">
    <config>
        <item name="KeepPercentage">0.1</item>
        <item name="KeepHighestCoefficients">false</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.2.10 PAAFilter

This filter uses the Piecewise Aggregate Approximation (PAA) approach to discretize the incoming time series. PAA is described in more detail in Section 3.1.3.

There is one possible configuration option available:

- **FrameSize** - Specifies the size of the frame to average.

A typical configured filter looks like:

Listing A.12: Typical PAAFilter filter configuration

```
<filter assembly="LiFE" type="PAAFilter">
    <config>
        <item name="FrameSize">5</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.3 PAFF package

A.3.1 AveragingFilter

Accepts multiple time series and outputs a single time series which is the average of the inputs. For example, you could have 3 input time series:

```
TimeSeries1 [0][1][2]...
```

```
TimeSeries2 [3][4][5]...
TimeSeries3 [6][7][8]...
```

The resultant time series would be:

```
TimeSeriesResult [0+3+6 / 3][1+4+7 / 3][3+5+8 / 3]...
```

The time series entries are combined based on their array index by default. To get them to align by time values, enable the AlignDataStreams option. There are several possible configuration options available:

- **CombiningInstances** - Number of records in the multiple data streams to combine. The default is to use the size of the shortest data stream.
- **AlignDataStreams** - When enabled, the time series entries will be aligned by time value rather than array index. This will likely mean that additional points, calculated by interpolation are added to the time series. For an illustration see Figure 3.4.

A typical configured filter looks like:

Listing A.13: Typical AveragingFilter filter configuration

```
<filter assembly="PAFF" type="AveragingFilter">
    <config />
    <filter ...>
        <config>
          ...
        </config>
    </filter>
    ...
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.3.2 CorrelatingFilter

This filter also combines multiple time series like the AveragingFilter. It is the basis of the hybrid algorithm described in Section 4.2.4. It will output a value of 1 if a specified number of input streams are above a threshold, otherwise the output is 0. This is a simple voting algorithm.

There are several possible configuration options available:

- **CombiningInstances** - Number of records in the multiple data streams to combine. The default is to use the size of the shortest data stream.
- **AlignDataStreams** - When enabled, the time series entries will be aligned by

time value rather than array index. This will likely mean that additional points, calculated by interpolation are added to the time series. For an illustration see Figure 3.4.

- **RequiredVotes** - The number of data streams that need to be over the threshold for the filter to output 1.
- **Threshold** - Specifies the value a data stream must meet before it constitutes a vote.

A typical configured filter looks like:

Listing A.14: Typical CorrelatingFilter filter configuration

```
<filter assembly="PAFF" type="CorrelatingFilter">
    <config>
        <item name="RequiredVotes">2</item>
        <item name="Threshold">0</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
    ...
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.3.3 SquaredDifferenceFilter

This filter also combines multiple time series like the AveragingFilter. Instead of averaging the data streams though, it takes the difference of them and squares that value for the result. This filter only supports 2 input data steams. This filter also forms part of the time aligned euclidean difference algorithm as described in Section 4.2.1.

There are several possible configuration options available:

- **CombiningInstances** - Number of records in the multiple data streams to combine. The default is to use the size of the shortest data stream.

- **AlignDataStreams** - When enabled, the time series entries will be aligned by time value rather than array index. This will likely mean that additional points, calculated by interpolation are added to the time series. For an illustration see Figure 3.4.

A typical configured filter looks like:

Listing A.15: Typical SquaredDifferenceFilter filter configuration

```
<filter assembly="PAFF" type="SquaredDifferenceFilter">
    <config />
    <filter ...>
        <config>
          ...
        </config>
    </filter>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.4 PATH package

A.4.1 MultivariatePathBasedAnomalyFilter

This filter comprises the novelty detection algorithm created by Mahoney et al. [10] as described in Section 4.2.3. This filter is based off of the LearningFilter which takes in two types of data: training data and test data. Any incoming test data is queued until the filter has processed enough training data.

There are several possible configuration options available:

- **FilterTime** - This specifies the number of data samples to use for the window used in the low pass filter. A value of 1 essentially disables this feature.
- **ModelSegments** - This specifies the number of segments to reduce the original data stream to using the custom segmentation algorithm presented in [10]. A value of -1 disables this extra segmentation.
- **NumDimensions** - The number of dimensions to use when calculating the distance. Default is 3.
- **SubSampleRate** - Defines the sub sample rate which effectively averages the incoming data. Default is 1 (i.e. no averaging).

A typical configured filter looks like:

Listing A.16: Typical MultivariatePathBasedAnomalyFilter filter configuration

```
<filter assembly="PATH" type="MultivariatePathBasedAnomalyFilter">
    <config>
       <item name="FilterTime">1</item>
       <item name="ModelSegments">-1</item>
       <item name="NumDimensions">4</item>
       <item name="SubSampleRate">1</item>
    </config>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
    <filter ...>
        <config>
          ...
        </config>
    </filter>
    ...
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.5 TARZAN package

A.5.1 AbsFilter

This filter simply outputs the absolute value of the input time series. There are no configuration options. A typical configured filter looks like:

Listing A.17: Typical AbsFilter filter configuration

```
<filter assembly="TARZAN" type="AbsFilter">
    <config />
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.5.2 NormalizeFilter

This filter outputs a normalized version of the incoming time series where the mean is 0 and standard deviation is 1.

There is one configuration option available:

- **BatchSize** - Specifies the number of records in the time series to batch before processing. The default is to process each time series regardless of size (i.e. no batching).

A typical configured filter looks like:

Listing A.18: Typical NormalizeFilter filter configuration

```
<filter assembly="TARZAN" type="NormalizeFilter">
    <config />
    <filter ...>
        <config>
          ...
        </config>
    </filter>
</filter>
```

A.5.3 TarzanFilter

This filter comprises the novelty detection algorithm created by Keogh et al. [30] as described in Section 4.2.2. This filter is based off of the LearningFilter which takes in two types of data: training data and test data. Any incoming test data is queued until the filter has processed enough training data.

This filter requires that the input stream is normalized and that it contains character values as output by the SAXFilter.

There are several possible configuration options available:

- **WindowLength** - The size of the sliding window to use when testing data patterns for novelty.

A typical configured filter looks like:

Listing A.19: Typical TarzanFilter filter configuration

```
<filter assembly="TARZAN" type="TarzanFilter">
    <config>
```

```
   <item name="WindowLength">4</item>
</config>
<filter assembly="LiFE" type="SAXFilter">
    <config>
        <item name="AlphabetSize">4</item>
    </config>
    <filter assembly="TARZAN" type="NormalizeFilter">
        <config />
        <filter ...>
            <config>
              ...
            </config>
        </filter>
    </filter>
</filter>
<filter assembly="LiFE" type="SAXFilter">
    <config>
        <item name="AlphabetSize">4</item>
    </config>
    <filter assembly="TARZAN" type="NormalizeFilter">
        <config />
        <filter ...>
            <config>
              ...
            </config>
        </filter>
    </filter>
</filter>
```

```
</filter>
```

Appendix B

File Formats

B.1 Filter Template File (FTF)

In the testing framework described in Section 4.1, filters that perform specific tasks are assembled together to form more complex processes. The Filter Template File (FTF) format is used to describe how these filters are configured and how their inputs and outputs are connected.

XML is the obvious choice for the FTF format. XML provides human readability, parsing support in many languages, built in composability, and structure defining languages like DTD and XML Schema. Listing B.1 shows the XML Schema representation of an FTF.

Listing B.1: The Filter Template File (FTF) XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="filter">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="config" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="filter" minOccurs="0" maxOccurs="unbounded"
           />
      </xs:sequence>
      <xs:attribute name="assembly" use="required" type="xs:string"/
         >
      <xs:attribute name="id" use="required" type="xs:string"/>
      <xs:attribute name="type" use="required" type="xs:string"/>
      <xs:attribute name="vis" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="config">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="item"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="item">
    <xs:complexType mixed="true">
      <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing B.2: In this example of composing filters, messages would flow from filter D to filter A.

```
<?xml version="1.0" encoding="utf-8" ?>
<filter id="A" assembly="..." type="...">
  <config />
  <filter id="B" assembly="..." type="...">
    <config />
    <filter id="C" assembly="..." type="...">
      <config />
      <filter id="D" assembly="..." type="...">
        <config />
      </filter>
    </filter>
  </filter>
</filter>
```

B.1.1 Composing Filters

Composing filters together automatically links the inputs and outputs. In Listing B.2, filter A receives input from the output of filter B, filter B receives input from the output of filter C, and so on. Filter D has no composed filter so it is inferred to be a source of data.

B.1.2 Configuring Filters

To configure a filter, a "config" element is used. Configuration items require a name attribute and a value. The value can be any string, however it is up to the filter used

Listing B.3: Filters can be configured with the "config" element.

```
<?xml version="1.0" encoding="utf-8" ?>
<filter id="A" assembly="..." type="...">
  <config>
    <item name="BatchSize">800</item>
    <item name="FeatureWindowSize">=Math.Floor((Math.Log(0.5 * 256)
        / Math.Log(4)) - 1)</item>
  </config>
</filter>
```

to convert this string into a meaningful type. You can also specify a C# expression by prefixing the value with an equals character. See Listing B.3 for an example of this.

B.1.3 Identifying and Visualizing Filters

In order for the testing framework to load a particular filter, a .NET assembly and class name are required - these are specified as attributes. See Appendix A for possible combinations.

Visualizing the output of a particular filter is as simple as specifying the "vis" attribute. The visualization can either be a chart (vis=Plot) or a table of values (vis=Grid).

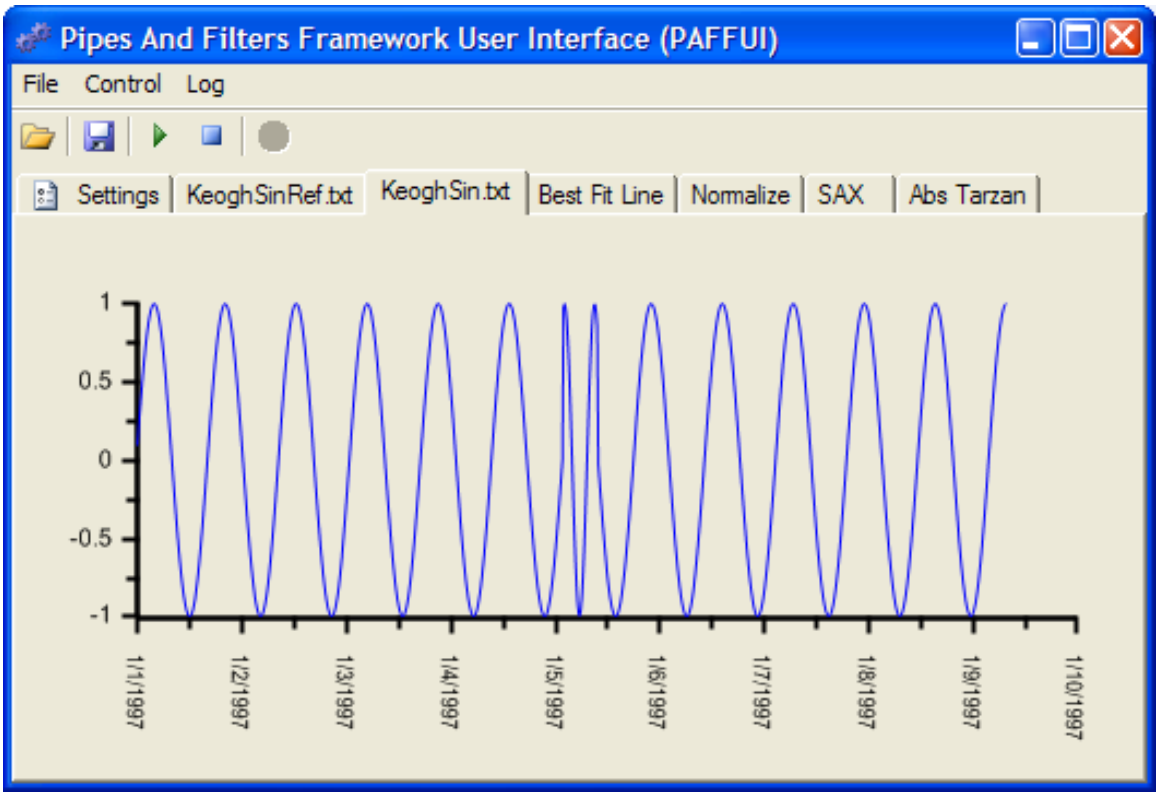

Figure B.1: By specifying a chart visualization type, the PAFFUI will load a tab containing a chart of the data at that point in the filter chain.

