Automatic Detection of Abnormal Behavior in Computing Systems

THESIS

A thesis submitted in partial
fulfillment of the requirements for
the degree of Master of Science in
the College of Engineering at the
University of Kentucky

By
J. Frank Roberts
Lexington, Kentucky

Director: Dr. Raphael A. Finkel, Professor of Computer Science
Lexington, Kentucky 2013

ABSTRACT OF THESIS

Automatic Detection of Abnormal Behavior in Computing Systems

I present RAACD, a software suite that detects misbehaving computers in large computing systems and presents information about those machines to the system administrator. I build this system using preexisting anomaly detection techniques. I evaluate my methods using simple synthesized data, real data containing coerced abnormal behavior, and real data containing naturally occurring abnormal behavior. I find that the system adequately detects abnormal behavior and significantly reduces the amount of uninteresting computer health data presented to a system administrator.

KEYWORDS: Anomaly detection, computer system health monitoring, SAX

Author's signature: J. Frank Roberts

Date: April 30, 2013

Automatic Detection of Abnormal Behavior in Computing Systems

By
J. Frank Roberts

Director of Thesis: Raphael A. Finkel

Director of Graduate Studies: Raphael A. Finkel

Date: April 30, 2013

I dedicate this thesis to my grandparents, Joe and Loretta Roberts.

ACKNOWLEDGMENTS

I'd like to thank Professor Hank Dietz, who encouraged me to engage in research early in my undergraduate career. Dr. Dietz conceived NodeScape and has helped me maintain a practical focus in my research activities.

I'd like to thank Professor Raphael Finkel for guiding my research and keeping me focused, and for teaching me the proper style for technical writing.

I'd like to thank the Department of Computer Science and the KAOS research group for allowing me to run my monitoring software on their machines.

I'd like to thank my lab-mates Paul Eberhart and Matt Sparks for providing helpful discussion and general companionship as I've worked to complete my research. In particular, I'd like to thank Matt for providing the idea that eventually led to the name for RAACD.

I'd like to thank my parents, James and Donna Roberts, for instilling in me the motivation and discipline necessary to complete the goals that I set for myself.

I'd like to thank my grandparents, Joe and Loretta Roberts, for their patient and unconditional support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

This thesis presents a new abnormal-behavior detection scheme for networked computers. I base the design for this scheme on the anomaly detection scheme presented by Wei et al. [16]. I use the Symbolic Aggregate approXimation (SAX) method for discretizing a time series [11]. The SAX method converts a series of real-valued samples into a symbolic representation. I present three approaches to searching for anomalies in the symbolic representation: window-pair analysis, baseline analysis, and profile search. These approaches compute a series of anomaly scores, with higher values corresponding to more anomalous behavior. Window-pair analysis slides two concatenated windows across the series and computes the distance between the contents of the windows at each offset in the series. Baseline analysis slides a single window across the series and computes the distance between that window and a precomputed profile of the expected behavior for the series. A profile search looks for a particular anomaly by computing the distance between the contents of a sliding window and a precomputed profile for an anomaly. I use these three methods to build a multidimensional analysis algorithm for detecting abnormal behavior. I evaluate these methods using synthetic and real time series. The real time series include both natural and coerced behavior.

I implement this detection scheme in a software package called RAACD, pronounced "racked." RAACD stands for Roberts' Automatic Abnormal Conduct Detector. I chose this name for two reasons: the purpose of the software is to automatically detect abnormal behavior (or conduct), and the pronunciation has the convenient connotation of a machine room full of servers. I implement RAACD as an analysis and presentation package for use with NodeScape, a software package that monitors the health of a group of computers.

Throughout this thesis, I use the term anomalous or anomaly to refer to behavior that is notable with regard to the series in which it appears, but that a system administrator may not consider otherwise significant. I use the term abnormal to refer to behavior that the system administrator may find significant. Usually abnormal behavior indicates a problem with the machine on which it is observed. Anomalous behavior does not always indicate abnormal behavior, but abnormal behavior usually is also anomalous. I present the system administrator with information about hosts that behave anomalously, a superset of the machines that exhibit problematic or otherwise significant behavior.

Chapter 2 Relevance

Monitoring of machine health is important to the maintenance of any group of computers. Health monitoring allows administrators to prevent unplanned downtime by detecting and preemptively repairing potential problems. As our computing infrastructure continues to grow, administrators encounter more difficulty when trying to detect small problems. Our ability to collect and store data has scaled well with the size of our infrastructure, but our ability to analyze and present that data has not.

In spring 2011, Aggregate.org began research on how to better analyze and present computer monitoring data. **NodeScape** was the first product of this research [7]. The first version of NodeScape (**NodeScape v1**) presents health data for the nodes in a cluster as a painted image of that cluster. Our initial work on NodeScape v1 focused on finding ways to make interesting properties of monitoring data apparent to the viewer. NodeScape v1 solves this problem in the way that it presents the data. A viewer can easily see at a glance which machines have warm CPUs or which machines have a high load average. These machines stand out because they are colored differently from the rest of the cluster.

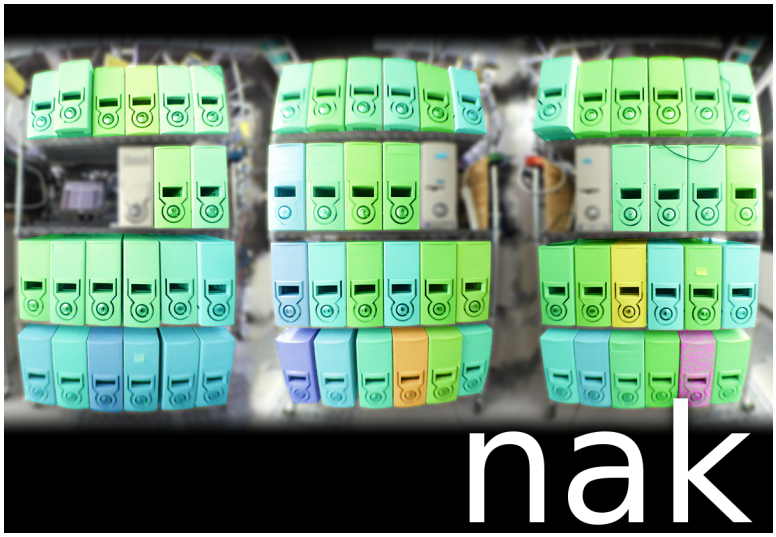

Figure 2.1: A temperature display from NodeScape v1

While this presentation format works well for observing values of a particular property across the nodes in a cluster at a particular moment in time, it is not useful for observing trends or detecting patterns of abnormal behavior. This presentation format also will not scale well. An administrator may reasonably scan the image of a 100-node cluster, but visually scanning the image or images of a several-thousand-node cluster would be cumbersome.

We must reduce the amount of data that our tools present to the system administrator. We can take advantage of the relative scarcity of abnormal behavior; most machines behave normally most of the time. The administrator does not consult a health monitoring system to see which machines are behaving normally; rather, the administrator uses the health monitoring system to look for unhealthy machines. We can build a better monitoring system by detecting abnormal behavior and presenting only that behavior to the administrator.

One approach to detecting abnormal behavior is to analyze the data for anomalies. Others have developed anomaly detection schemes for use in data mining and process control [6, 9, 16]. I extend one of these schemes to detect abnormal behavior in computing systems.

Chapter 3 Related Work

My work ties together research from two areas: computer monitoring systems and anomaly detection. My work is informed by results from both areas.

3.1 Other Monitoring Systems

Munin

Munin is a computer monitoring system written in Perl that is compatible with most Unix/Linux derivatives [3]. Munin uses a server-pull architecture; that is, the Munin master process, which runs on a central server, queries `munin-node` processes running on the monitored nodes. The set of properties that Munin monitors by default varies with the installation environment, but it generally includes memory and disk usage, system load, device latencies, processor, usage and process count. Users may extend `munin-node` by writing plugins to monitor additional properties of a computer system such as web, FTP, and mail services, custom database and filesystem tools, or virtualization environments. A `munin-node` plugin script must return a single value that corresponds to the current state of the property that it monitors. The Munin-master builds graphs from the data and presents them through an HTML web interface. Munin builds graphs for periods of one day, one week, one month, and one year.

Ganglia

Ganglia was designed specifically for use in high-performance computing environments [2]. Its implementation emphasizes efficiency in its algorithms and data structures, making Ganglia a popular choice for monitoring high-performance or large scale environments. Ganglia monitors the same properties as Munin. Like Munin, Ganglia uses a server-pull model for monitoring multiple machines. Ganglia provides a web interface for most configuration and usage, but also presents data in a command-line environment. Ganglia supports a hierarchical monitoring structure and scales to very large installations. Ganglia is in production use by many computing centers and corporations.

Cacti

Cacti[1] is a front-end for rrdtool[5], a graphing and logging tool also utilized by Munin and Ganglia. Cacti uses a polling system to collect data from remote hosts over SNMP. Cacti provides a full featured interface for creating graphs from monitoring data, and allows the user to define custom scripts for collecting data. The user may also define custom templates for adding new graphs, hosts, and data sources. Cacti provides a user account system for restricting access to certain data.

Nagios

Nagios is the self-declared industry standard in IT infrastructure monitoring [4]. Nagios presents information about machine resources and services via a web interface. Nagios features an extensive set of tools to detect outages in network services. The system administrator may define checks both for hosts and individual services on a host. Nagios uses a server-pull approach to collecting data. A Nagios monitoring server runs the specified checks on the remote machines and collects the results. Nagios determines from the result of the host checks whether a host is UP, DOWN, or UNREACHABLE. A service check returns either OK, WARNING, UNKNOWN, or CRITICAL. Services checks query include properties like temperature, memory usage, and system load as well as the state of software running on a host like an HTTP or FTP server. When an outage occurs for a host or service, Nagios notifies specific contacts or contact groups. Nagios allows the administrator to define custom plugins to perform both host and service checks.

Pulsar

Pulsar is a client-push tool for monitoring large Unix sites [8]. Pulsar has three components: a presenter, a scheduler, and pulse monitors. The scheduler runs on each machine that is to be monitored. A configuration file determines which pulse monitors the scheduler is to run on each host and how often. When a pulse monitor runs, it first invokes a command to gather data from the machine. The pulse monitor converts the result of the command (also called an alarm) to a discomfort level and reports that discomfort level to the presenter. The presenter runs on the administrator's machine and displays an icon for each alarm that it receives. The presenter colors the icons based on the reported discomfort levels. The presenter does not display icons for alarms with a discomfort level of zero. As the discomfort level rises, the presenter colors the alarm icon green, then yellow, then red. The administrator may get more information about an alarm by selecting the alarm.

NodeScape

The NodeScape computer monitoring system highlights important computer health information, reducing the amount of information presented to the humans responsible for monitoring large scale computer systems. Dr. Hank Dietz wrote the first version of NodeScape, NodeScape v1, for use in a compute cluster[7]. NodeScape v1 consists of a monitor and a collector. The monitor runs on each machine being monitored. It may run as a daemon, or it may be scheduled by an external program. The monitor is configured by the arguments passed when it is run.

The collector maintains an image of the cluster. The collector colors each node in the image based on a single property (e.g. CPU temperature, load average). The collector updates the image immediately upon receiving status information from the monitor processes. High status values correspond to redder colors in the image; low status values correspond to bluer colors. The collector determines the

"height" of a status value based on the distance between that value and the current and historical averages for that property.

We have written a second version of NodeScape. We designed Nodescape v2 to handle monitoring duties in any computing environment, not just clusters. Nodescape v2 separates the collection and storage of data from the analysis and presentation of data. The Nodescape v2 back-end handles the collection and storage of data, and is comprised of two components. The monitor runs periodically on each machine being monitored. Each time it runs, the monitor reads a list of commands to be executed. Each command measures some property and prints the measurement to standard output. The monitor runs these commands, captures the output, and sends the output, with the corresponding property label, to the collector.

The collector listens for updates from monitors. When the collector receives an update, it unpacks and stores the update in a database. Programs that analyze and present the collected data are called front-ends. NodeScape front-ends retrieve data from the database. This design allows multiple front-ends to share the same back-end.

3.2 Anomaly Detection and Time Series Analysis Methods

Motifs

Lin et al. [12] present a method for efficiently detecting frequently occurring patterns in a time series. The preprocessing for their method applies the Piecewise Aggregate Approximation (PAA) algorithm for reducing the dimensionality of a series. The SAX discretization method (Section 4.1) also applies PAA before before assigning symbols. The ability to detect frequently occurring patterns may also be useful for anomaly detection. The set of most frequently occurring patterns in a time series could be used as a profile for a time series. A time series containing a very different set of common patterns may be anomalous.

Novelty detection using immunology

The human immune system employs T-cells to distinguish the body's cells from foreign cells. This approach may be generalized and used to detect anomalous behavior in a time series [6]. This method begins by generating a multiset of strings that represent normal behavior for the series of interest. The method employs negative selection to detect anomalous behavior. New behavior in the series passes through a set of detectors that do not match the series. Thus, if new behavior matches any of the detectors, then that behavior cannot match the normal behavior for the series.

Dasgupta et al. test their implementation of this method in two situations [6]. First, they employ their method to detect tool breakage in a simulated milling operation. The time series begins with normal operation but contains irregular behavior toward the end, after the cutting tool has broken. During normal milling

operation, the detectors in their system experience no matches. However, the anomalous behavior caused by the broken tool matches a number of detectors, increasing the anomaly score. They also apply their method to detecting noise in a signal processing application. Again, the portion of the time series where the signal behaves normally matches no detectors, but small region of noise matches several detectors.

Use of pattern frequency to determine novelty

Instead of analyzing the structure of parts of the time series, this approach analyzes how often different structures occur in the time series. Keogh et al. [9] implement this approach by encoding the subwords of a discretized time series and the number of occurrences of each in a suffix tree. Their method the compares new time-series information, also encoded in a suffix tree, to the initial tree. A pattern is considered anomalous if the number of occurrences in the new data is significantly different from the number of occurrences in the old data.

Keogh et al. compare their approach with other anomaly-detection approaches, including the immunology based approach introduced by Dasgupta. They apply their method to a noisy sine wave containing a synthetic anomaly and power-demand data from a Dutch research facility. Of all the methods they present, theirs is the only method to detect the synthetic anomaly. They apply the method to an entire year's worth of power demand data. The three most anomalous weeks flagged by their algorithm each contain a national holiday.

Assumption-free anomaly detection

Wei et al. [16] present a technique for making anomalous behavior easier for humans to recognize. Their method presents the user with a graphical representation of a series of anomaly scores. Higher scores correspond to more anomalous behavior. Their examples demonstrate that the anomaly score contains an obvious rise surrounding anomalous behavior. Wei calls the method **assumption-free** because it is domain-agnostic; this method properly detects anomalies in data from a variety of domains. The authors demonstrate that this method correctly detects anomalies in ECG data.