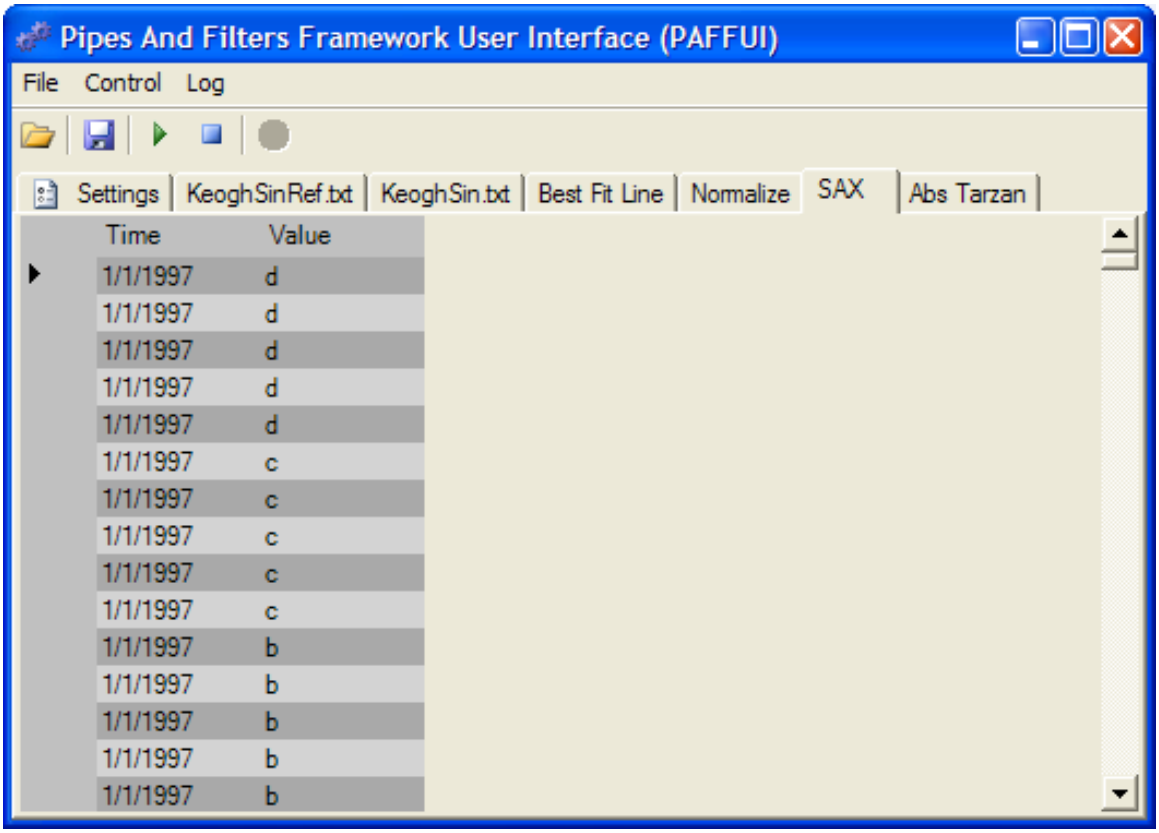

Figure B.2: By specifying a grid visualization type, the PAFFUI will load a tab containing a table of the data at that point in the filter chain. This is useful for displaying filters that have a non-numeric output data type.

B.2 Test Runner File (TRF)

The test runner file (TRF) is used to specify how to reconfigure an FTF for different data, algorithm parameters, and expected anomaly events. As shown in its schema definition in Listing B.4, its very simple. It is mostly a way to specify replacement values for template variables defined in an FTF for each executed test.

Listing B.4: The Test Runner File (TRF) XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
  <xs:element name="configuration">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="templates"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="templates">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="template"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="template">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element maxOccurs="unbounded" ref="entry"/>
      </xs:sequence>
      <xs:attribute name="file" use="required" type="xs:string"/>
      <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="entry">
    <xs:complexType mixed="true">
      <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Appendix C

Datasets

This appendix covers the datasets that were used during the experimentation in this text. As explained in Section 5.2.1, there are thirteen data tests in all. For each data test, there is a test data set known to have an anomaly which is fed into a trained anomaly detection technique. Techniques are trained on two training data sets which represent the normal behaviour.

C.1 Event 1 Datasets

The "Event 1" datasets all use the NodeQuality fiber data property. The 2 normal datasets used for training are shown in Figure C.1. The test dataset with a single highlighted anomalous region is shown in Figure C.2.

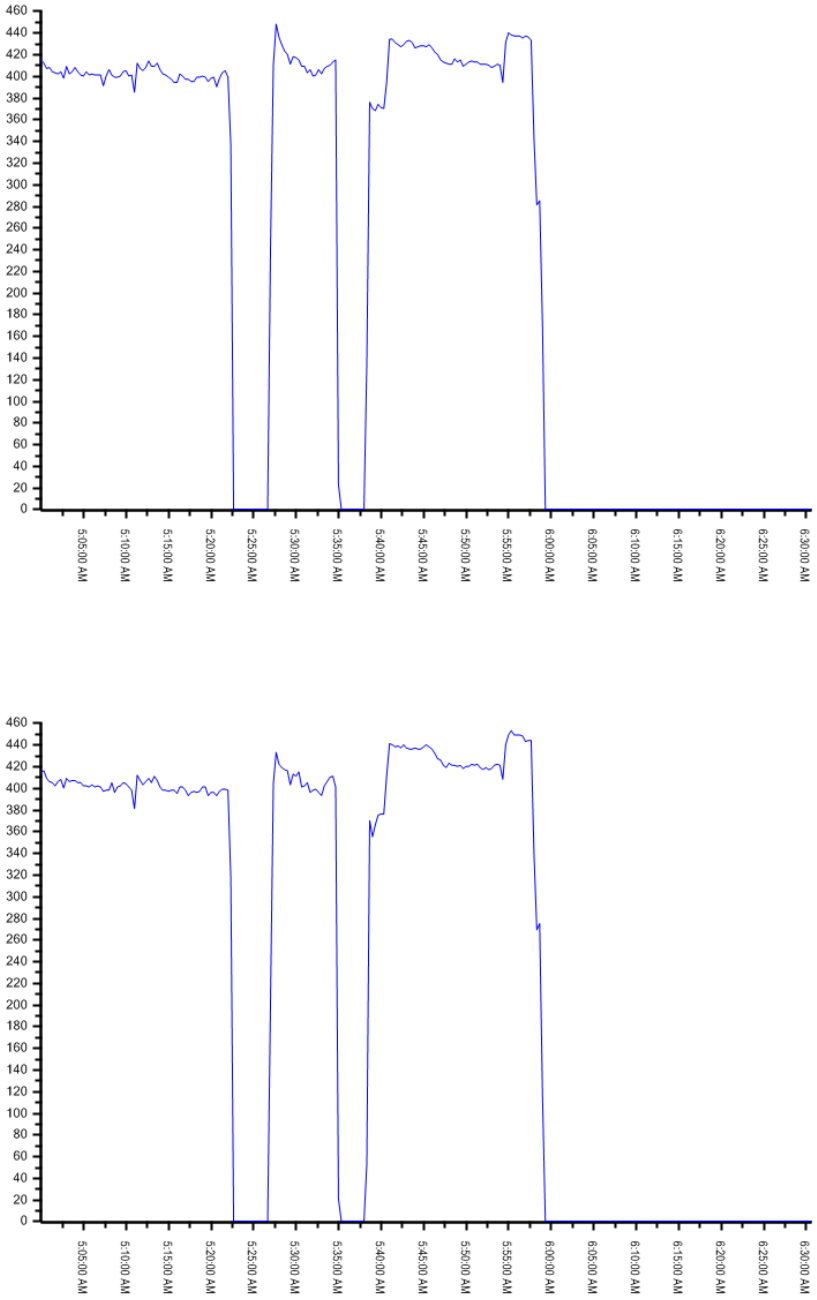

Figure C.1: Normal datasets for event 1.

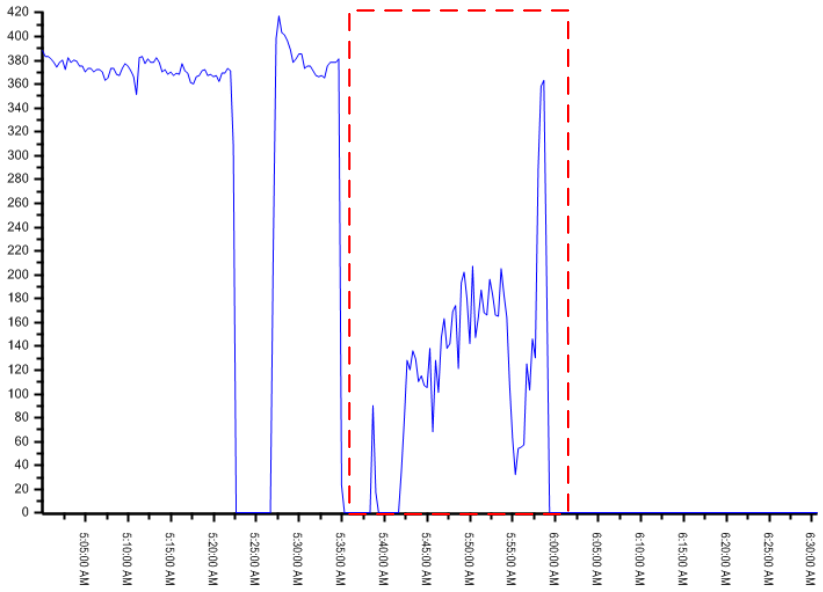

Figure C.2: Test dataset containing event 1 anomaly.

C.2 Event 2 Datasets

The "Event 2" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.3. The test dataset with a single highlighted anomalous region is shown in Figure C.4.

C.3 Event 4a Datasets

The "Event 4a" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.5. The test dataset with a single highlighted anomalous region is shown in Figure C.6.

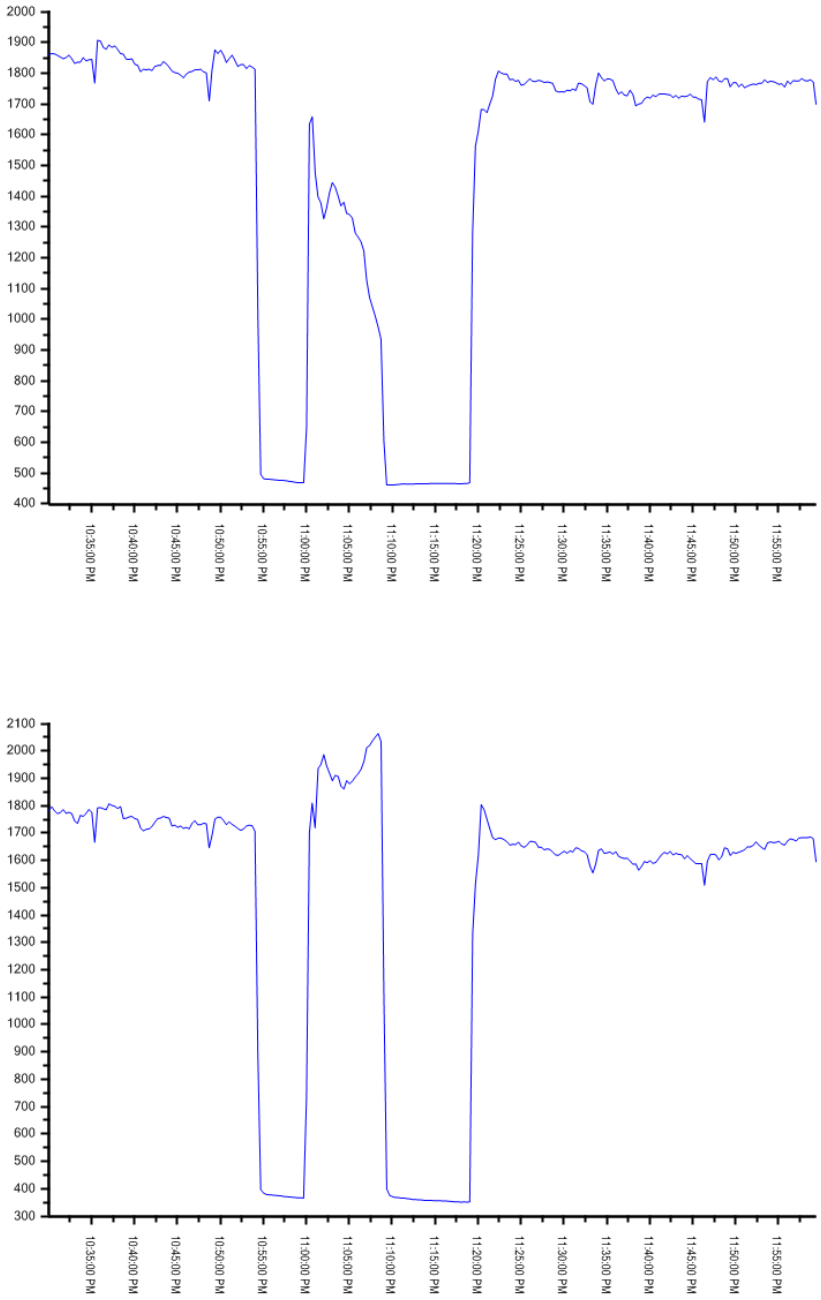

Figure C.3: Normal datasets for event 2.

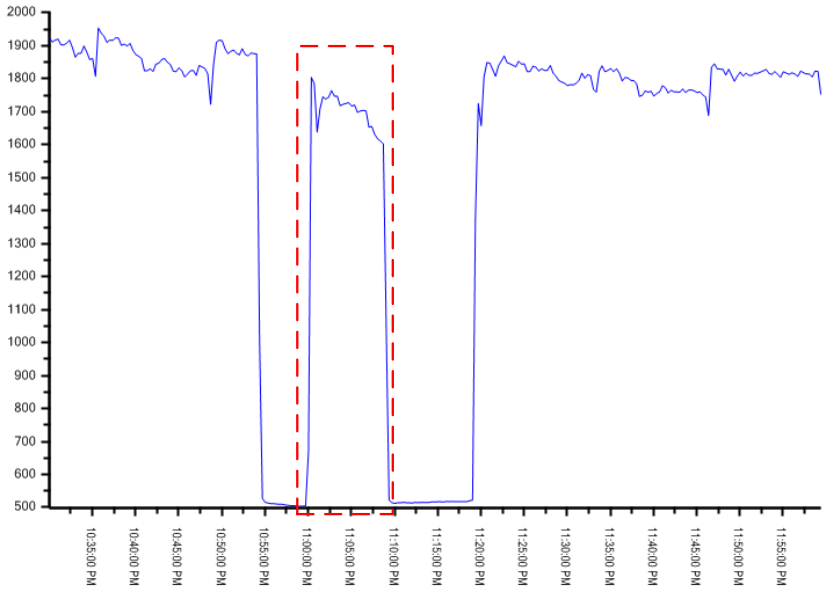

Figure C.4: Test dataset containing event 2 anomaly.

C.4 Event 4b Datasets

The "Event 4b" datasets all use the NodeQuality fiber data property. The 2 normal datasets used for training are shown in Figure C.7. The test dataset with a single highlighted anomalous region is shown in Figure C.8.

C.5 Event 5a Datasets

The "Event 5a" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.9. The test dataset with a single highlighted anomalous region is shown in Figure C.10.

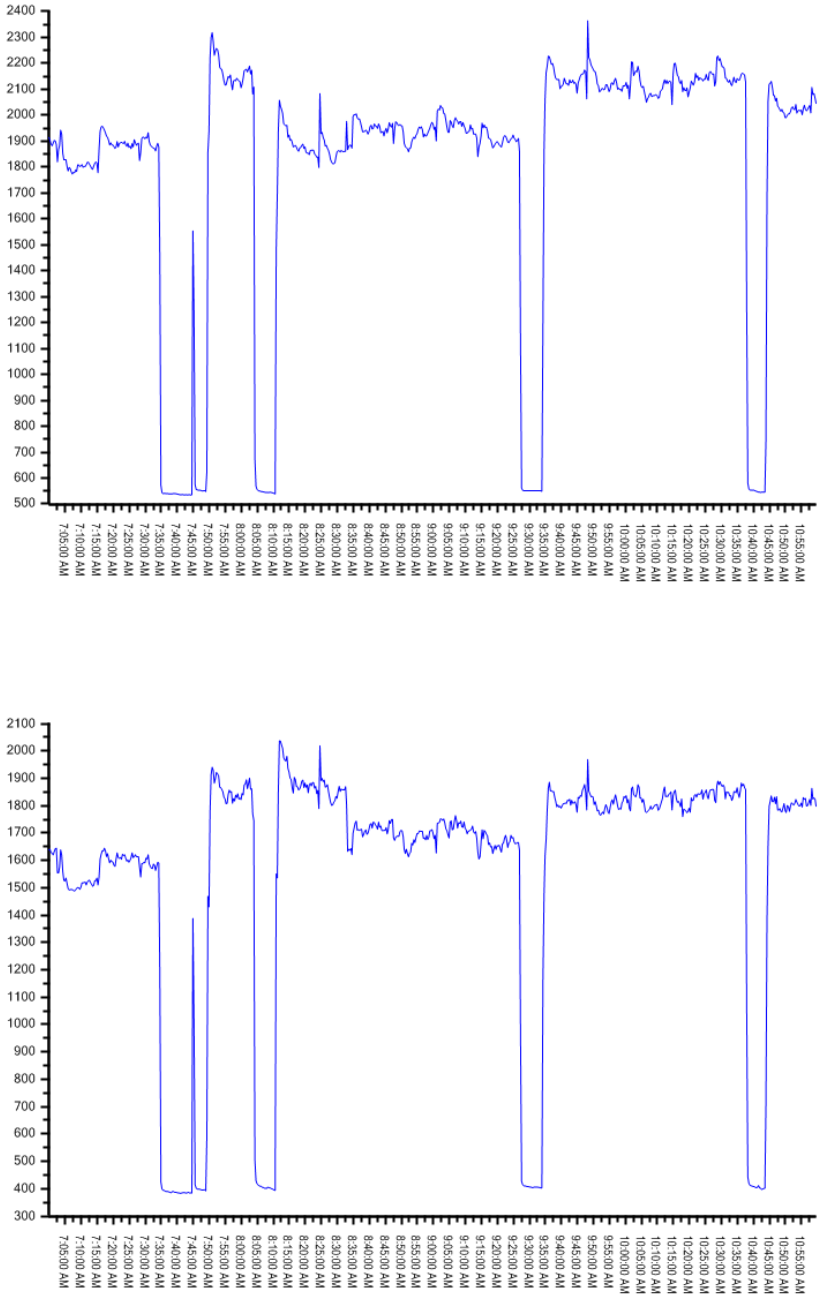

Figure C.5: Normal datasets for event 4a.

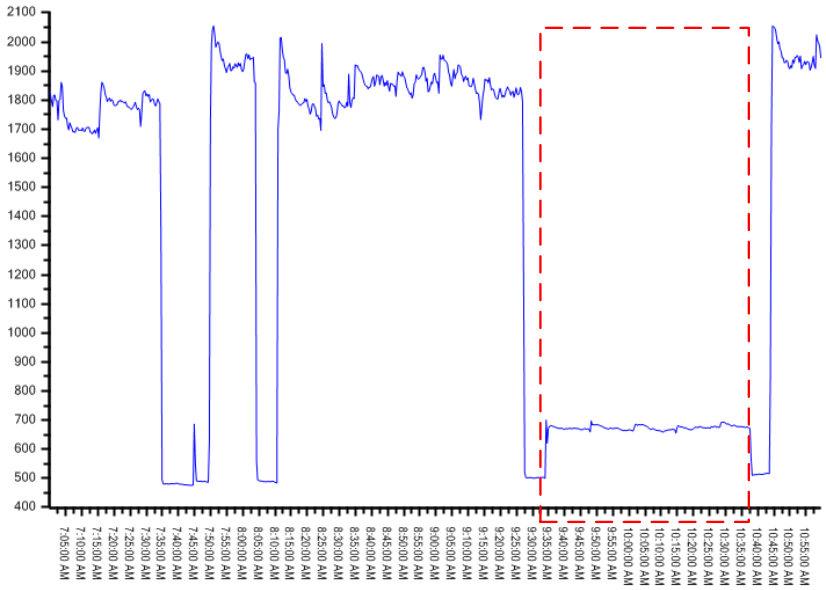

Figure C.6: Test dataset containing event 4a anomaly.

C.6 Event 5b Datasets

The "Event 5b" datasets all use the NodeQuality fiber data property. The 2 normal datasets used for training are shown in Figure C.11. The test dataset with a single highlighted anomalous region is shown in Figure C.12.

C.7 Event 6 Datasets

The "Event 6" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.13. The test dataset with a single highlighted anomalous region is shown in Figure C.14.

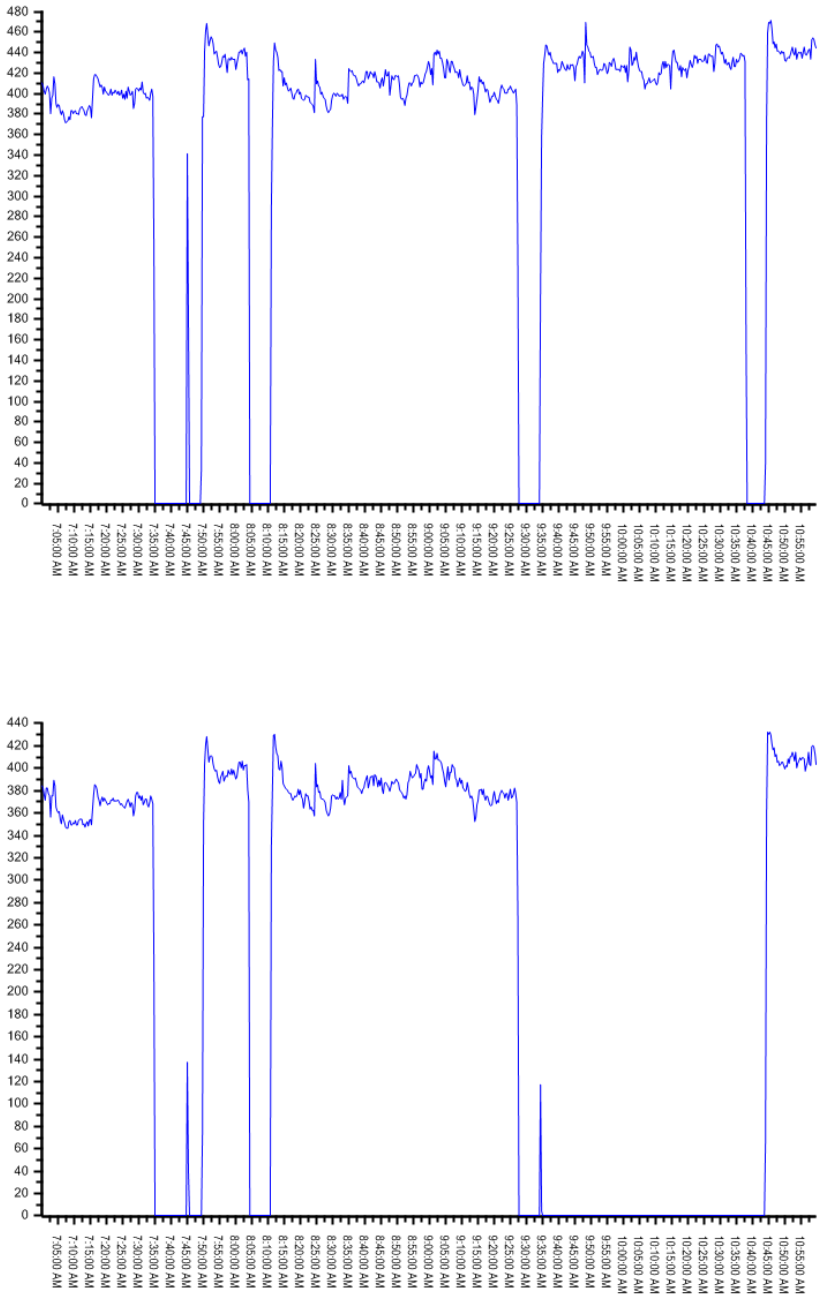

Figure C.7: Normal datasets for event 4b.