3.3 Other uses of SAX

Identifying multi-headed attack tools by time signature

Often, network-attack tools combine multiple exploits into a single tool. These tools are called multi-headed attack tools. Multi-headed attack tools are difficult to distinguish from other attack tools. The conventional approach of fingerprinting attack tools based on the type of attack fails because multi-headed attack tools use different attacks against different hosts. Pouget et al. [13] propose that multi-headed attack tools can be identified by the timing and frequency of their attacks. Pouget's method collects timing information for different attacks across multiple hosts and detects attacks that demonstrate similar trends. Pouget uses a 7-symbol SAX method to discretize the attack information before searching for attack patterns with similar trends. Pouget transforms each time series into a single SAX word and computes distance between SAX words directly instead of counting subwords.

Trend-based Symbolic Approximation (TSX)

Although SAX conversion preserves information about the mean value of each segment, it does not preserve information about trends in the time series. Li et al. [10] present TSX for preserving information about both the mean value of each segment and the trends present in the time series. For some types of data, financial data in particular, it is important to preserve trends when discretizing the data.

In addition to calculating the mean of each segment, TSX conversion collects slope information about each segment. TSX applies similar steps to SAX to assign symbols based on the slope information, though instead of using breakpoints from a normal distribution, TSX sets breakpoints based on angles. TSX encodes the time series as a sequence of tuples. Each tuple contains the SAX symbol for that segment in addition to the TSX symbols assigned based on the slope information from that segment. TSX may be applicable to searching for abnormal behavior. TSX would allow RAACD to perform trend analysis.

Preserving patterns when anonymizing data

It is important to preserve privacy when releasing a dataset. K-anonymity is a conventional approach to anonymizing data. To k-anonymize a dataset, one removes information is removed from the dataset until each record is identical to at least k-1 other records [15]. Although k-anonymity preserves most information about the values in the dataset, it does not effectively preserve pattern information from the dataset [14]. Shang et al. [14] propose an approach called (k,P)-anonymity that publishes value data and pattern data separately. The (k,P)-anonymity model uses SAX to encode information about patterns present in the dataset.

3.4 Connections to Related Work

I have chosen to implement Wei's method as the core of my abnormality detection scheme. Unfortunately, I found that the magnitude of the score produced by the assumption-free method varies significantly with the configuration parameters and with the properties of the series being analyzed. This inconsistency makes it very difficult to automatically detect when there is anomalous behavior. My methods extend this technique so that it can be used to detect anomalous and abnormal behavior automatically.

Besides NodeScape and Pulsar, I am not aware of any freely available or widely used software package that provides health monitoring capabilities. Munin, Ganglia, Cacti, and other similar packages provide ways to access and visualize computer monitoring data, but they do not analyze or provide explicit information about machine health. My package, RAACD, pairs behavior analysis and anomaly detection techniques with the presentation paradigms of common computer monitoring packages.

Chapter 4 Algorithms for Anomaly Detection

At its core, my method for detecting anomalies and abnormalities is based on the anomaly detection algorithm presented by Wei et al. [16]. Wei presents his algorithm as a method for automatically marking parts of a time series for further inspection by either humans or by other anomaly-detection algorithms. I employ two significant elements from the assumption-free method: symbolic aggregate approximation and subword-count histograms.

4.1 Symbolic Aggregate Approximation

Symbolic Aggregate approXimation (SAX) is a technique for converting a real-valued time series to a discrete representation [11]. A benefit of using a discrete representation is that it summarizes the time series and makes it easier to classify parts of the series. The assumption-free method employs SAX because SAX is well suited to data-mining tasks and because distance calculations in the SAX domain can be used to derive lower bounds for distances in the original series [11].

SAX starts by dividing the series into short subregions, usually fewer than 10 samples long. SAX assigns a symbol to each subregion based on where the average value of the subregion falls in the normal distribution. Symbols are assigned so that each symbol has equal probability of appearing at a given location in the symbolic representation. SAX does not process the entire series at once. Instead, SAX builds a list of words, with one word beginning at each sample in the time series.

Table 4.1: SAX conversion example

Sample Value	Normalized Value	Symbol Average	Symbol Assigned
25	1.73	1.43	d
24	1.58		
20	0.98		
15	0.23	-0.08	b
12	-0.23		
12	-0.23		
12	-0.23	-0.23	b
13	-0.08		
11	-0.38		
9	-0.68	-1.13	a
6	-1.13		
3	-1.58		

Table 4.1 shows an example of how the SAX method assigns symbols to parts of a time series. This example converts a 12-sample long region of a larger series to the word ”dbba”. First, SAX normalizes the entire region by subtracting from every sample the average value of the region and then dividing by the standard deviation. SAX then divides the region into subregions of length 3 and computes the average value of each subregion. SAX assigns a symbol to each subregion based on the break points listed in Table 4.2.

Table 4.2: Breakpoints based on the normal distribution

Symbol	Range
a	$<$ -0.675
b	-0.675 .. 0.0
c	0.0 .. 0.675
d	$>$ 0.675

4.2 Computing an Anomaly Score

Before computing the distance between two series, SAX build histograms of the appearance of subwords, that is, sequences of symbols with a given length. For example, the SAX word ”abbbd” contains 3 subwords: ”ab” with count of 1, ”bb” with a count of 2, and ”bd” with a count of 1. SAX normalizes the histograms to have a maximum value of 1 so that series of different lengths may be compared. SAX subtracts corresponding subword counts between the two histograms to obtain a series of distances. Finally, SAX computes the 2-norm of the series of distances. This computation produces a single **anomaly score**. I include pseudo-code for this process in listings 4.1 and 4.2.

```
1 func count_subwords(words, subword_length):
2
3   subword_count = init_map(subword_length)
4
5   for word in words:
6     for i from 0 to word.length - subword_length - 1:
7       subword_count[word[i:i+subword_length] += 1
8
9   max = maxval(subword_count)
10  for subword in subword_count:
11    subword_count[subword] /= max
12
13  return subword_count
```

Listing 4.1: Algorithm to compute subword count from a list of words

```
1 func histogram_distance(A, B):
2   dist = 0
3   for subword in union(A, B):
4     dist += (A[subword] - B[subword]) ** 2
5   return dist
```

Listing 4.2: Algorithm to compute the distance between two subword histograms

Table 4.3: Computing an anomaly score

Subword	Count 1 (Normalized)		Count 2 (Normalized)		Difference (Squared)	
aa	4	(1)	3	(0.6)	0.4	(0.16)
ab	2	(0.5)	3	(0.6)	-0.1	(0.01)
bc	1	(0.25)	1	(0.2)	0.05	(0.025)
cc	1	(0.25)	5	(1)	-0.75	(0.5625)
cd	2	(0.5)	0	(0)	0.5	(0.25)
					Total	1.0075

In Table 4.3 I document the steps for computing the distance between two subword histograms. The columns Count 1 and Count 2 refer to the subword counts in the first and second histograms, respectively. The histograms are sparse, containing instances of only five subwords between them. The table shows the unnormalized count as well as the normalized count for each subword. For each possible subword I compute the difference between the two histograms. The total distance (the anomaly score) is the sum of the squared differences.

I use this method to implement three anomaly-detection techniques. Each technique compares a sliding window, which I refer to as the **inspection window**, from one time series to another time series. Window-pair analysis compares the window to another part of the same series; baseline analysis and the profile-search technique use precomputed subword histograms.

4.3 Characteristics of Subword-count Histograms

Understanding the behavior of a series of anomaly scores requires some explanation of the behavior of subword-count histograms. In particular, normalization, subword variety, and the exponentiation in the distance calculation have significant effects on the final anomaly score for a pair of histograms.

Normalization

The maximum value in a normalized subword-count histogram is always 1. If multiple subwords share the highest unnormalized count, then they share the value 1 in the normalized histogram. If one subword significantly outnumbers the rest of the subwords, that subword may be the only one to make a meaningful contribution to the anomaly score.

Subword variety

The inspection window contains a fixed number of subwords of a particular length. If the count for one subword increases then the count must decrease for some other subword. The sum of the normalized counts increases with the number of unique subwords in the inspection window. This relationship means that inspection windows with a larger variety of subwords tend to cause the anomaly score to increase regardless of the contents of the second histogram. A histogram containing a large variety of subwords necessarily has a lower maximum subword count. In this situation, all subwords contribute more evenly to the total distance because each subword count is closer to the maximum subword count.

Exponentiation in the distance calculation

The formula for calculating the distance between two subword histograms squares the difference between the normalized counts for a particular subword. A linear increase in the difference for a particular subword causes a quadratic increase in that subword's contribution to the distance. The interaction between this method and the effects of differences in subword variety has dramatic effects on the behavior of the anomaly-score. For example, if the unnormalized maximum subword count decreases by 1, the contributions from all of the other subwords in the histogram increase quadratically, even though only one of the other unnormalized subword counts increases.

4.4 Window-pair Analysis

Window-pair (WP) analysis is an implementation of the assumption-free anomaly detection method. WP analysis slides two concatenated windows, the lead and lag windows, across the series. In WP analysis, the inspection window is the lead window. At each time step in the series, WP analysis builds a subword histogram for each window and computes the distance between the two histograms. WP analysis associates the anomaly score with the border between the two windows. The lead window should be long enough to capture entire anomalous events. The lag window should be at least as long as the lead window.

WP analysis treats the lag window as a reference for normal behavior. However, the distance computation does not differentiate between the two windows. If an anomaly appears in the lag window but no anomaly appears in the lead window, then the distance between the two windows is the same as when the anomaly appears in the lead window. When analyzing a series that contains a clear anomaly, WP analysis produces a **double peak** in the anomaly-score vector surrounding the anomaly. The first peak occurs due to the anomaly appearing in the lead window. As the anomaly begins to straddle the two windows, they being to look more alike, and the anomaly score falls. Once the anomaly has moved entirely into the lag window, the anomaly score rises again, resulting in a second peak.

The ratio of the lengths of the two windows affects the signature of an anomaly in the vector of anomaly scores. When the two windows are the same length, we see a symmetric double peak. As we increase the length of the lag window, we see the second peak decrease in height. The longer the lag window contains enough normal behavior to outweigh a short anomaly.

4.5 Baseline Analysis

Baseline analysis slides a single window across the series. Wei et al. present a similar method, called the "unsupervised" method, with their assumption-free method [16]. At each time step in the series, baseline analysis computes the subword histogram for the inspection window and computes the distance from that histogram to a precomputed subword histogram. The precomputed histogram is built from a sample of the expected baseline behavior for the series. The distance between the two histograms is recorded as the anomaly score for the approximate midpoint of the current inspection window. I discuss the method for associating anomaly scores with particular samples in section 4.8.

4.6 Profile Search

A **profile search** works the same way as baseline analysis, only the anomaly score is computed differently, because the comparison histogram does not represent normal behavior. After computing the anomaly score for all time steps, I subtract the maximum distance from every score, shifting all anomaly scores to be on or below the x-axis. I multiply all anomaly scores by -1 to reflect the distance curve across the x-axis so that higher scores correspond to smaller distances between the two histograms.

4.7 Multi-property Search

The above methods are intended to automatically detect anomalous behavior. I am able to detect abnormal behavior by applying these methods to analyze multiple properties on a single host at once. After running baseline analysis on the time series for each property for a particular host, I normalize each anomaly-score vector to have a maximum value of one. I then look to see if multiple vectors exceed some threshold value at the same place in the series. If enough vectors exceed the threshold in the same place, then this machine is considered to be exhibiting abnormal behavior.

To make this approach work, I must be able to ignore scores that remain high as a result of a near single-valued series. Series containing very little variation tend to have consistently high values for the anomaly score. To work around this problem, I ignore any series that has a small standard deviation.

4.8 Computing Ideal Offset

Each anomaly score must be associated with a particular position in the series. An anomalous sample begins to affect the anomaly score as soon as the leading word in the inspection window includes that sample. In general, the number of samples which the algorithm looks forward from the first sample in the inspection window is:

$$V = W * S + I - 1 \tag{4.1}$$

where:

V = view length (samples)
I = inspection window length (samples)
W = word length (symbols)
S = samples per symbol

I call this value the inspection window's **view**. The view includes every sample beginning with the first letter of the first word in the inspection window and continuing forward in the time series to the last sample of the leading word in the inspection window. Because the search methods analyze whole words at a time, words that begin close to the end of the inspection window contain information from samples past the end of the inspection window. We account for this by adding the length of the leading word in the inspection window to the length of the inspection window. We subtract one from the total because the last sample in the inspection window is the first sample in the leading word of the inspection window.

Using 3 samples per symbol, a window length of 3 samples, and a word length of 6 symbols, the algorithm sees 20 samples from the start of the inspection window:

$$6 * 3 + 3 - 1 = 20$$

In Figure 4.1, the window begins at sample 89 and extends 3 samples; it considers the words beginning at samples 89, 90, and 91. Each symbol in the word beginning at sample 91 represents an average of three samples. The last symbol in the word ends at sample 108, which is the 20th sample from the start of the window.

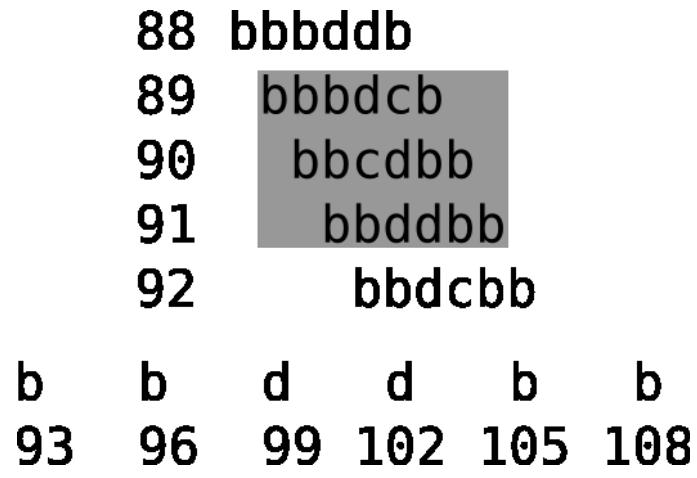

Figure 4.1: A visualization of inspection-window view

For baseline analysis, we associate an anomaly score with the center of the current view. To compute the offset of the anomaly score from the beginning of the inspection window, we divide the view length (equation 4.1) by 2:

$$O = (W * S + I - 1)/2 \tag{4.2}$$

where:

O = offset

We use a different formula for associating scores with anomalies in WP analysis. WP analysis generates a double peak for a point anomaly. We would like to center the trough between the peaks around the corresponding anomaly:

$$O = S * W/2 \tag{4.3}$$

We modify equation 4.2 to move the score backwards half of one inspection-window length. Because we do not add the inspection-window length, we are no longer double counting the last sample in the window, so we do not subtract 1.

Chapter 5 Implementation

I have built several tools to study the behavior of my anomaly detection methods. The tools depend on two libraries, which I also wrote. The first library, **nsutil**, provides functions that handle NodeScape v2 monitoring data. These functions allow my tools to read NodeScape and SAX configuration parameters from a file, to retrieve data from a database or from a local file, to write the data series and distance vectors to a file, and to generate plots from the data series and distance vectors. The second library, **saxutil**, provides functions that implement SAX, that generate subword count histograms from SAX words, and that compute the distance between subword histograms. I wrote my tools and the two libraries in the Go programming language.

I present three tools here. They are `build-profile`, `multi-search`, and `check-prop`. `Multi-search` performs WP analysis, baseline analysis, and profile search. `Build-profile` computes a subword histogram from a series of raw data and stores it for later use. `Check-prop` uses baseline analysis to complete a multi-dimensional search across all properties for each host.

`Build-profile` retrieves a single series from either a file or a database and converts it to a series of SAX words. A configuration file provides a list of subword lengths. `Build-profile` counts the occurrences of subwords of each length for the entire list of words. `Build-profile` writes the unnormalized subword histogram out to a file.

`Multi-search` uses a command-line flag to determine whether to retrieve the series of interest from a local file or from a database. A configuration file includes a list of baselines to compare against and a list of profiles to search for. `Multi-search` reads the precomputed baselines and profiles into subword histograms. `Multi-search` then converts the series of interest into a list of SAX words. The conversion function normalizes the subregion of samples for each word to have a mean of 0 and a standard deviation of 1 before assigning symbols to the region. The conversion function generates one word for each sample in the series. When the conversion nears the end of the series, it generates shorter words that only go up to the end of the series. The configuration file provides a list of subword lengths to process. `Multi-search` generates a subword count for each subword length for both the initial lead and lag windows. `Multi-search` also generates ordered lists of all possible subwords for the given lengths. `Multi-search` uses these lists to control iteration over the subword histograms, because Go does not guarantee consistent ordering when iterating over maps. `Multi-search` slides the lead and lag windows through the series one sample at a time. For each sample, `multi-search` adjusts the lead and lag subword histograms to add one new word and remove one old word. At each sample, `multi-search` normalizes the lead and lag subword histograms to have a maximum value of 1. Normalization allows `multi-search` to compare subword counts from windows of different lengths. `Multi-search` computes the

distance between the lead and lag subword histograms as well as the distance between the lead window and each of the baseline and profile subword histograms. `Multi-search` computes separate distances for each subword length and multiplies them together to obtain the final anomaly score. `Multi-search` applies equation (4.2) to associate the anomaly score for each location of the lead window with a particular sample. `Multi-search` continues moving the lead and lag windows through the series until the leading edge of the lead window reaches the end of the series. `Multi-search` then writes the series, the WP analysis distance vector, and the distance vectors for each baseline and profile to a file. Finally, `multi-search` invokes gnuplot to generate plots of the original series and of all the distance vectors.

`Check-prop` implements multi-property analysis on top of baseline analysis and WP analysis. `Check-prop` begins by retrieving a list of all known hosts from the NodeScape database. For each host, `check-prop` queries the database for all properties being monitored on that host. For each property, `check-prop` looks in the local filesystem for pre-computed baseline profiles. `Check-prop` performs baseline analysis for each baseline profile that it finds and normalizes the resulting distance vector to have maximum value of 1. Normalization allows `check-prop` to easily determine which regions of a distance vector are most anomalous. If `check-prop` does not find any baseline profiles, it continues to the next property. After computing distance vectors for all of the properties of a host, `check-prop` simultaneously scans all of the distance vectors. If it finds one location were three or more distance vectors have a value above 0.6, it flags this host's behavior as abnormal.

`Check-prop` also performs WP analysis on each property, though it does not consider the WP anomaly score when searching for abnormal behavior. Near single-valued series often have consistently high (> 0.6) anomaly scores. `Check-prop` ignores these series because they make a host's behavior more likely to look abnormal even though they do contain any abnormal behavior. `Check-prop` ignores any property series that has a very low standard deviation.

`Check-prop` handles the analysis portion of a NodeScape v2 front-end. When `check-prop` finds a host to be behaving abnormally, it generates plots of the series for each property, the WP analysis distance vector, and the distance vectors for all precomputed baseline profiles. It also generates an HTML document for each host that includes these plots. Finally, `check-prop` generates an HTML file containing the list of abnormally behaving hosts and thumbnails for the first four property and anomaly plots. A cron job invokes a script every 10 minutes that invokes `check-prop`.

Chapter 6 Evaluation

6.1 Detecting Anomalies in a Single Series

I test my methods using both real and synthesized data. I use simple, synthesized data to test for basic functionality. The set of synthesized data includes an impulse series, a unit step series, a sine wave series that changes frequency, a series containing a slow rise and fall, a series of uniformly distributed random data, and a series of data taken from a normal distribution.

I begin each test by synthesizing a time series. The synthesized time series contains either a specific anomaly or samples from a random distribution and is approximately 200 samples in length. The series are analyzed for several configurations using both the WP and baseline analysis methods. The configurations vary four parameters: inspection window length, word length, symbol size, and subword lengths.

I build a baseline profile for each test from a portion of the series that does not contain the anomaly. The length of the window used in baseline analysis is the same as the length of the lead window in WP analysis. I test each synthetic series with `multi-search`. For tests with multiple subword lengths, `multi-search` multiplies together the anomaly scores for all subword lengths to obtain a single anomaly-score vector. For example, the anomaly-score vector produced by configuration 1 (Table 6.1) is the element-wise multiplication of the length-2 anomaly-score vector by the length-4 anomaly-score vector. For each test, we look to see whether the anomaly score increases around the anomaly.

Impulse

The first test analyzes a time series containing an impulse (Figure 6.1, top). The series is 200 samples in length, with a 4-sample wide impulse appearing in samples 98-101. This series contains no noise.

I search for 2 profiles in this series. I used `build-profile` to built the first profile. The profile is built from a series that is 26 samples in length and contains the same impulse as the series of interest at samples 12-15. I chose a length of 26 because it is the smallest length that is still as long as the inspection window. I want the series to contain as little normal behavior as possible. I used a modified version of configuration 1 that counts subwords of lengths 1-6 to generate the profile.

The second profile I built manually. It contains the exact subword count histogram from the inspection window starting at sample 84 in the series of interest. The distance from the inspection window to this profile should be 0 when the inspection window reaches sample 84.

I analyze this series using five configurations (Table 6.1). For all configurations, I set the inspection-window length to the same value as the lead length.

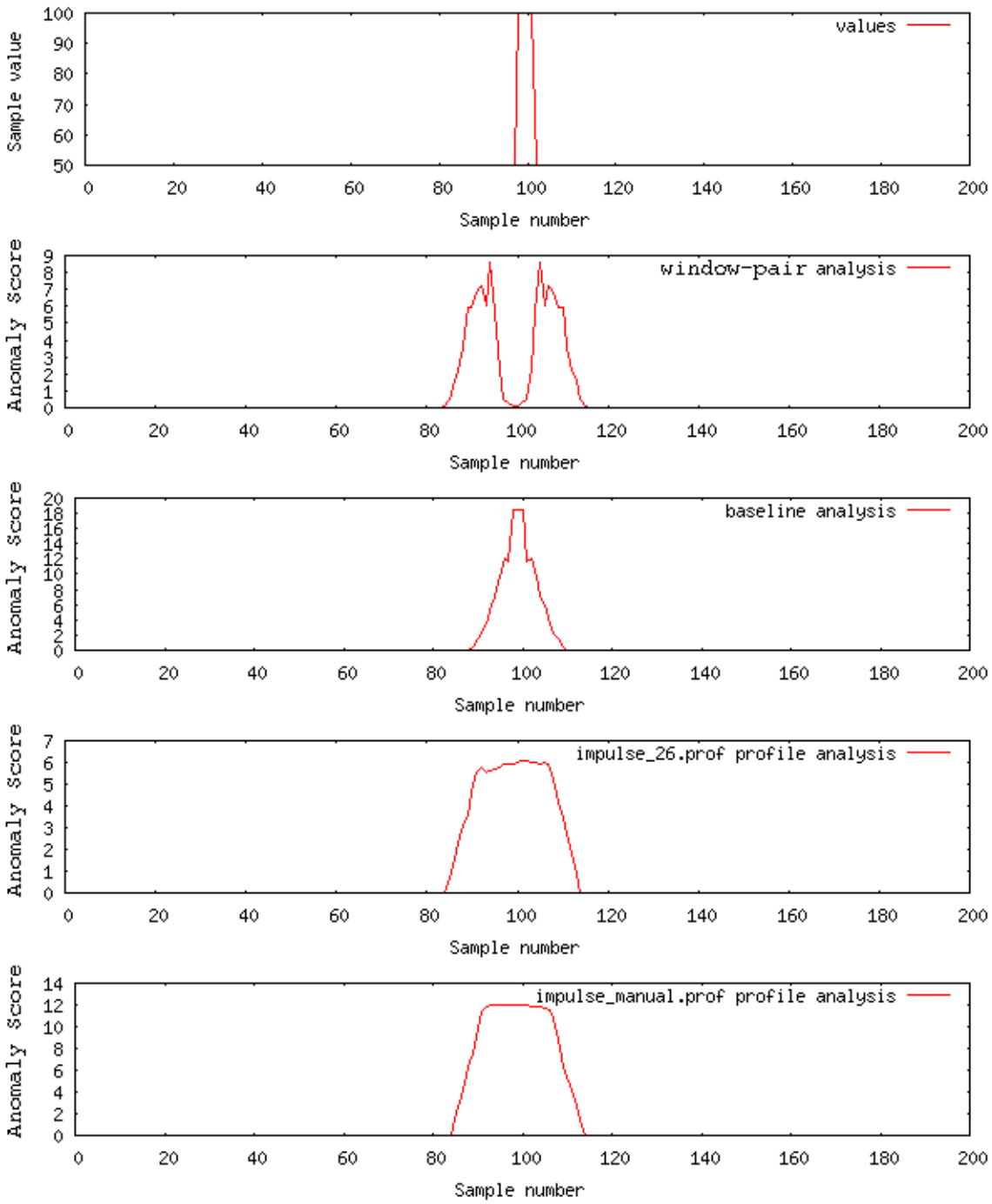

Figure 6.1: Configuration 1, impulse test.

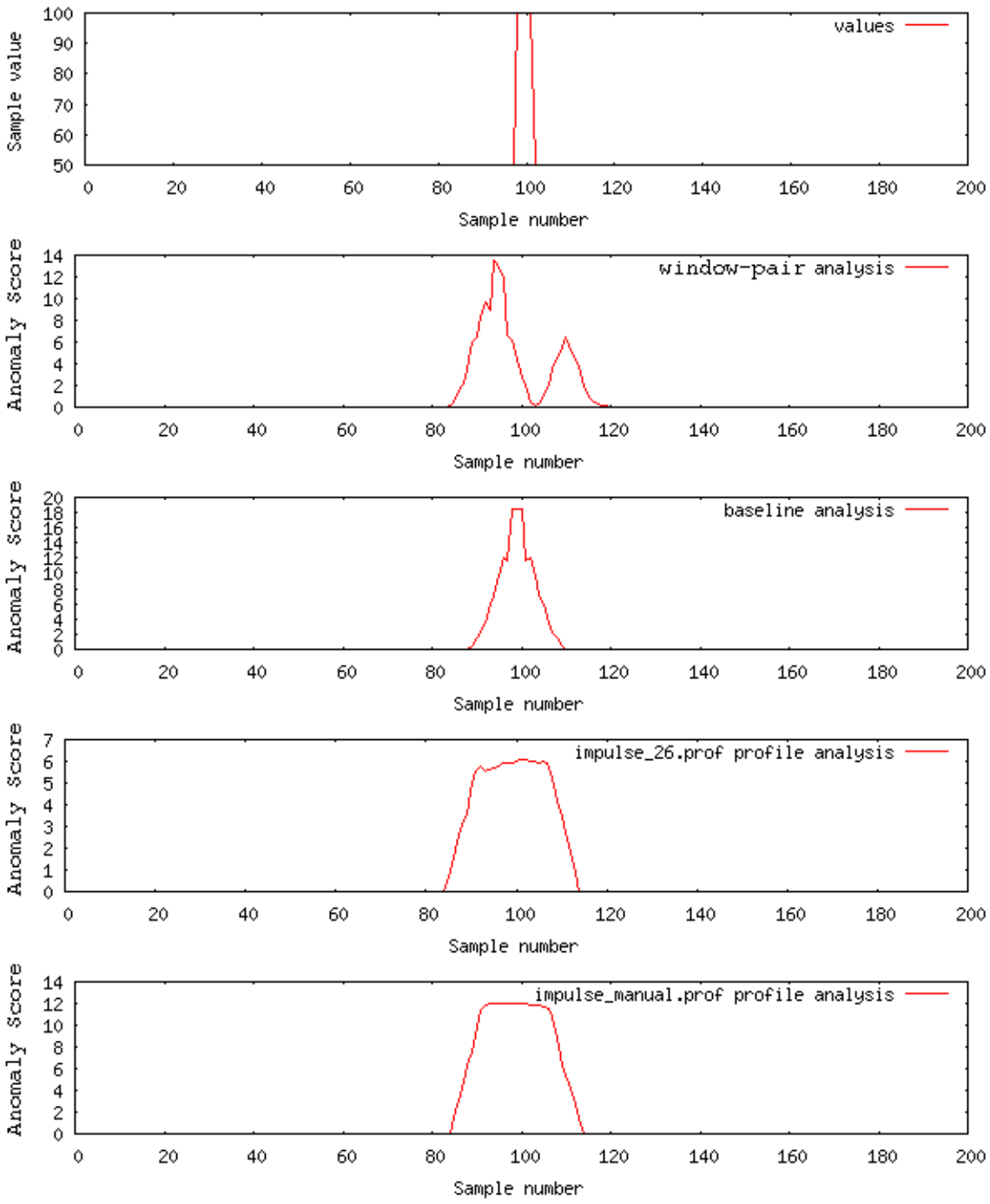

Figure 6.2: Configuration 2, impulse test.