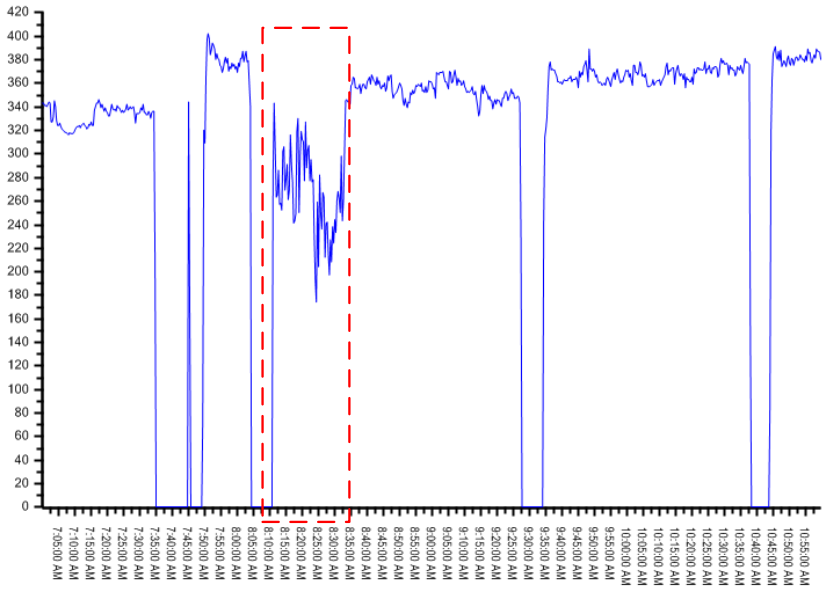

Figure C.8: Test dataset containing event 4b anomaly.

C.8 Event 7 Datasets

The "Event 7" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.15. The test dataset with 2 highlighted anomalous regions is shown in Figure C.16.

C.9 Event 8 Datasets

The "Event 8" datasets all use the NodeCount fiber data property. The 2 normal datasets used for training are shown in Figure C.17. The test dataset with a single highlighted anomalous region is shown in Figure C.18.

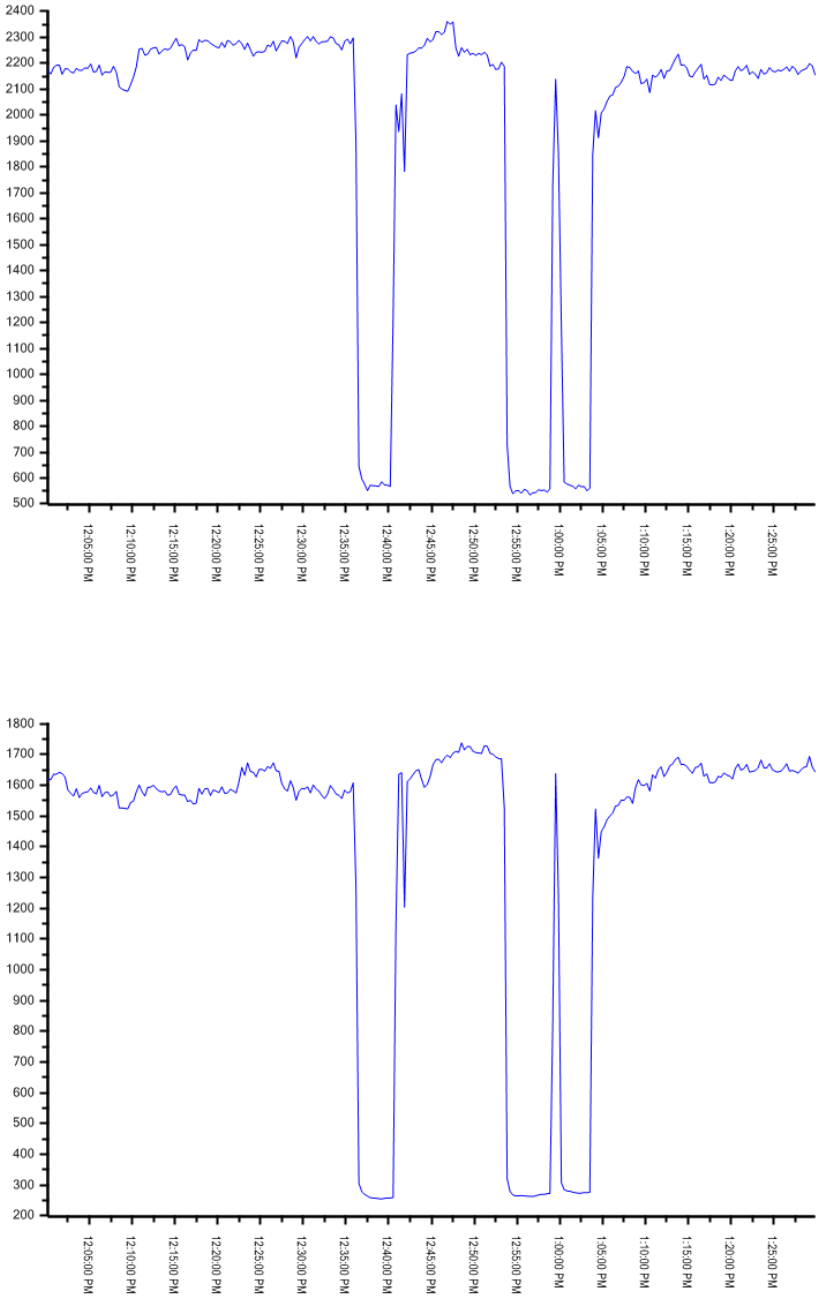

Figure C.9: Normal datasets for event 5a.

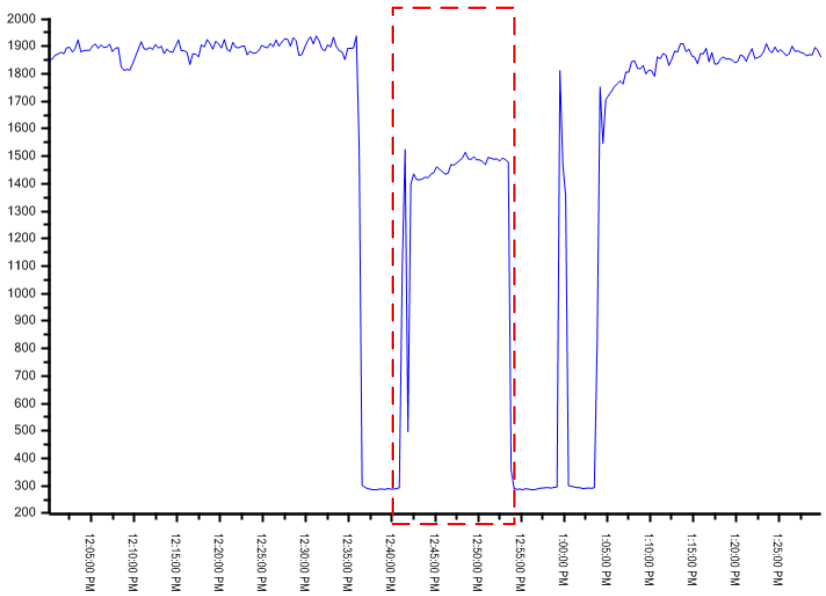

Figure C.10: Test dataset containing event 5a anomaly.

C.10 Event 10 Datasets

The "Event 10" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.19. The test dataset with a single highlighted anomalous region is shown in Figure C.20.

C.11 Event 11 Datasets

The "Event 11" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.21. The test dataset with four highlighted anomalous regions is shown in Figure C.22.

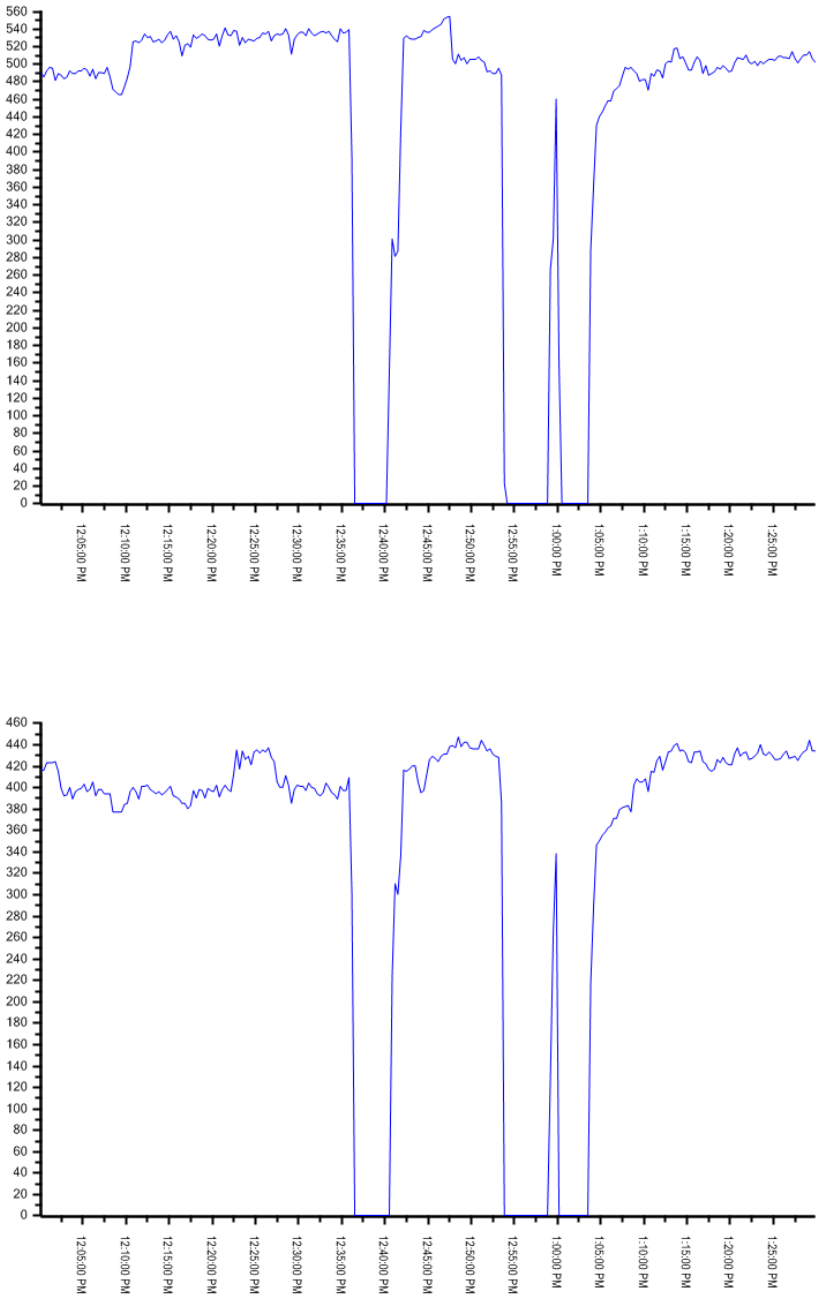

Figure C.11: Normal datasets for event 5b.