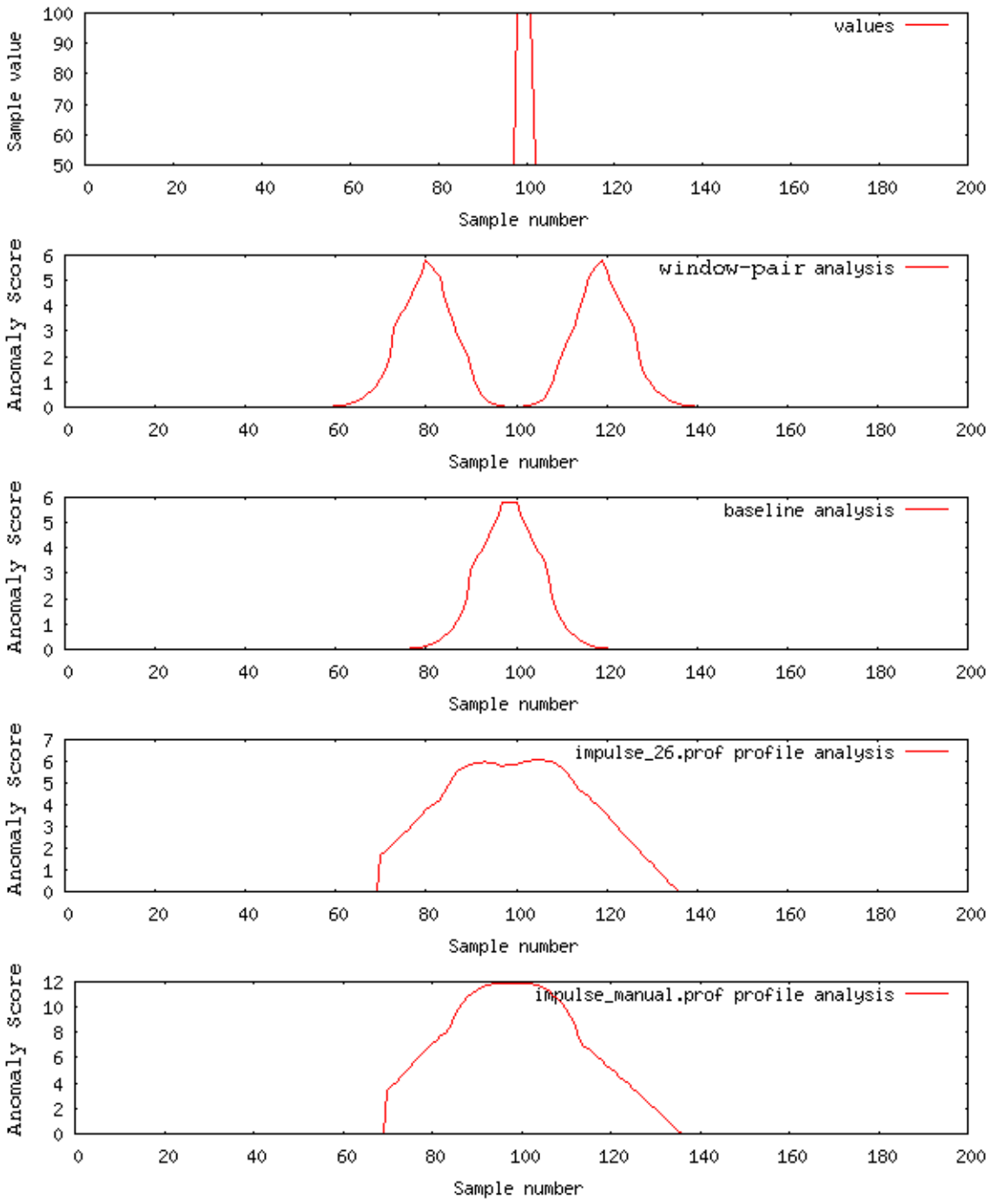

Figure 6.3: Configuration 3, impulse test.

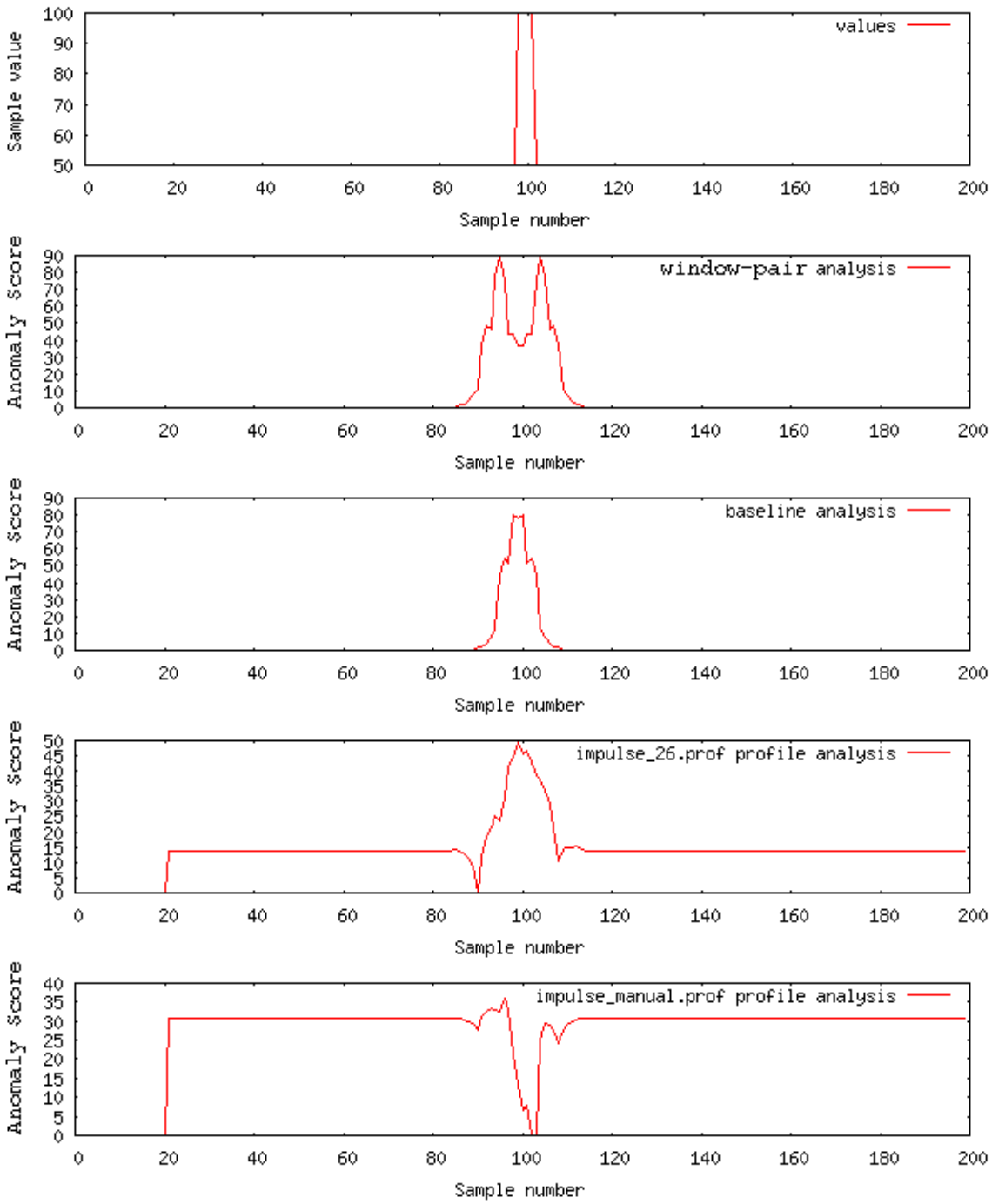

Figure 6.4: Configuration 4, impulse test.

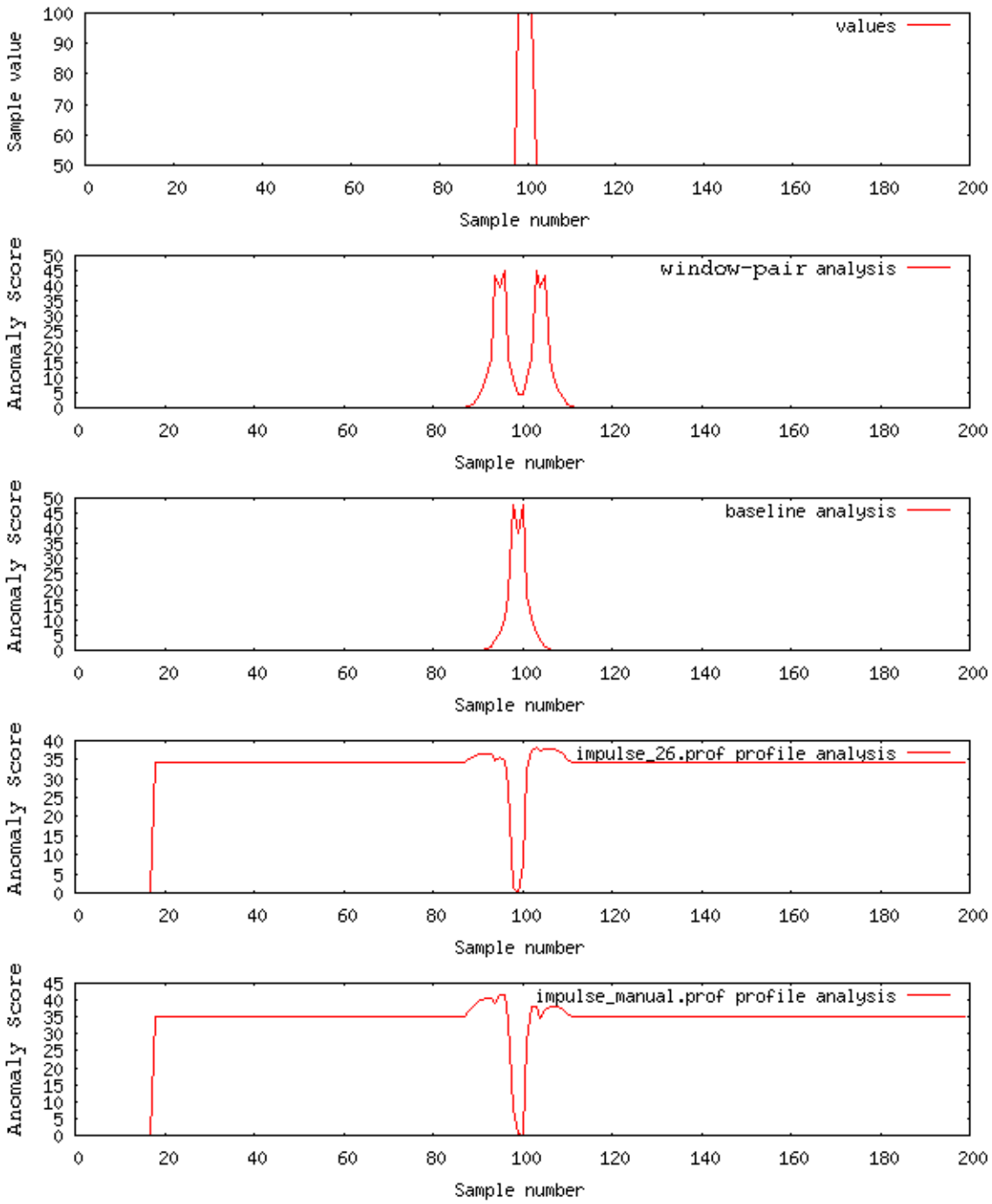

Figure 6.5: Configuration 5, impulse test.

Table 6.1: Standard testing configurations

Configuration number	1	2	3	4	5
Samples per symbol	3	3	6	3	3
Word length (symbols)	6	6	6	6	4
Subword lengths (symbols)	2,4	2,4	2,4	4,6	2,4
Lead window length (samples)	9	9	36	9	9
Lag window length (samples)	9	18	36	9	9

WP analysis and baseline analysis and both profile searches produce a high anomaly score surrounding the impulse. For tests where the lead and lag window have the same length, WP analysis produces a symmetric double peak surrounding the impulse. WP analysis produces this double peak because the score reported is the distance between the lead and lag windows. The distance between the lead and lag windows increases as the impulse enters the lead window. The distance falls when the impulse transitions between the two windows, because both windows contain some anomalous behavior. The second peak appears when the lag window contains the entire impulse; the anomaly score is high because the lead window now contains only normal behavior. Testing with the second configuration demonstrates that increasing the length of the lag window dampens the second peak.

For configurations 1, 3, 4, and 5, the window of interest is narrow, causing sharp peaks in the anomaly scores produced by WP analysis and baseline analysis. Configuration 5 demonstrates the effect of using a smaller word size; the peaks in the anomaly scores are taller and sharper than in configuration 1. Configuration 3 increases the number of samples per symbol and doubles the width of the lead and lag windows. This change flattens the anomaly scores from all three methods because the impulse is always a smaller portion of the inspection window. When compared with the baseline profile, the contents of the inspection window never appear as anomalous as in the other configurations.

The two profile searches produce similar anomaly-score plots for all configurations except configuration 4. The anomaly-score plot for the manually built profile behaves differently because it does not contain subword counts for subwords of length 6; configuration 4 is the only configuration that counts subwords of length 6. When the inspection window does not contain the impulse, the only word in the window is "cccccc". In this region, the distance between the two histograms for subwords of length 6 is 1 because "cccccc" is the only subword in the inspection window histogram and the profile histogram contains no subwords of length 6.

When the impulse enters the inspection window, we first see the anomaly-score rise. This rise occurs because the histograms for subwords of length 4 begin to look similar. However, the increasing number of different subwords with length 6 quickly overcomes the similarity between the length-4 histograms.

Neither profile search generates a peak for configuration 5. I generated the two profiles from a word length of 6 symbols, but configuration 5 generates words con-

taining only 4 symbols. Different word lengths generate very different groups of symbols. I counted the number of unique words generated from configuration 5. The list of words contains only twelve 4-symbol words. The 26-sample profile contains occurrences of eleven 4-symbol subwords, but has only two 4-symbol words in common with the list generated from configuration 5. Thus, the inspection window is most similar to the 26-sample profile when it contains the fewest subwords, that is, when the inspection window does not contain the impulse. When the impulse enters the inspection window, the distance between the inspection window and the 26-sample profile actually increases. The manually built profile suffers from the same problem.

The profile searches produce the best results for configurations 1 and 2. The peaks in the anomaly-score plots are not as sharp as for baseline analysis or WP analysis, but the anomaly scores away from the impulse are 0, and the peaks are still obvious. These results are unsurprising because configurations 1 and 2 are the most similar to the configurations that I generated these profiles from.

Configurations 1, 2 and 4 show small plateaus on either side of the peak in the anomaly-score plots for baseline analysis. The scores that form the leftmost plateau for configuration 1 are associated with samples 96 through 98. Scores for samples to the left of 96 consistently increase, and scores to the right of 98 remain the same until the other side of the peak. The score for sample 97 causes the plateau.

The plateaus are an result of the way that subwords move in and out of the inspection window as the window crosses the impulse. Before the inspection-window view contains the impulse, SAX conversion generates only the word ”cccccc”. The series is single-valued at that point, so the average for any subregion of the series is 0. SAX always assigns the symbol ”c” to subregions with an average value of 0. [1]

As the impulse comes into the view of the inspection window, the inspection window begins to contain higher numbers of words that begin with consecutive instances of the symbol ”b”. As a result, subwords beginning with consecutive instances of ”b” have high counts in the subword histogram. At the same time, the histogram contains fewer and fewer subwords with consecutive instances of ”c”, so the anomaly score rises.

When the impulse reaches the halfway point in the inspection-window view, the number of subwords that begin with consecutive instances of ”b” begins to decrease; they are replaced with words that *end* with consecutive instances of ”b”. As the words beginning with ”b” exit the inspection window, the counts for subwords starting with ”b” do not decrease uniformly. In particular, the subword ”bbbd” continues to have the same unnormalized count. The other subwords starting with ”b” now have lower counts, and thus contribute significantly less to the anomaly score.

[1] The series used to build the baseline profile for this test is also single-valued. The subword histogram thus contains entries for only the subwords that contain exclusively the symbol ”c”.

The new subwords that end in ”b” also have low unnormalized counts. The combined effect of the movement of these subwords is that the anomaly score decreases for one sample. When the inspection window moves to the next sample, the count for the subword ”bbbd” decreases, allowing the other subwords to again contribute significantly to the anomaly score.

Impulse with noise

The second test evaluates a series similar to the first, except that I have added some noise to the series. The noise is added from a normal distribution with a standard deviation of 5. The series is still 200 samples long with a 4-sample wide impulse in samples 98-101.

Table 6.2: Testing configuration number 6

Configuration number	6
Samples per symbol	6
Word length (symbols)	4
Subword length (symbols)	2,4
Lead window length (samples)	9
Lag window length (samples)	9

In addition to the 5 configurations used in the previous tests, I test this series with a 6th configuration (Table 6.2). This configuration is the same as configuration 5 except that the number of samples per symbol is doubled. This change should reduce the effect of the noise on the generation of symbols.

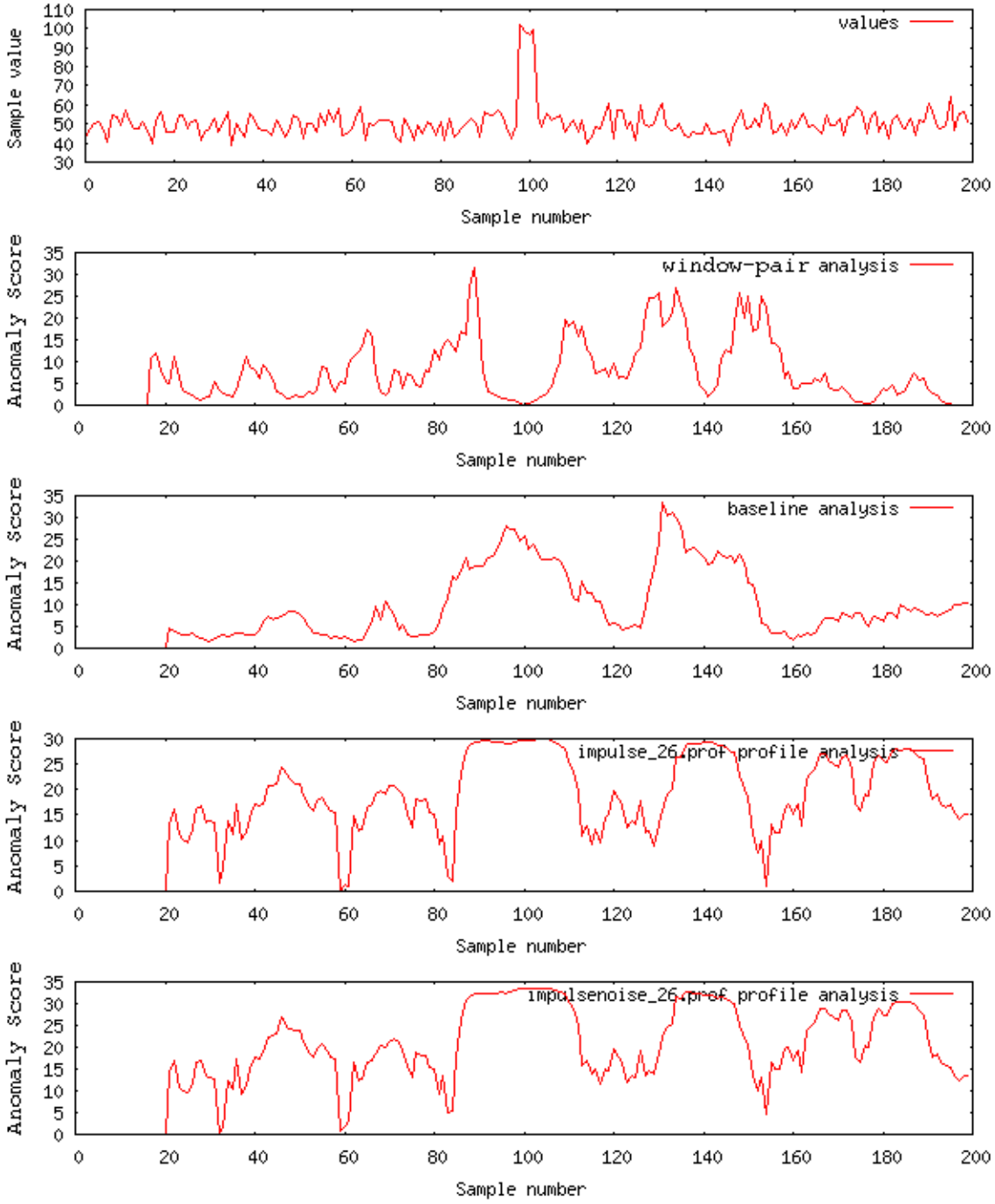

Figure 6.6: Configuration 1, noisy impulse test

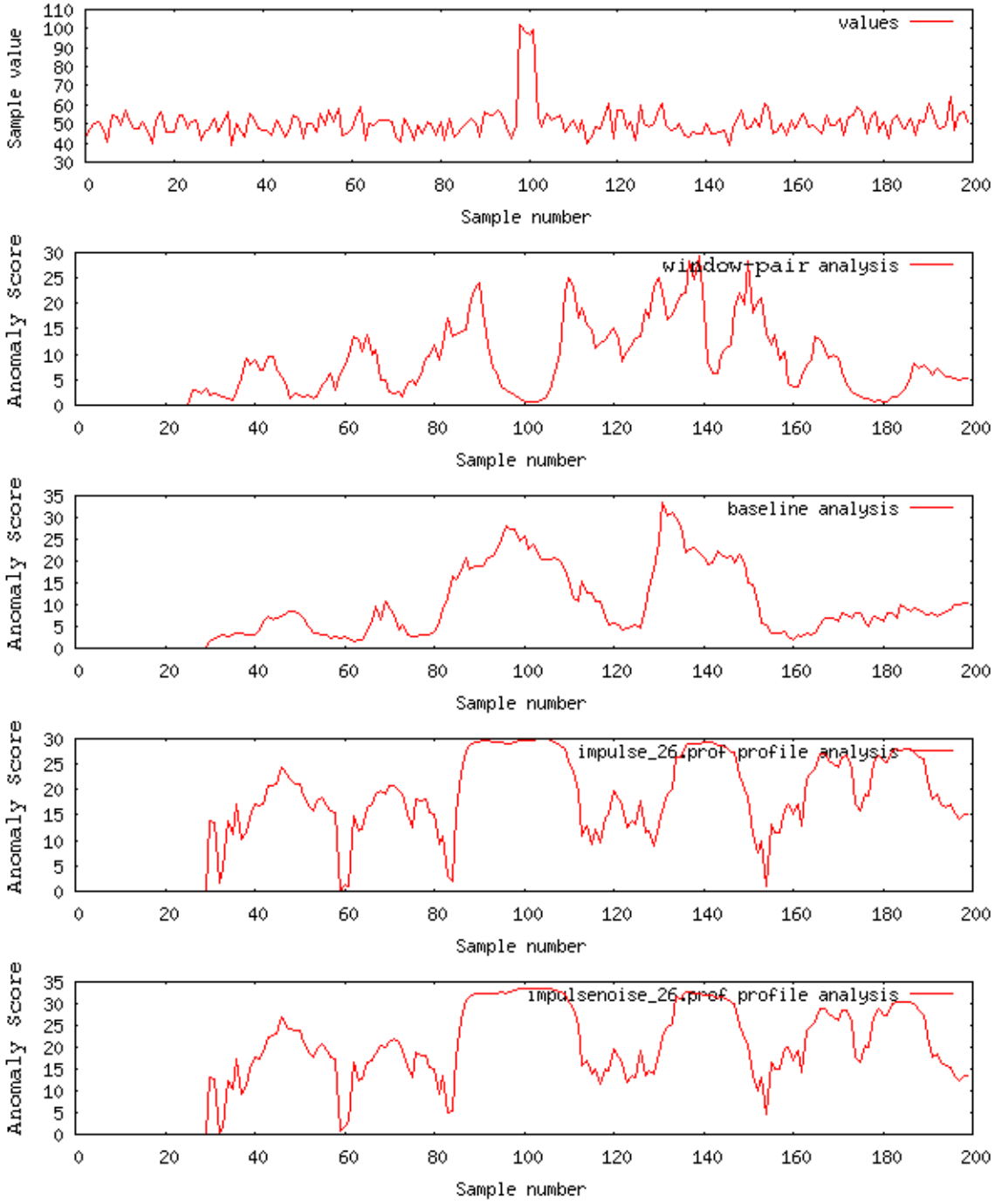

Figure 6.7: Configuration 2, noisy impulse test

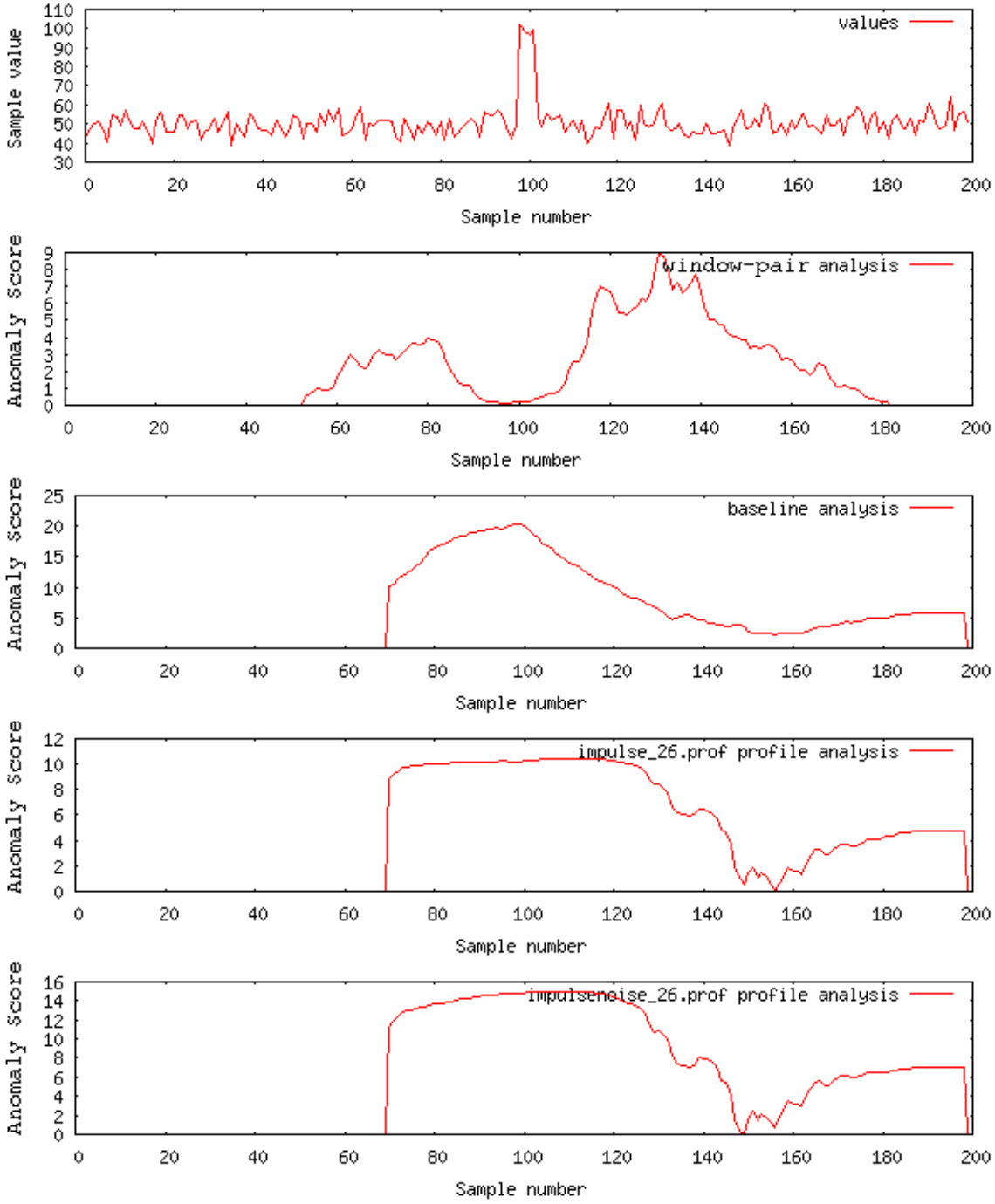

Figure 6.8: Configuration 3, noisy impulse test

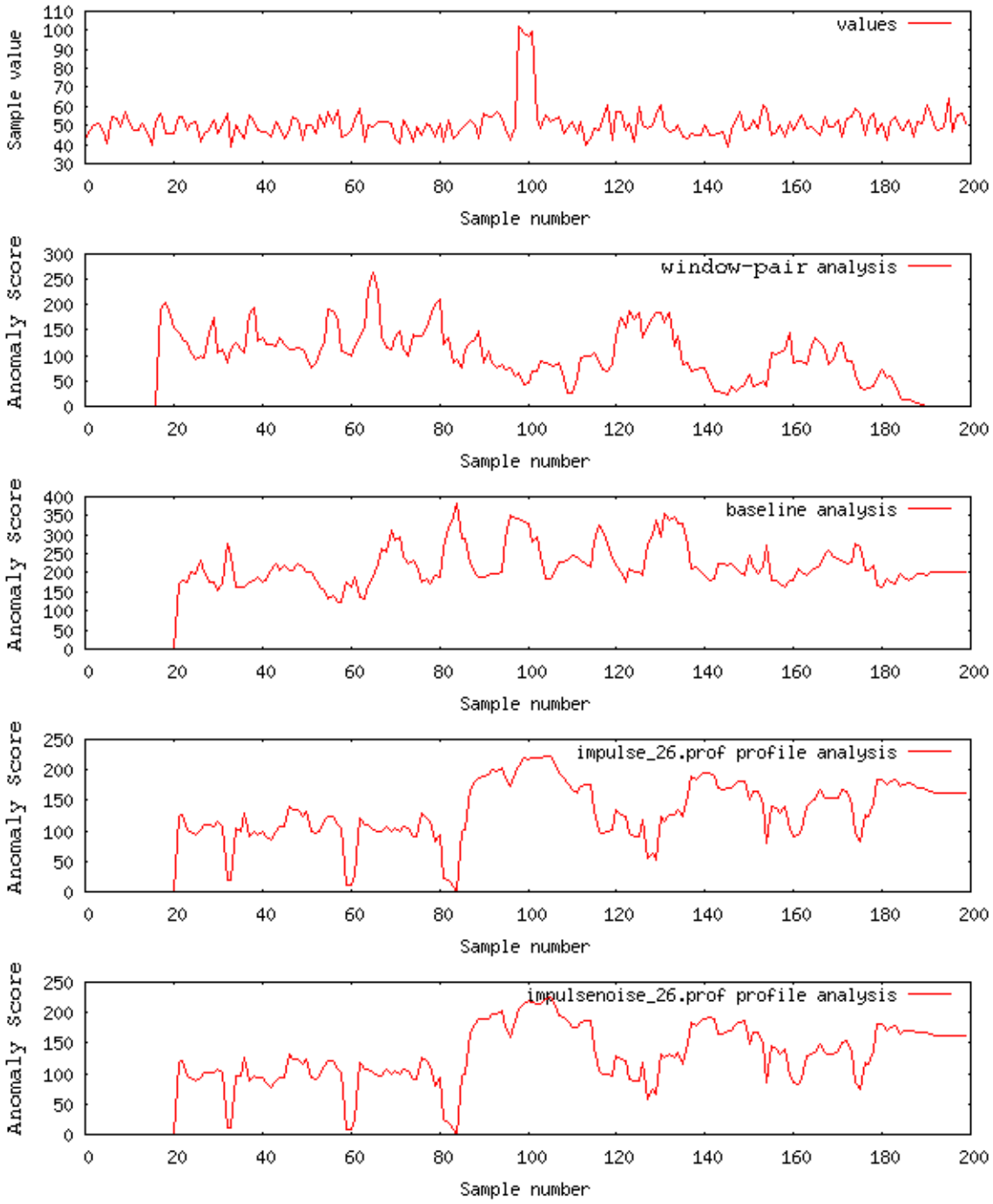

Figure 6.9: Configuration 4, noisy impulse test

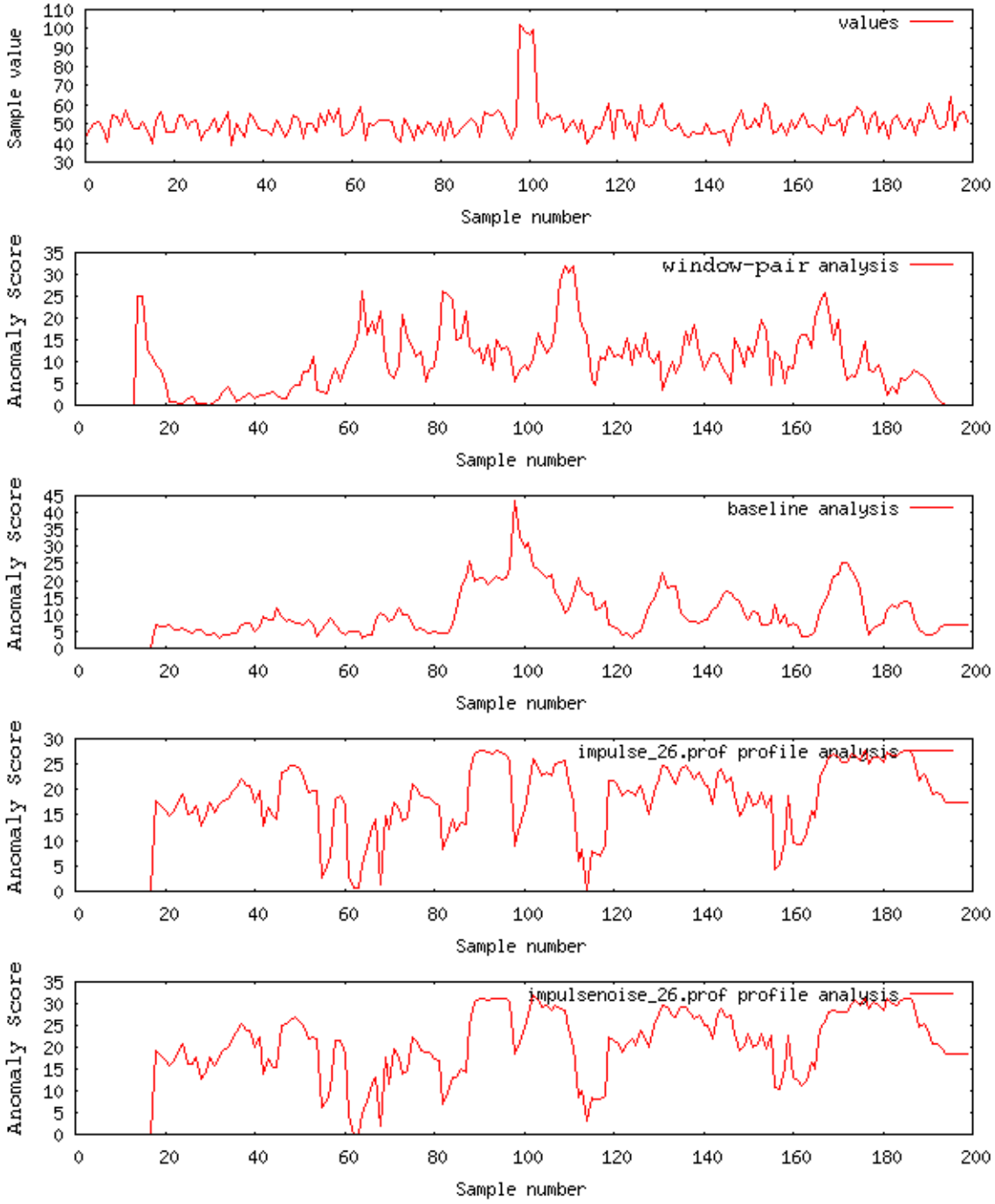

Figure 6.10: Configuration 5, noisy impulse test

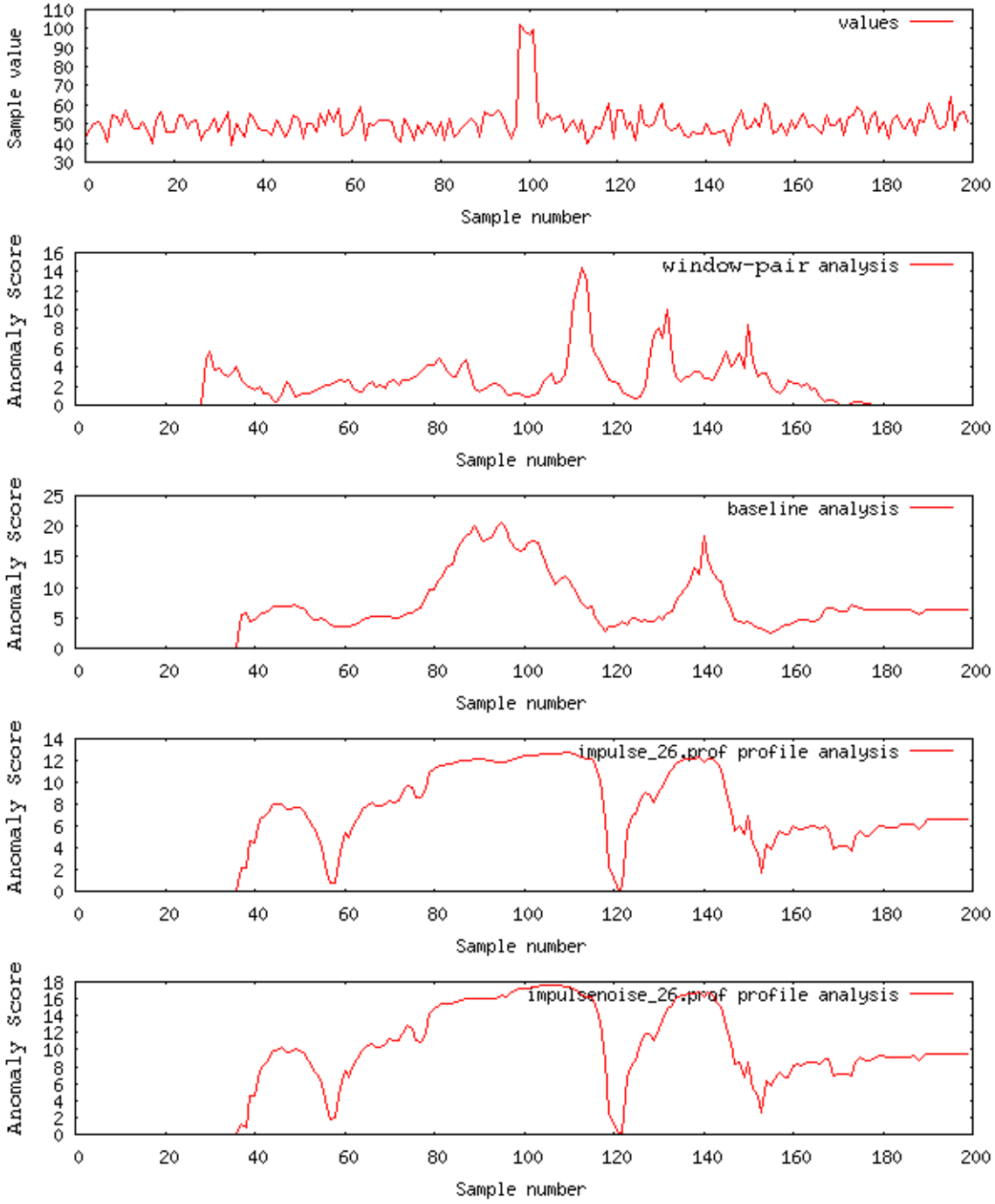

Figure 6.11: Configuration 6, noisy impulse test

The WP anomaly-score plots for configurations 1 and 2 show a clear double peak around the anomaly (Figure 6.6, 6.7). These two configurations also show a second double peak centered at approximately sample 141. Sample 141 has a value of 50, while the average of the ten surrounding samples is approximately 45. The value 50 is anomalous in this limited context, but it does not constitute abnormal behavior. This value stands out only because the surrounding samples consistently have the value of approximately 45.

Configuration 6 shows a clear double peak around sample 126. Configurations 1, 2, and 3 also have double peak in this region, but they are disguised by the peaks that surround the anomalies that occur at samples 100 and 141. The peak to the left of sample 126 merges with the peak to the right of sample 100, and the peak to the right of sample 126 merges with the peak to the left of sample 141. The values of the samples surrounding sample 126 vary more than in the surrounding area, so this double peak is not surprising.

The WP anomaly-score plots for configurations 4 and 5 do not show any clear anomalies. I do not find this behavior surprising in either case. Configuration 4 counts subwords of lengths 4 and 6. Other configurations show that counting subwords of length-4 works well but in this case the anomaly-score plot from the length-4 subwords is overwhelmed by the anomaly-score plot from length-6 subwords.

Counting subwords of length 6 does not work well for two reasons. First, the number of possible subwords of length 6 is significantly larger than for subwords of length 4. There are likely to be more different subwords of length 6. Second, the word length for configuration 4 is 6 symbols; by counting subwords of length 6, configuration 4 counts occurrences of full words. In general, adjacent words are very unlikely to have the same value. Thus, in addition to subwords of length 6 having a larger possible variety, subwords of length 6 are also very unlikely to appear near each other under this configuration. Configuration 5 suffers from the same problem; it runs the analysis with a word length of 4 symbols and counts subwords of lengths 2 and 4.