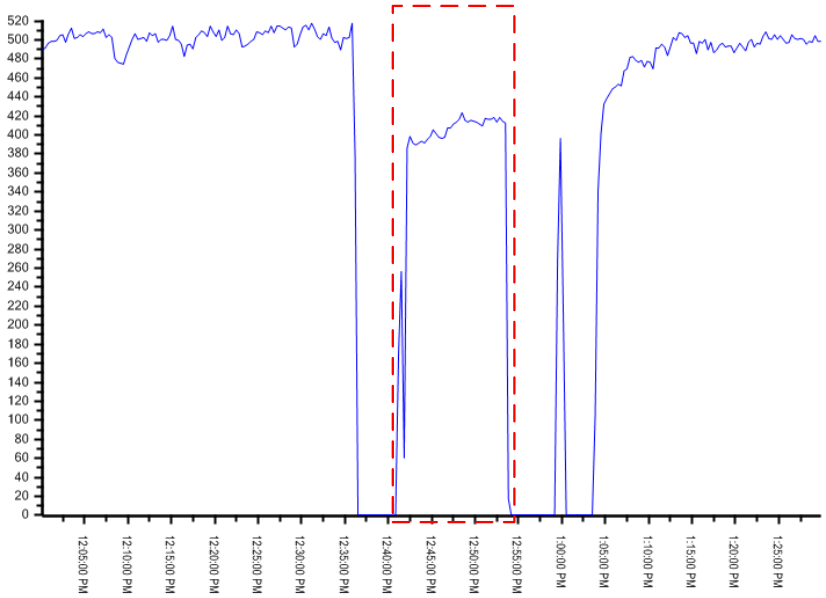

Figure C.12: Test dataset containing event 5b anomaly.

C.12 Event 14 Datasets

The "Event 14" datasets all use the NodeCount fiber data property. The 2 normal datasets used for training are shown in Figure C.23. The test dataset with a single highlighted anomalous region is shown in Figure C.24.

C.13 Event 15 Datasets

The "Event 15" datasets all use the Magnitude fiber data property. The 2 normal datasets used for training are shown in Figure C.25. The test dataset with a single highlighted anomalous region is shown in Figure C.26.

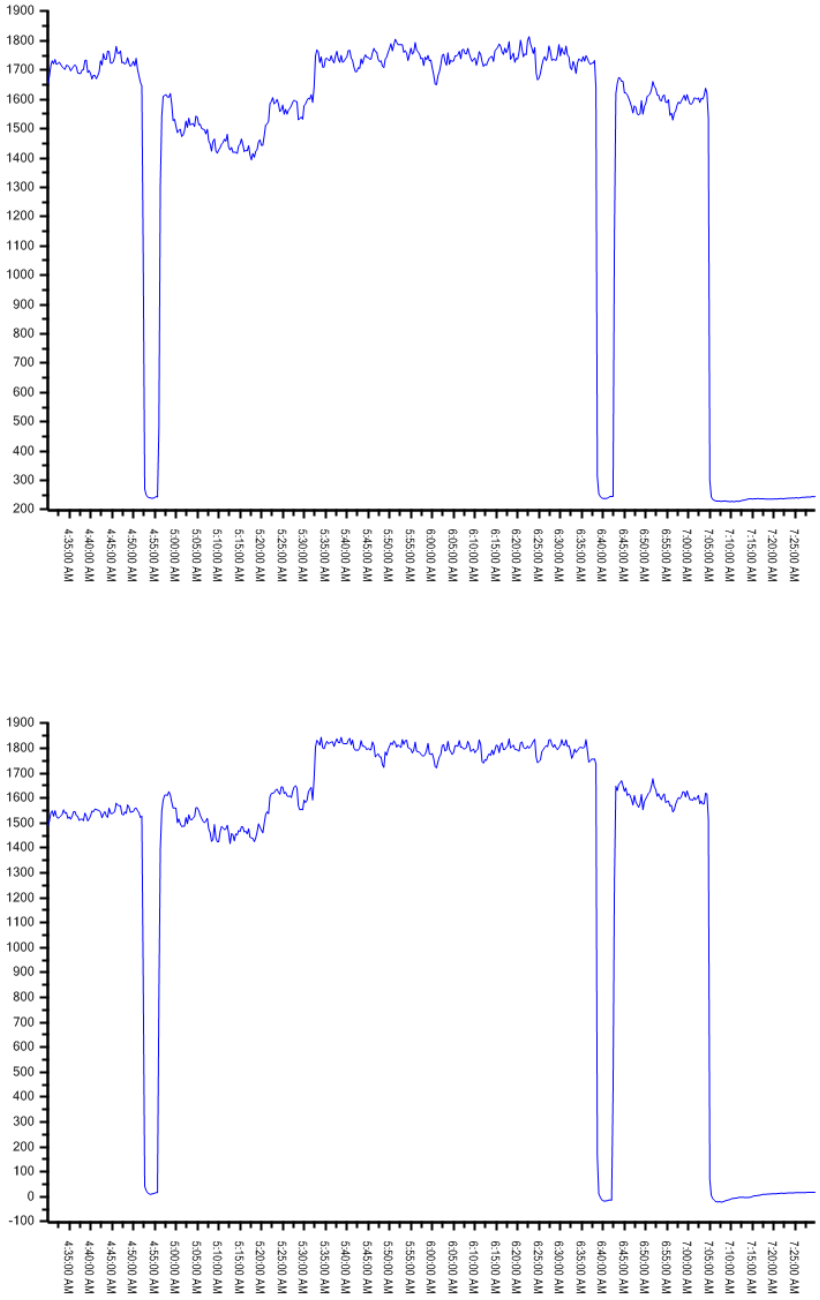

Figure C.13: Normal datasets for event 6.

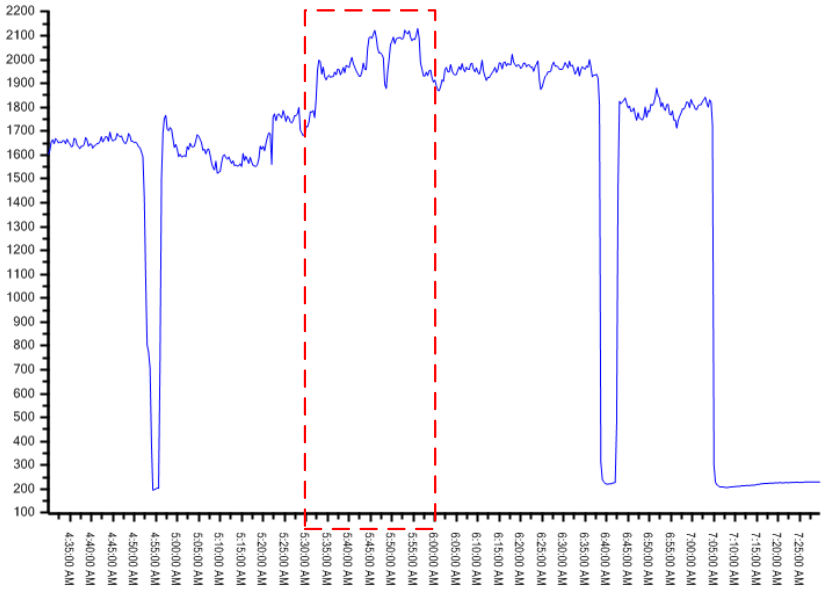

Figure C.14: Test dataset containing event 6 anomaly.

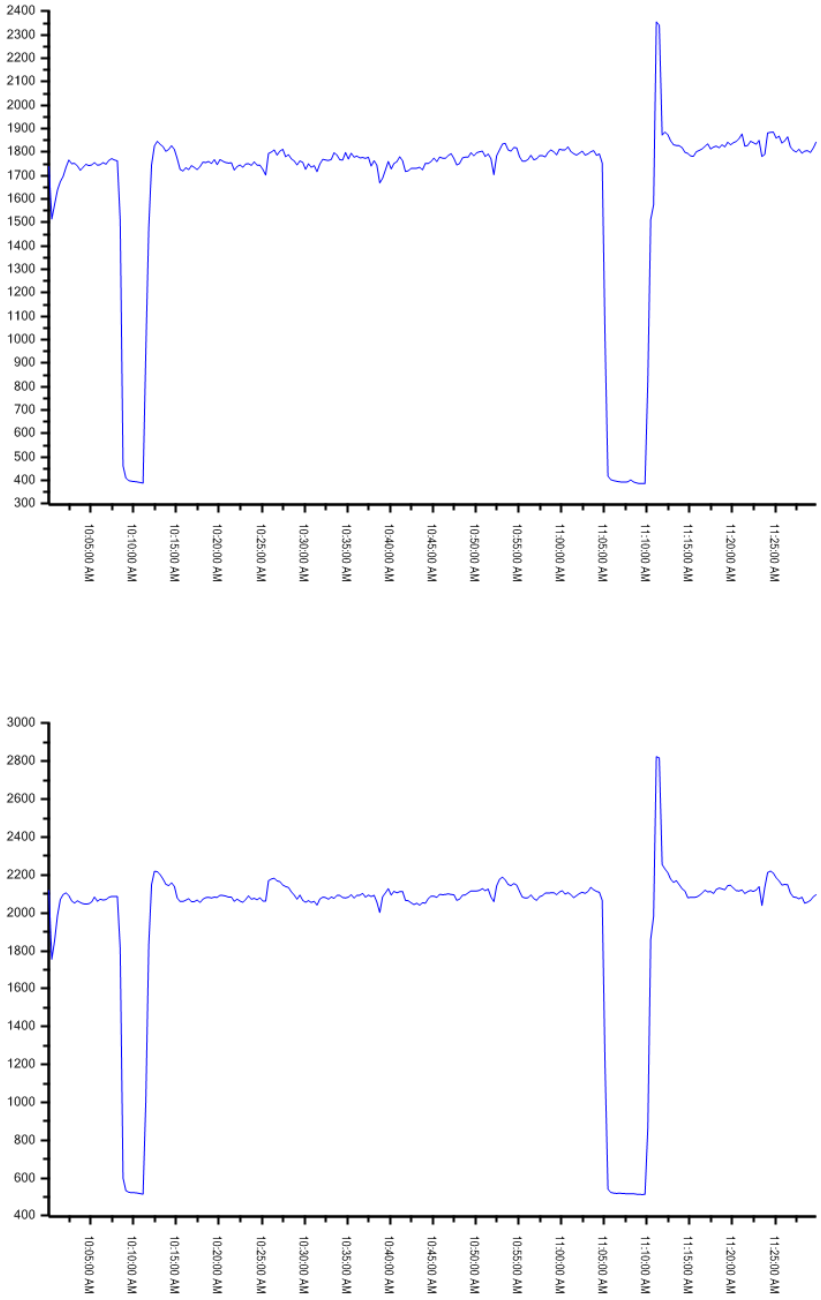

Figure C.15: Normal datasets for event 7.