All configurations except configuration 4 provide good results from baseline analysis. All configurations find the artificial anomaly in samples 98 through 101, although configuration 4 also shows a high anomaly score for many other samples, most of which are not anomalous. All configurations except configuration 4 show a high anomaly score around the anomaly in sample 141. Configuration 5 is the best; it shows a sharp peak in the anomaly score right at the beginning of the impulse. As expected, the longer inspection window and larger symbol size in configuration 3 dramatically smooth the anomaly-score plot. Configuration 6 strikes a better balance; the curve is smooth but the peaks are still sharp.

Baseline analysis does a good job of picking the impulse out of the surrounding noise, but there is no simple way to automatically detect whether any part of the series is anomalous. The anomaly score vectors vary significantly in range and are only useful for telling which part of the series is most anomalous.

I search for two profiles in this series. The first is the same 26-sample profile that I searched for in the previous series. The second profile I built from a 26-sample

series that contains a 4-sample wide impulse and noise from a normal distribution. The anomaly-score plots from these searches do not provide much information about whether the series contains the corresponding anomalies. The plots from configuration 3 have maximum values surrounding the anomaly, but the peaks are too wide to provide useful information about where the anomaly is.

The plots from the profile searches for configurations 1, 2, and 6 detect both the impulse and the anomaly at sample 141. These anomalies cause the highest peaks in these plots. The peak surrounding the impulse is broader; aside from this difference, the peaks are mostly the same for the two anomalies.

Sine function

This test applies WP and baseline analysis to a time series representing a sine function that doubles in frequency 700 samples into the series. The series is 1200 samples long. For the first 700 samples, the sine function has a period of 100 samples. Beginning at sample 700, the period changes to 50 samples.

I present three new configurations for testing this series (Table 6.3). These configurations all use the same number of samples per symbol and the same word length. The inspection window in configuration 7 covers exactly one of the longer periods and the lag window is twice the length of the inspection window. The inspection window in configuration 8 is exactly long enough to cover one shorter period. The inspection window in configuration 9 is not a whole number of periods in length.

I test this series with two baseline profiles. The first profile I built from the first 200 samples of this series; this baseline represents two of the longer periods. I expect this long-period baseline to produce a low anomaly score for the first 700 samples and to produce a high anomaly score for the remainder of the series. The second profile I built from the 200-sample section beginning at sample 700 and continuing to sample 899; this baseline represents four of the shorter periods. I expect this short-period baseline to produce a high anomaly score for the first 700 samples and a low anomaly score after that. I used configuration 7 to build both profiles.

I first test configurations 7 and 8 with the long-period baseline (Figures 6.12 and 6.13). I then test configurations 7 and 8 with the short-period baseline(Figures 6.14 and 6.15). Finally, I test configuration 9 with the long-period baseline (Figure 6.16).

Table 6.3: Sine function testing configurations

Configuration number	7	8	9
Samples per symbol	5	5	5
Word length (symbols)	5	5	5
Subword lengths (symbols)	2,4	2,4	2,4
Lead window length (samples)	100	50	75
Lag window length (samples)	200	100	150

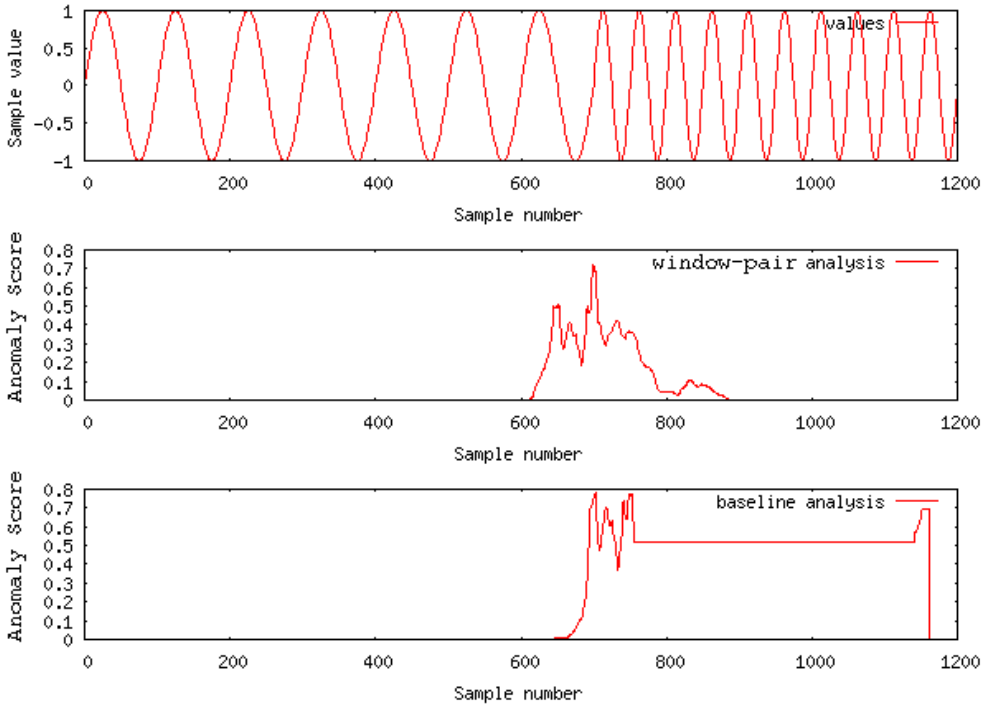

Figure 6.12: Configuration 7, long-period baseline, sine function test

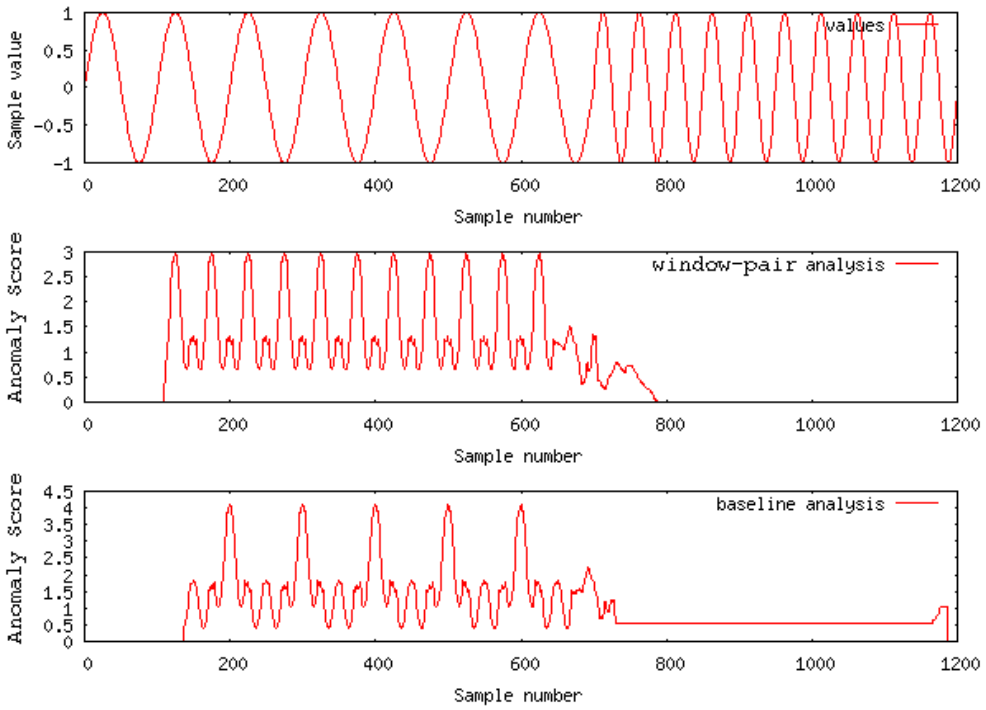

Figure 6.13: Configuration 8, long-period baseline, sine function test

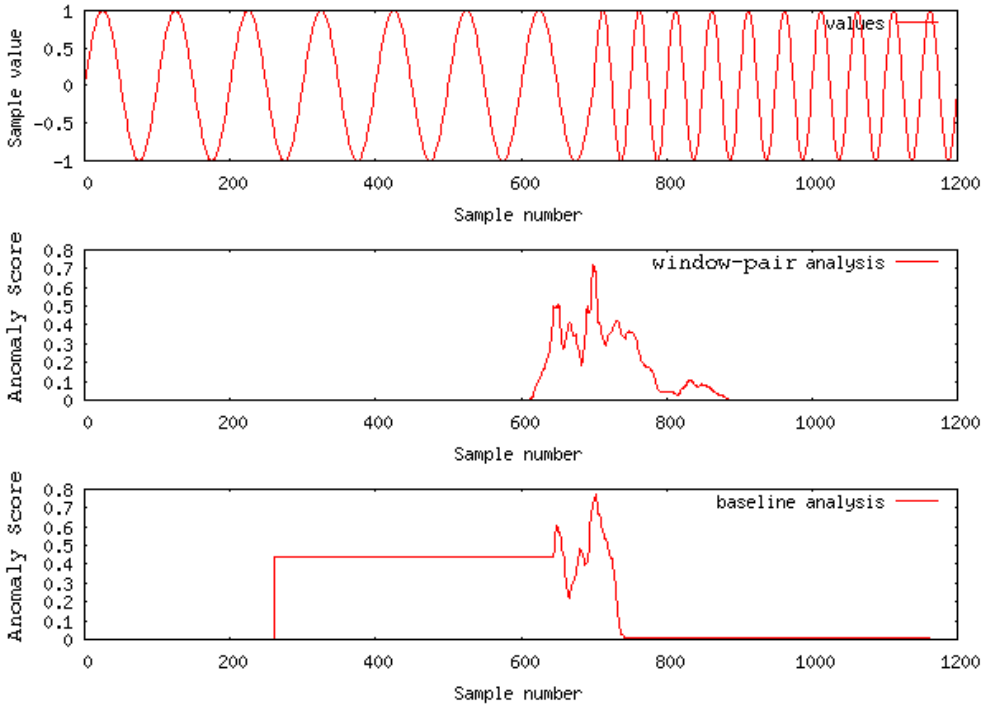

Figure 6.14: Configuration 7, short-period baseline, sine function test

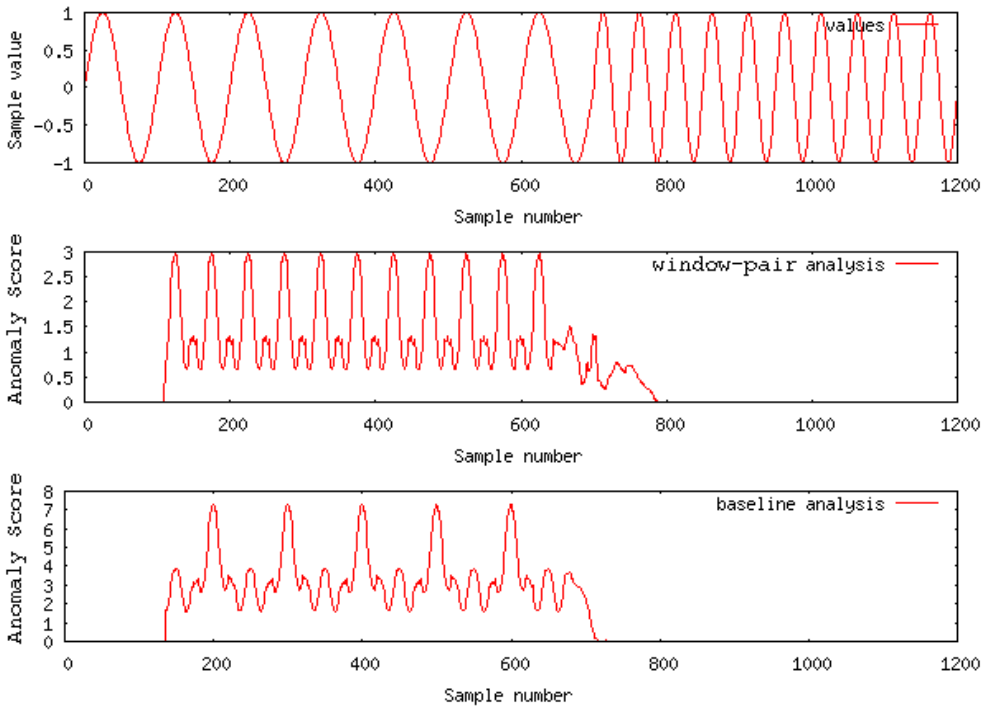

Figure 6.15: Configuration 8, short-period baseline, sine function test

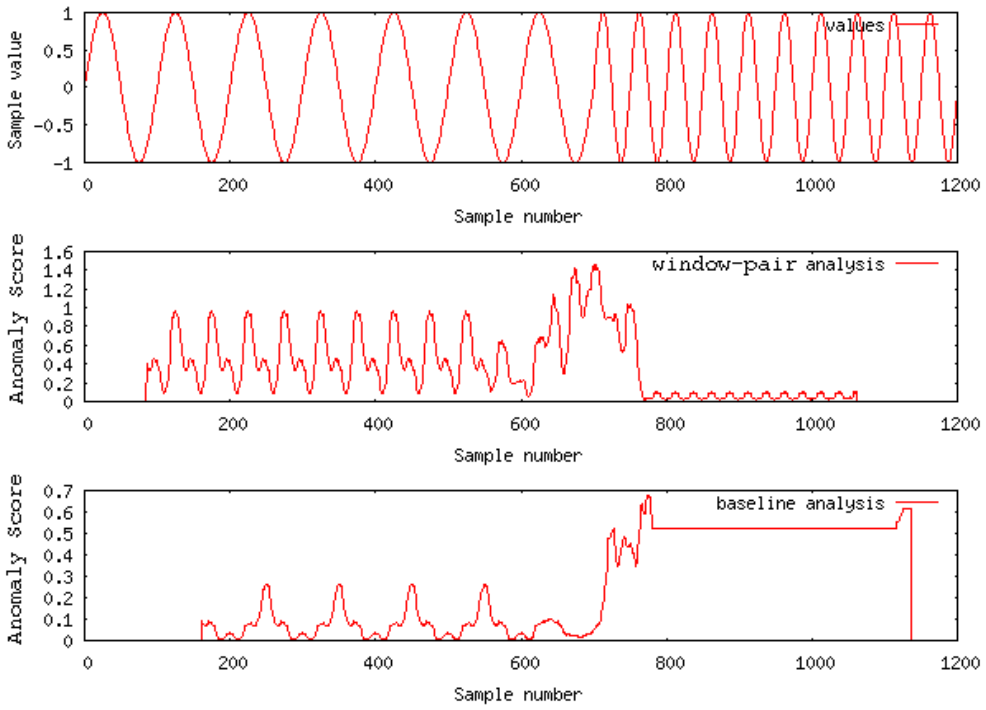

Figure 6.16: Configuration 9, long-period baseline, sine function test

Both WP and baseline perform well in the first test (Figure 6.12). The WP anomaly-score vector has the value 0 everywhere except for around the change in frequency. The maximum value in the WP anomaly-score vector is 0.718 and occurs at samples 698 and 699. WP analysis does not produce a double peak for this type of anomaly; it instead produces a single peak at the location of the anomaly. This result makes sense: the lead and lag windows are most different when the lead window contains only the short-period sine function and the lag window contains only the long-period sine function. This situation occurs when the border between the windows lies at sample 700.

Baseline analysis produces an anomaly score of almost 0 until around sample 650; this location corresponds to when the leading edge of the inspection window reaches the change in frequency at sample 700. The anomaly score rises quickly to a peak at sample 703; at this point the inspection window contains one entire period of the higher frequency waveform. Regular changes in the slope of the waveform as it enters the inspection window cause the anomaly score to vary between samples 703 and 750. The anomaly score reaches a second peak at sample 750 that corresponds to the point at which the lag window contains only the short-period waveform. The anomaly score remains constant until sample 1139, at which point the shorter words at the end of the series begin to enter the inspection window, causing the anomaly score to rise again.

In test 2 (Figure 6.13) I employ a shorter inspection window that contains only one half-period of the long-period waveform. Although I still use the baseline profile built from the long-period waveform, the anomaly score is higher in the region of the long-period waveform than in the region of the short-period waveform. This characteristic also applies to WP analysis for this test.

Configuration 8 never allows the lead and lag windows to share the same profile in the first 700 samples. The lag window always contains a single period from the long-period waveform and the lead window always contains one half-period of the long-period waveform. The WP anomaly score is highest when the lead window contains samples that have a monotonic slope. When this happens, the words in the lead window are either all decreasing or all increasing, that is, they all follow the regular expression $d^*c^*b^*a^*$ (decreasing) or the regular expression $a^*b^*c^*d^*$ (increasing). The lag window contains approximately the same number of each kind of word.

The peaks in the baseline analysis anomaly score occur for the same reason, but baseline analysis offsets the anomaly scores so that they are associated with the center of the inspection window. Thus, the peaks appear at the beginning of each period instead of at the peaks and troughs of the waveform. The anomaly scores for the second portion of the test series are lower because the inspection window contains an entire period of the short-period waveform. Although the words produced by the long-period waveform are different from those produced by the short-period waveform (the slope of the long-period waveform is less steep), the two sequences of words contain many of the same subwords.

The results from test 4 (Figure 6.15) resemble the results from this test. WP produces the same results because the SAX parameters are exactly the same. Baseline analysis produces a waveform similar to test 2 except that the anomaly score decreases to 0 once the inspection window contains only the short-period waveform.

The WP anomaly-score vector produced in test 3 (Figure 6.14)is the same as in test 1 because the SAX parameters did not change. I built the baseline profile used in this test from the short-period waveform. Baseline analysis does not record an anomaly score until sample 250; the inspection window is always in line with the lead window from WP analysis, and the lead window begins at sample 200 to account for the lag window. Again, we see two peaks in the baseline analysis anomaly-score plot. The first peak results from the frequency change entering the view of the inspection window for the first time. The second peak occurs at sample 700, when the inspection window contains one half-period of the long-period waveform and one whole-period of of the short-period waveform.

Test 5 demonstrates the behavior of WP and baseline analysis when the inspection window does not contain a whole number of periods. The normalized counts of the subwords contained in the lag window or baseline profile never exactly match the normalized counts in the inspection window, causing the anomaly score to rise and fall periodically. After sample 700, the inspection window contains exactly one and a half periods of the short-period waveform, causing the baseline analysis to behave in a way similar to the other tests.

6.2 Testing Multi-Property Detection

I use RAACD for testing my implementation of multi-property abnormal-behavior detection. I have a NodeScape v2 instance collecting health data from 81 hosts. I have built baseline profiles for all of the properties on 25 of these hosts. I build these baseline profiles using the RAACD-profile configuration (Table 6.4). RAACD uses the RAACD-search configuration (Table 6.5) to search for abnormal behavior. I chose a longer inspection window and a larger symbol size than configurations 1-6 to better fit the natural behavior of the machines that I am monitoring. Most of the machines send updates at 5-minute or 10-minute intervals. For machines that update at 10-minute intervals, the 12-sample inspection window covers on hour of behavior; for machines that update every 5 minutes, the inspection window covers 2 hours. I started using RAACD with the RAACD-profile configuration and have since adjusted the configuration to the one in Table 6.5. RAACD-search produces better results than RAACD-profile.