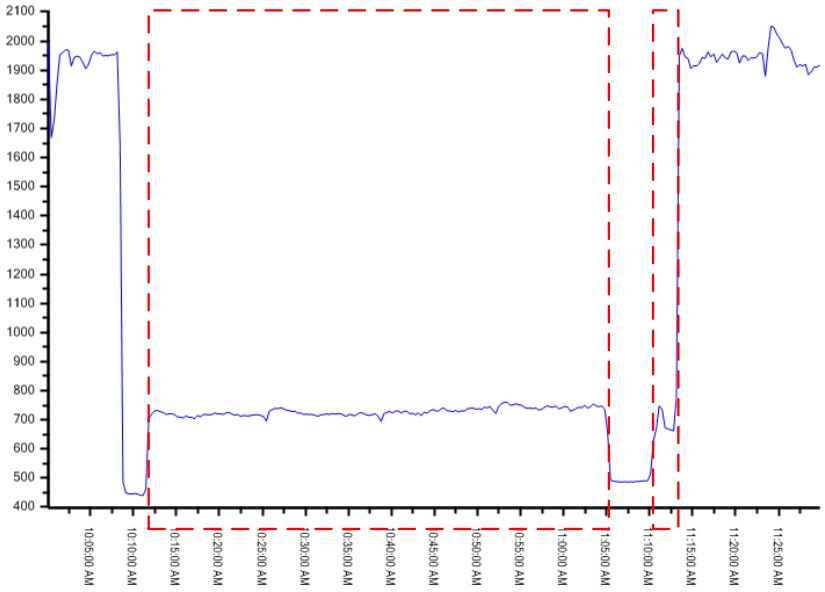

Figure C.16: Test dataset containing the event 7 anomalies.

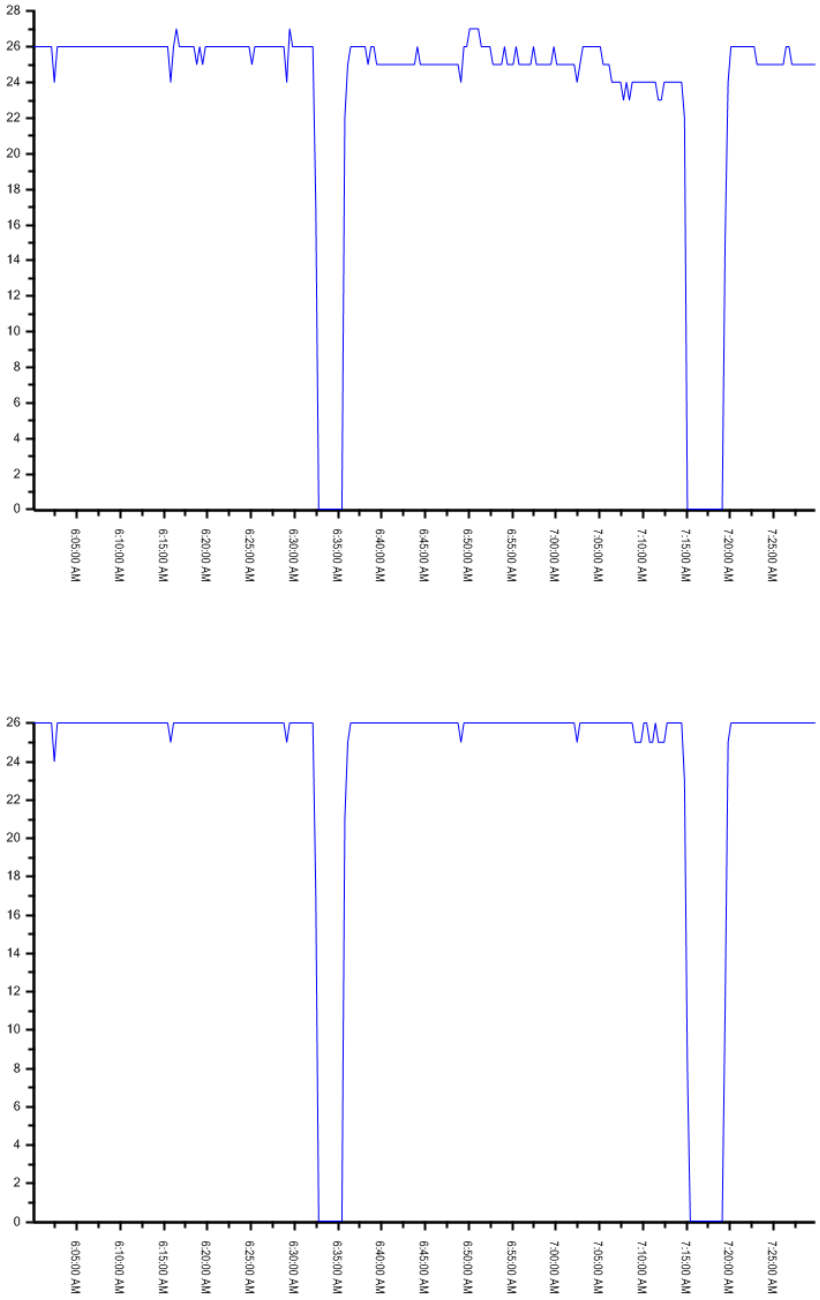

Figure C.17: Normal datasets for event 8.

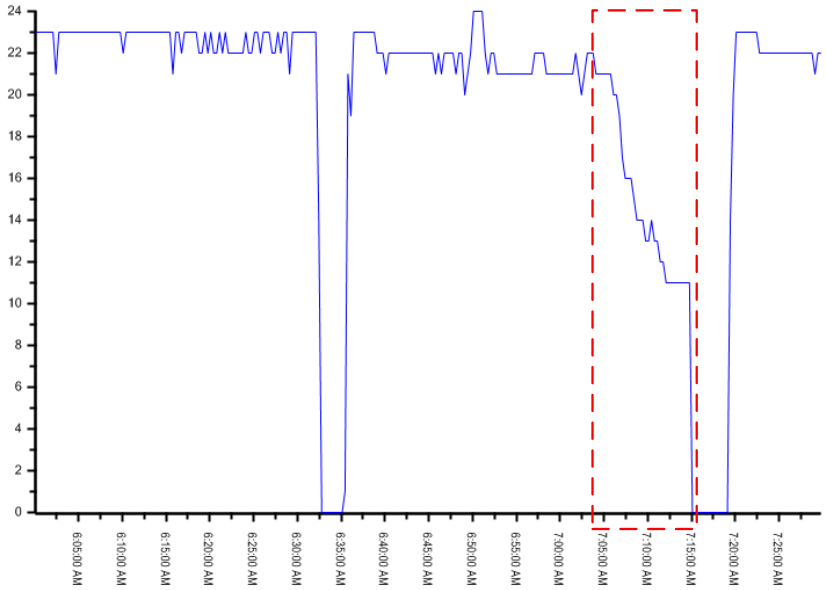

Figure C.18: Test dataset containing event 8 anomaly.

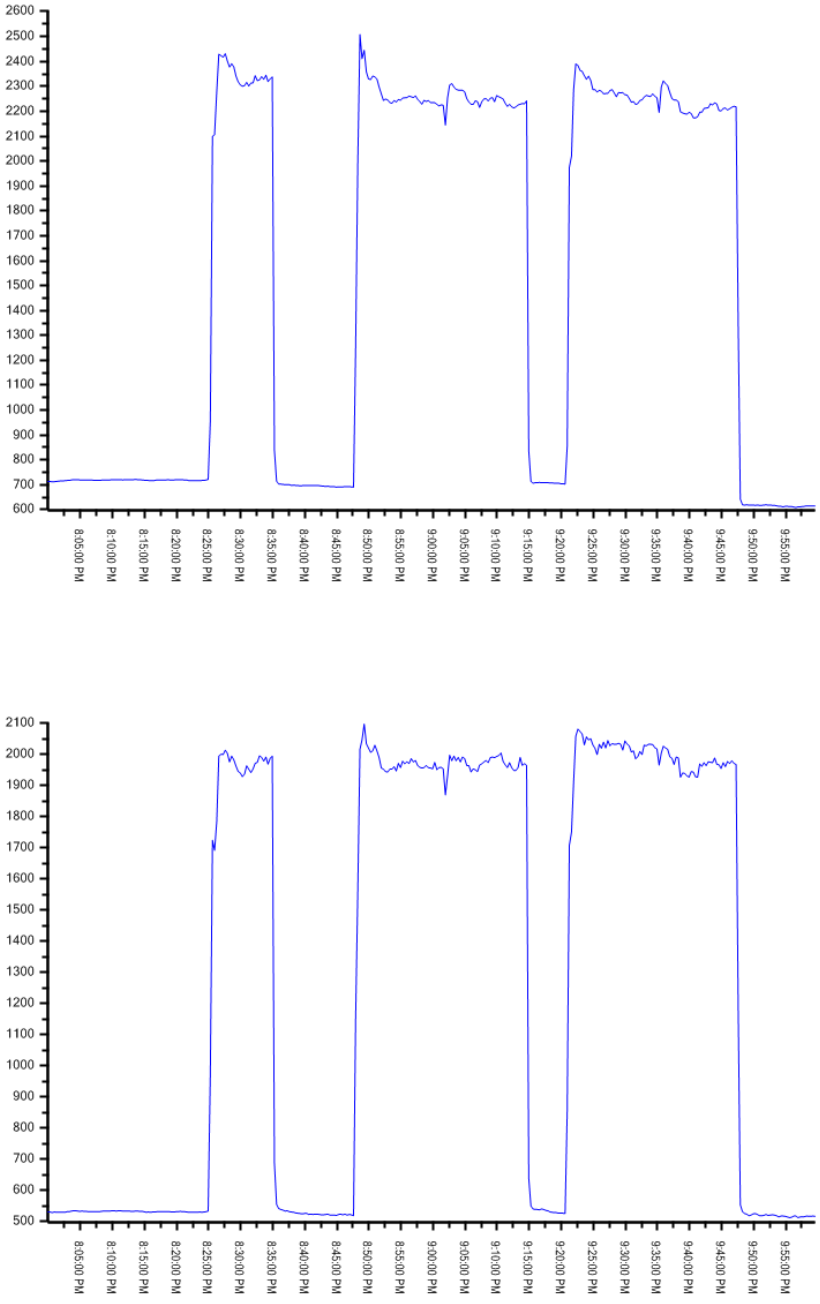

Figure C.19: Normal datasets for event 10.

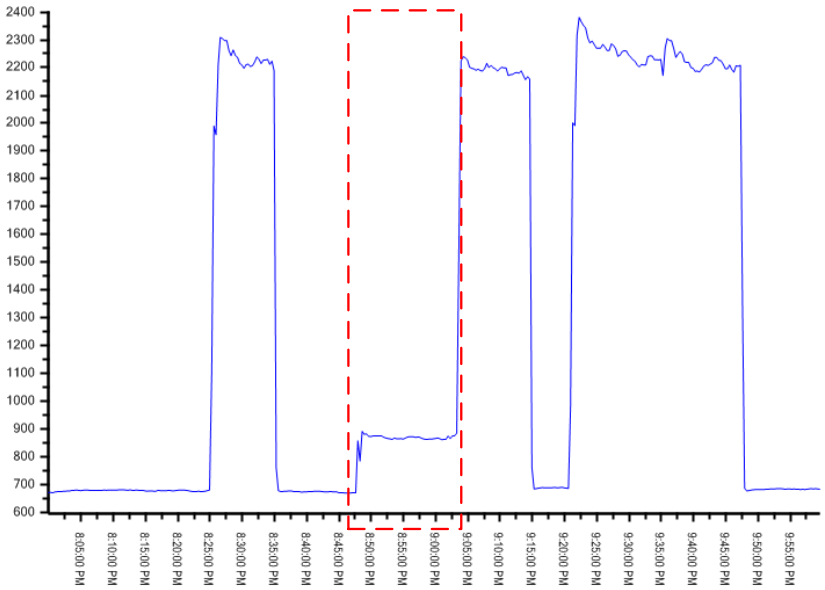

Figure C.20: Test dataset containing event 10 anomaly.

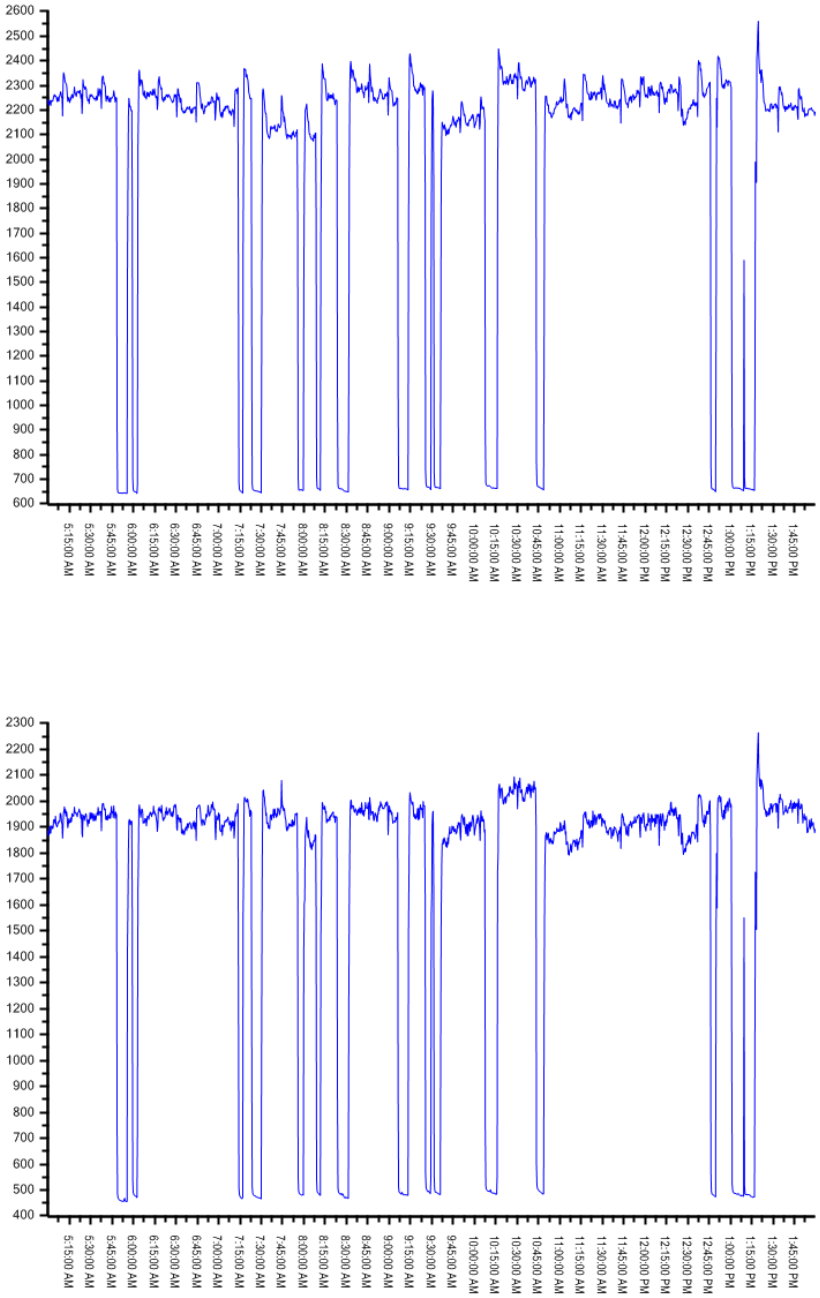

Figure C.21: Normal datasets for event 11.

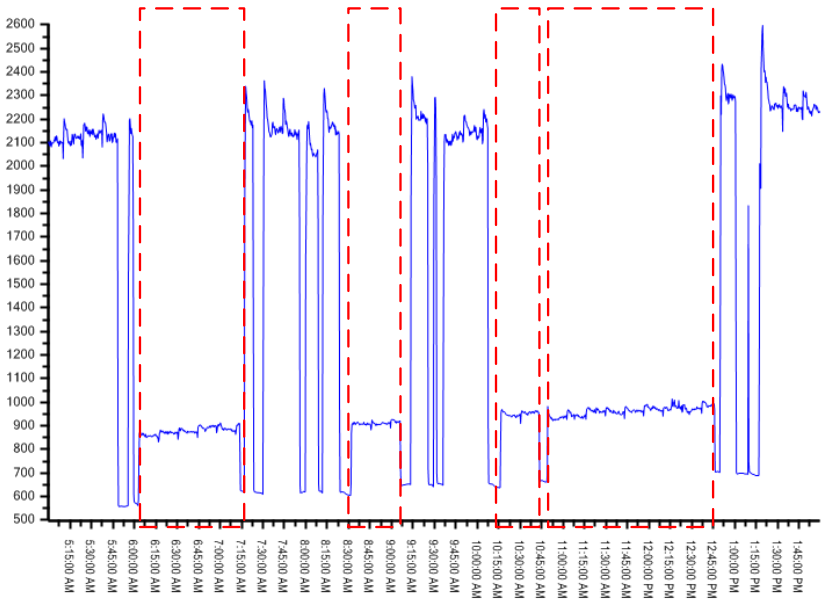

Figure C.22: Test dataset containing event 11 anomalies.

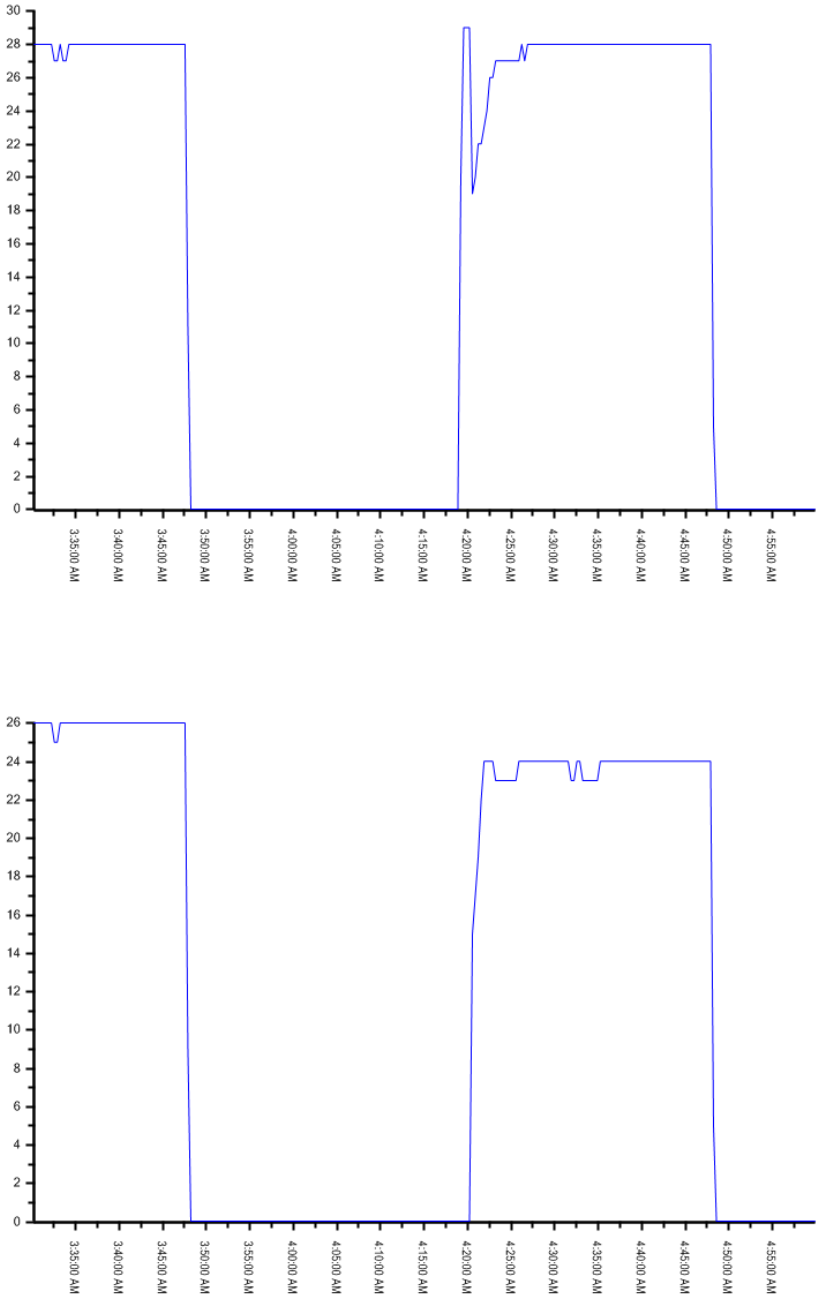

Figure C.23: Normal datasets for event 14.

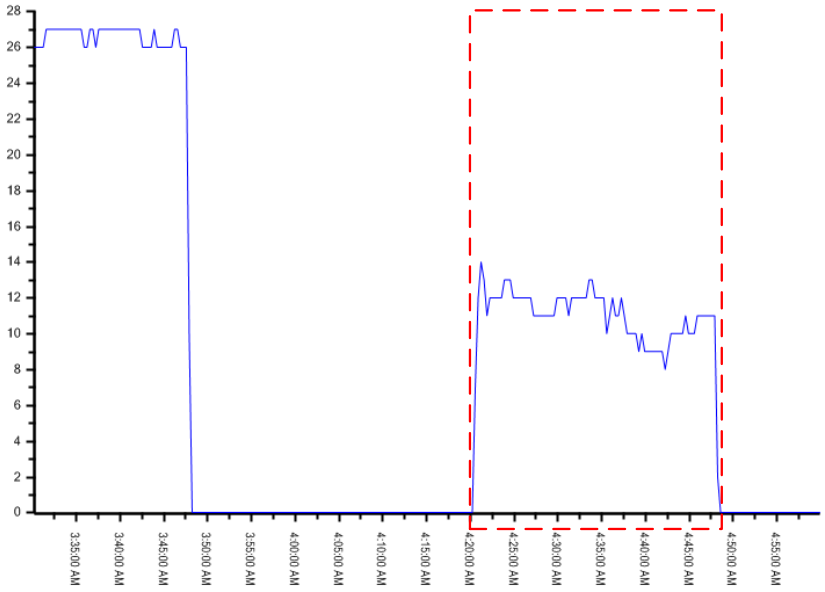

Figure C.24: Test dataset containing event 14 anomaly.

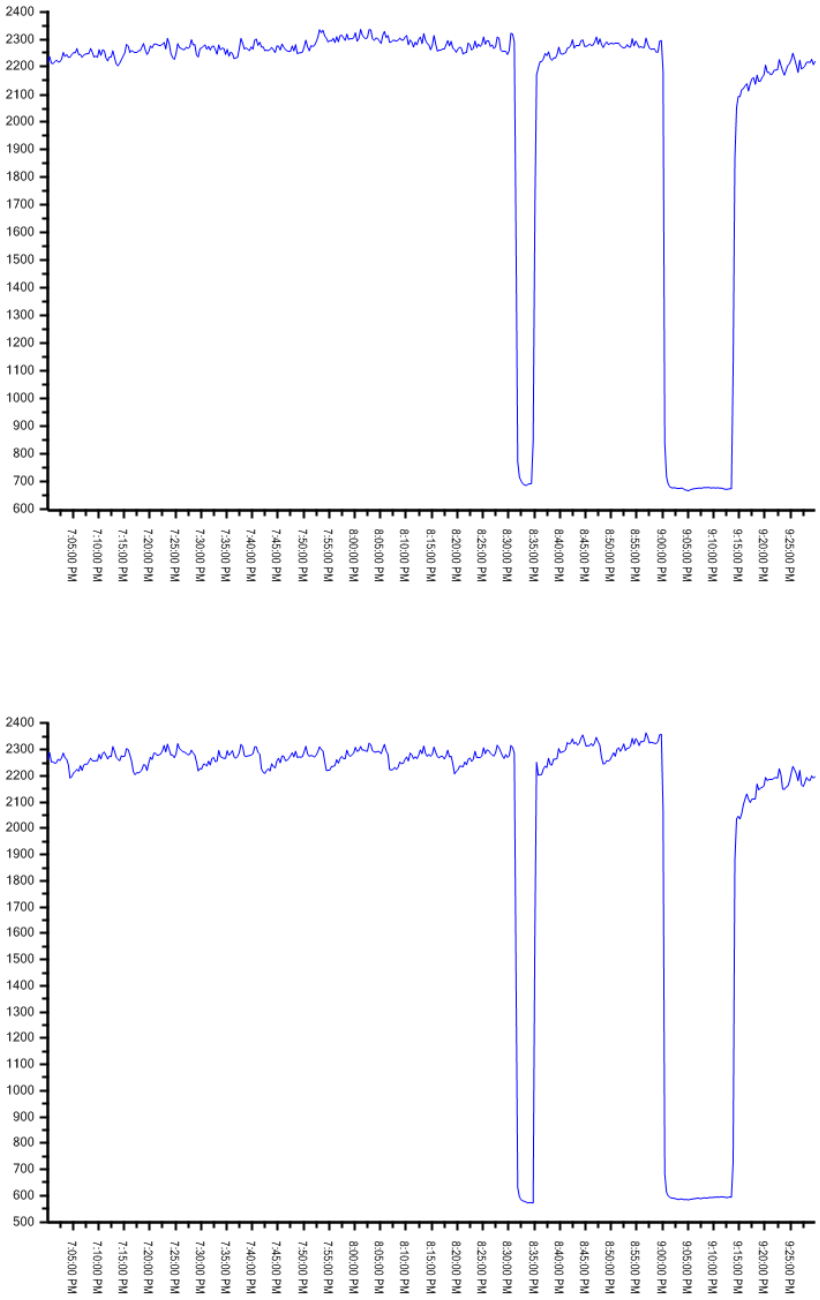

Figure C.25: Normal datasets for event 15.

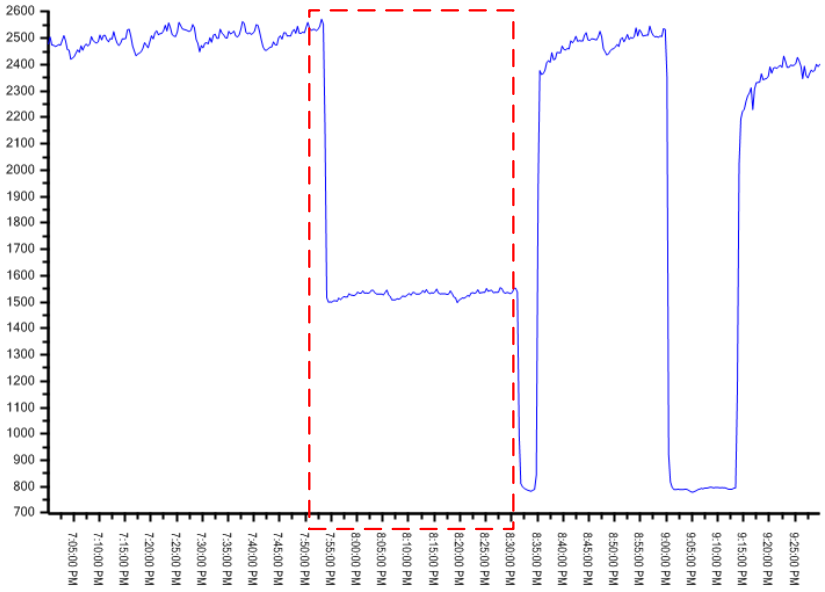

Figure C.26: Test dataset containing event 15 anomaly.

Bibliography

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO '93: Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, London, UK, 1993. Springer-Verlag.

[2] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 490–501, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[3] J. Anstey, D. Peters, and C. Dawson. Discovering novelty in time series data. In *NECEC '05: Proceedings of the 15th Annual Newfoundland Electrical and Computer Engineering Conference*, 2005.

[4] J. S. Anstey, D. K. Peters, and C. Dawson. An improved feature extraction technique for high volume time series data. In *SPPRA 2007: Proceedings of the Fourth IASTED International Conference on Signal Processing, Pattern Recognition, and Applications*, pages 74–81, Anaheim, CA, USA, 2007. ACTA Press.

[5] M. Chan. New online fiber sensor technology unlocks value in fiber manufacturing. *International Fiber Journal*, December 2000.

[6] M. Chan. Discovering the value of information: On-line monitoring of fiber characteristics. In *proceedings of the IFAI Technical Forum 2001*. IFAI, October 2001.

[7] M. Chan. Utilizing on-line information technology to revolutionize the production of fiber. *Chemical Fibers International*, April 2001.

[8] M. Chan. Instantaneous carpet production feedback. *International Fiber Journal*, November 2005.

[9] M. Chan. Maximizing benefits of online monitoring. *International Fiber Journal*, June 2006.

[10] P. K. Chan and M. V. Mahoney. Modeling multiple time series for anomaly detection. *ICDM '05: Proceedings of the 2005 IEEE International Conference on Data Mining*, 0:90–97, 2005.

[11] P. K. Chan and M. V. Mahoney. Trajectory boundary modeling of time series for anomaly detection. *Workshop on Data Mining Methods for Anomaly Detection, SIGKDD Conference*, 2005.

[12] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *Proceedings of the 5th International Conference on Intelligent Systems*, 1996.

[13] D. Dasgupta, Z. Ji, and F. Gonzlez. Artificial immune system (AIS) research in the last five years. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Canberra, Australia, 2003.

[14] I. Daubechies. *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[15] J. Dorrity, G. Vachtsevanos, and W. Jasper. Real-time fabric defect detection and control in weaving processes. In *National Textile Center Annual Report*, pages 113–122, 1996.

[16] U. EPA. *Compilation of Air Pollutant Emission Factors AP-42*, volume I, chapter 6: Organic Chemical Process Industry. U.S. Environmental Protection Agency, fifth edition, 1995.

[17] A. Fox, E. Kiciman, and D. Patterson. Combining statistical monitoring and predictable recovery for self-management. In *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 49–53, New York, NY, USA, 2004. ACM Press.

[18] K. Goebel, B. Wood, A. Agogino, and P. Jain. Comparing a neural-fuzzy scheme with a probabilistic neural network for applications to monitoring in manufacturing systems. In *Working Notes of the 1994 AAAI Spring Symposium: Detecting and Resolving Errors in Manufacturing Systems*, 1994.

[19] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2):50–61, 1995.

[20] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

[21] M. A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, 1998.

[22] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[23] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[24] A. C. Huang and A. Fox. Cheap recovery: a key to self-managing state. *ACM Transactions on Storage (TOS)*, 1(1):38–70, 2005.

[25] J. Hunter and N. McIntosh. Knowledge-based event detection in complex time series data. In *AIMDM '99: Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, pages 271–280, London, UK, 1999. Springer-Verlag.

[26] D. Isaac and C. Lynnes. Automated data quality assessment in the intelligent archive. White Paper Prepared for the Intelligent Data Understanding Program, January 2003.

[27] H. V. Jagadish, N. Koudas, and S. Muthukrishnan. Mining deviants in a time series database. In *VLDB '99: Proceedings of the 25th International Confer-*

ence on Very Large Data Bases, pages 102–113, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[28] E. Keogh. Mining and indexing time series data. In *The 2001 IEEE International Conference on Data Mining*, San Jose, CA, USA, 2001.

[29] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 102–111, New York, NY, USA, 2002. ACM Press.

[30] E. Keogh, S. Lonardi, and B. Chiu. Finding surprising patterns in a time series database in linear time and space. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 550–556, New York, NY, USA, 2002. ACM Press.

[31] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, pages 151–162, 2001.

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

[33] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 289–296, Washington, DC, USA, 2001. IEEE Computer Society.

[34] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining of concurrent text and time series. In *The 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Workshop on Text Mining*, pages 37–44, Boston, MA, USA, 2000.

[35] T. Li, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explorations Newsletter*, 4(2):49–68, 2002.

[36] X. Li. Fuzzy neural network and wavelet for tool condition monitoring. In *Computational Intelligence in Manufacturing Handbook*. CRC Press, 2001.

[37] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11, New York, NY, USA, 2003. ACM Press.

[38] J. Lin, E. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Visually mining and monitoring massive time series. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 460–469, New York, NY, USA, 2004. ACM Press.

[39] C. A. LLC. *Complete Textile Glossary*. Celanese Acetate LLC, fifth edition, 2001.

[40] J. Ma and S. Perkins. Online novelty detection on temporal sequences. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 613–618, New York, NY, USA, 2003. ACM Press.

[41] S. G. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[42] F. Mörchen. Time series feature extraction for data mining using DWT and DFT. Technical Report 33, Department of Mathematics and Computer Science, University of Marburg, Germany, 2003.

[43] A. L. Oliveira, F. B. Neto, and S. R. Meira. Combining MLP and RBF neural networks for novelty detection in short time series. In *MICAI 2004: Advances in Artificial Intelligence, Third Mexican International Conference on Artificial Intelligence*, pages 844–853, Mexico City, Mexico, 2004. Springer.

[44] S. Park, S.-W. Kim, and W. W. Chu. Segment-based approach for subsequence searches in sequence databases. In *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 248–252, New York, NY, USA, 2001. ACM Press.

[45] C.-S. Perng, H. Wang, S. R. Zhang, and D. S. Parker. Landmarks: A new model for similarity-based pattern querying in time series databases. In *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*, page 33, Washington, DC, USA, 2000. IEEE Computer Society.

[46] K. pong Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, page 126, Washington, DC, USA, 1999. IEEE Computer Society.

[47] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 212, Washington, DC, USA, 2002. IEEE Computer Society.

[48] S. Salvador, P. Chan, and J. Brodie. Learning states and rules for time series anomaly detection. In *Proceedings of the 17th international FLAIRS conference*, pages 300–305, 2004.

[49] C. Shahabi, X. Tian, and W. Zhao. TSA-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. In *SSDBM '00: Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM'00)*, page 55, Washington, DC, USA, 2000. IEEE Computer Society.

[50] T. Stibor, J. Timmis, and C. Eckert. On the appropriateness of negative selection defined over hamming shape-space as a network intrusion detection system. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 995–1002. IEEE Computer Society, 2005.

[51] D. Trowbridge, U. Roxburgh, G. Hohpe, D. Manolescu, and E. G. Nadhan. *Integration Patterns (Patterns & Practices)*. Microsoft Press, Redmond, WA, USA, 2004.

[52] J. Wang, W. S. Tang, and C. Roze. Neural network applications in intelligent manufacturing: An updated survey. In *Computational Intelligence in Manufacturing Handbook*. CRC Press, 2001.

[53] D. Wu, A. Singh, D. Agrawal, A. E. Abbadi, and T. R. Smith. Efficient retrieval for browsing large image databases. In *CIKM '96: Proceedings of the fifth international conference on Information and knowledge management*, pages 11–18, New York, NY, USA, 1996. ACM Press.

[54] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. A comparison of dft and dwt based similarity search in time-series databases. In *CIKM '00: Proceedings of the ninth international conference on Information and knowledge management*, pages 488–495, New York, NY, USA, 2000. ACM Press.

[55] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 385–394, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[56] Y. Zhu. *High Performance Data Mining in Time Series: Techniques and Case Studies*. PhD thesis, New York University, New York, 2004.