Table 6.4: RAACD-profile configuration

Configuration name	RAACD-profile
Samples per symbol	4
Word length (symbols)	6
Subword length (symbols)	1 .. 6
Lead window length (samples)	12
Lag window length (samples)	12

Table 6.5: RAACD-search configuration

Configuration name	RAACD-search
Samples per symbol	3
Word length (symbols)	5
Subword length (symbols)	2,4
Lead window length (samples)	15
Lag window length (samples)	15

I began testing this detection method by evaluating synthetic anomalies. We opened several sessions on the machine `violet.cs.uky.edu` (`violet`) and started a long-running program that causes a high but varying load on the machine. The graphs in Figure 6.17 show abnormal behavior in several properties beginning at approximately 11:00 pm UTC. Used memory, session count, process count, and load average all show increased values for the period between 11:00 pm and 5:00 am on the following day. Cache memory demonstrates tame behavior except for two impulses at 9:00 pm and 2:00 pm.

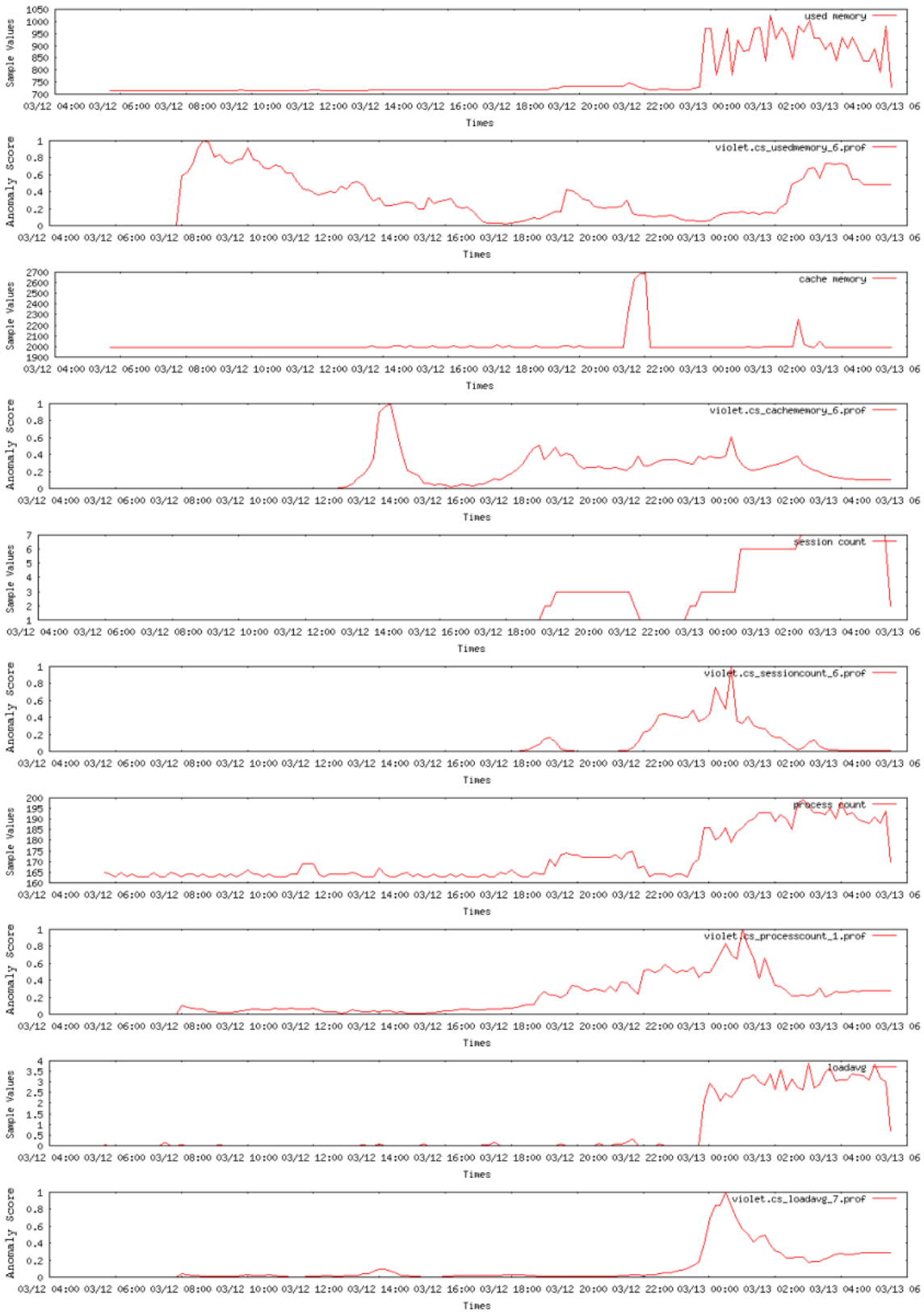

Figure 6.17: Monitoring data captured from `violet.cs.uky.edu`

The configuration used for this test considers behavior to be abnormal if three or more anomaly scores rise above 0.6 for the same sample. My algorithm considers `violet`'s behavior to be anomalous because the anomaly scores for load average, session count, and process count all rise above 0.6 at approximately 12:00 am. Although the anomaly scores for both used memory and cache memory rise above 0.6 in multiple places, they do not contribute to `violet`'s behavior being considered anomalous. Neither score rises above 0.6 at the same time as any other score.

The program fails to recognize the abnormal behavior in cache memory usage that occurs at 9:00 pm and at 2:00 pm. Even though two impulses, one of them rather large, appear in the monitoring data, this cache memory behavior, as a whole, is calmer than the baseline behavior for `violet`. The series used to build the baseline for `violet` actually contains an impulse similar to the one which occurs at 9:00 pm, which causes the impulse in the current series not to register as anomalous.

The anomaly detection program runs continually, generating an HTML document when it detects anomalous behavior. The program detected an anomaly on `iris.cs.uky.edu`; Figure 6.18 shows the corresponding sample series and anomaly scores. We see anomalous behavior between 2:00 pm and 9:00 pm. We see abnormal behavior primarily in process count, session count, and memory usage, and we see a small increase in memory being used as cache. Inspection of the anomaly scores shows that the scores for process count, session count, and memory usage all exceed the 0.6 threshold at 6:30 pm. In particular, the algorithm detects the relatively small change in behavior that occurs at 7:00 pm.

In this case, the algorithm detects abnormal behavior, but it does not detect it early enough to be useful. We would prefer to detect the change in behavior that happens shortly after 2:00 pm.

There is also some abnormal behavior occurring around 12:00 pm. We observe a short spike in load average, cached memory, and used memory. This behavior is not significant enough in magnitude or duration to trigger high anomaly scores, nor is it sufficiently abnormal to be of interest to the system administrator. The detection algorithm correctly ignores this spike. Figure 6.19 shows recent behavior from the machine labeled `conglomerate.kaos`. `Conglomerate` runs the abnormality detection algorithm once every ten minutes, generating a cyclic load. This cycle manifests itself mostly in process count, load average, and CPU1 temperature. Even though this behavior is normal (we expect the program to generate load every 10 minutes), the anomaly scores for process count and load average remain consistently high. The reason that these scores increase is that we built baseline profiles for `conglomerate` before the detection program was scheduled to run every 10 minutes. The previous normal behavior (Figure 6.20) is different enough from the new normal to look anomalous. I have built a new baseline profile for process count on `conglomerate`, but the anomaly-score plot for process count still shows consistently high scores. The frequency of the cycle in process count does not survive SAX conversion, but it does affect the list of words enough to make it difficult for any profile to match.

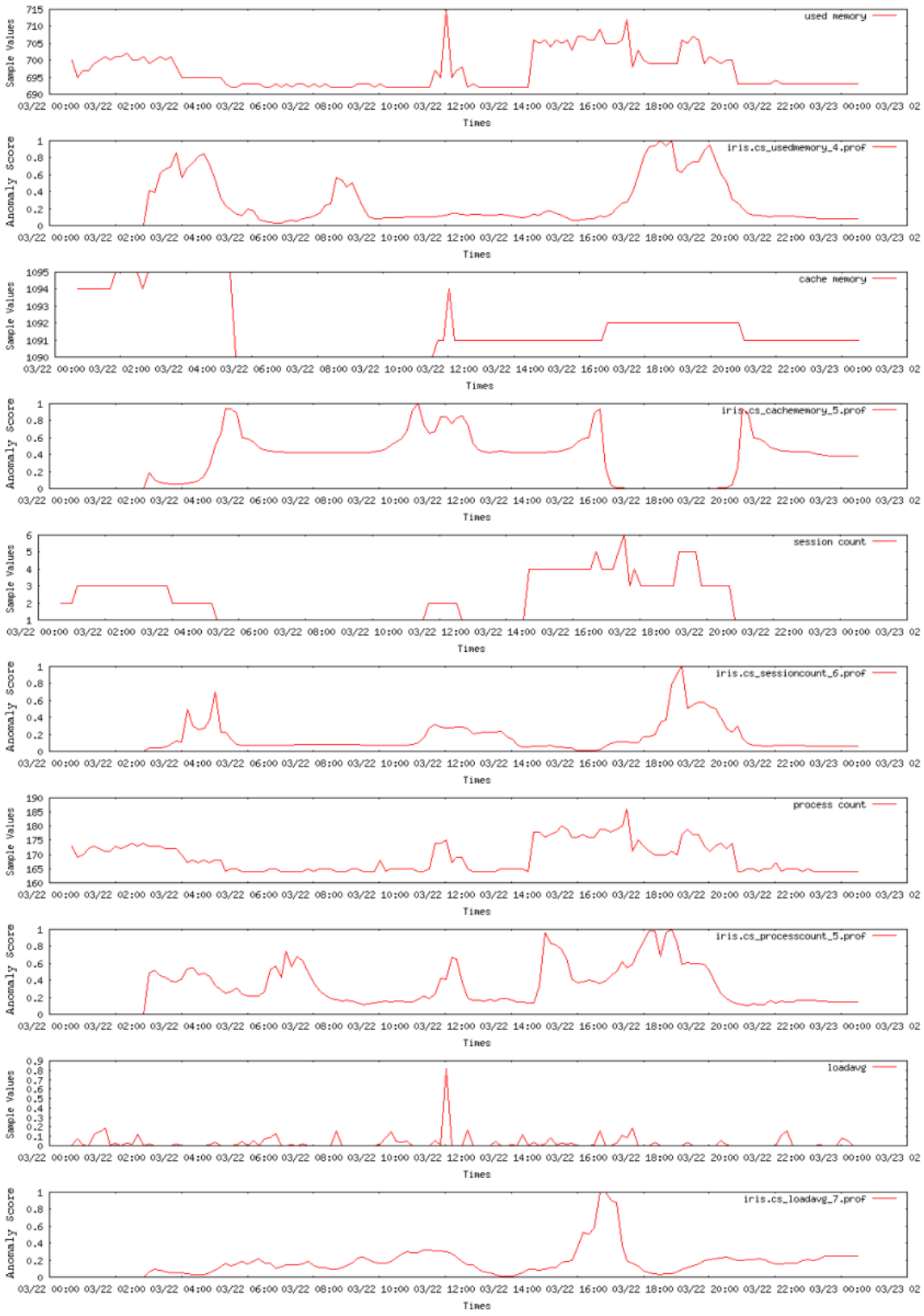

Figure 6.18: Monitoring data captured from `iris.cs.uky.edu`

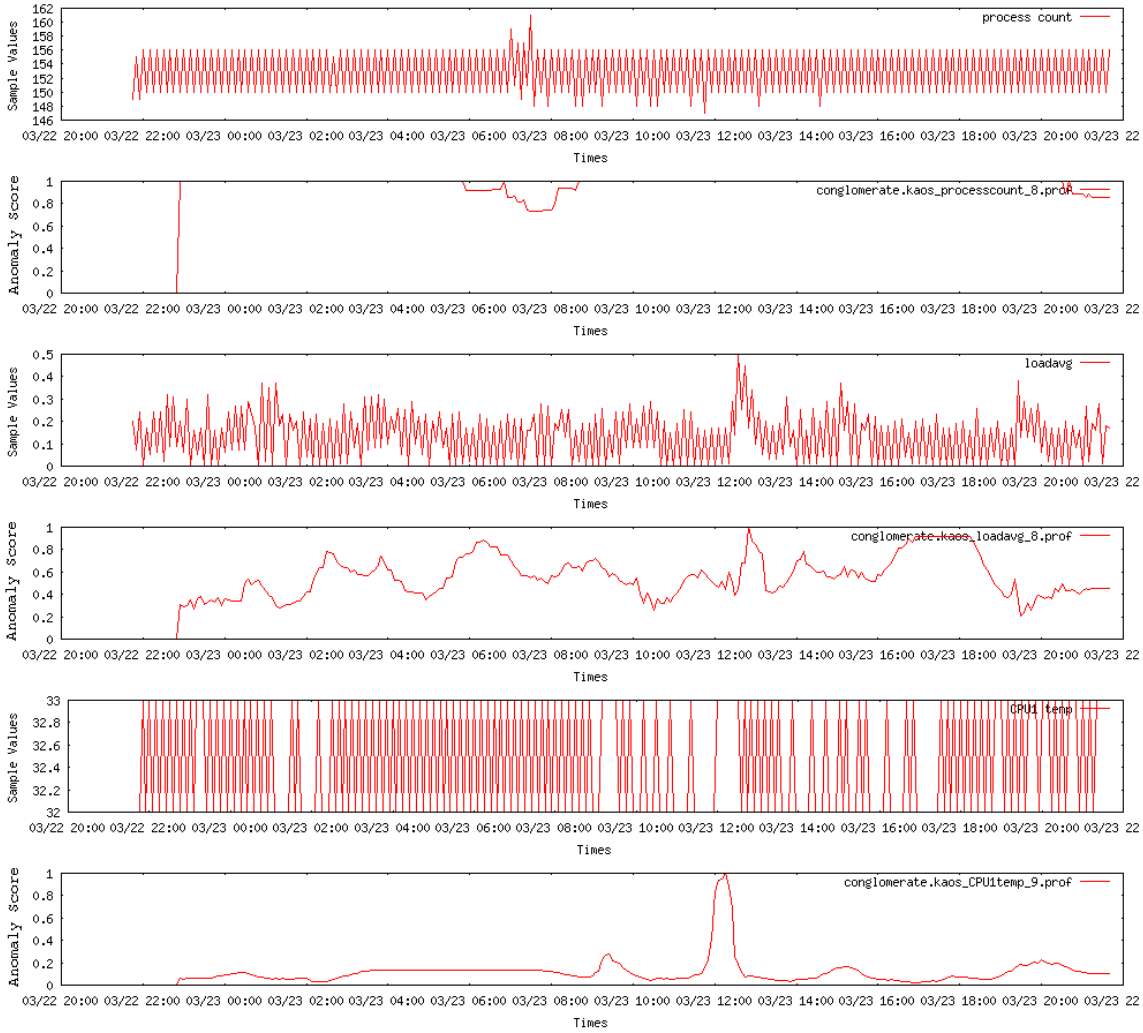

Figure 6.19: Monitoring data captured from `conglomerate`

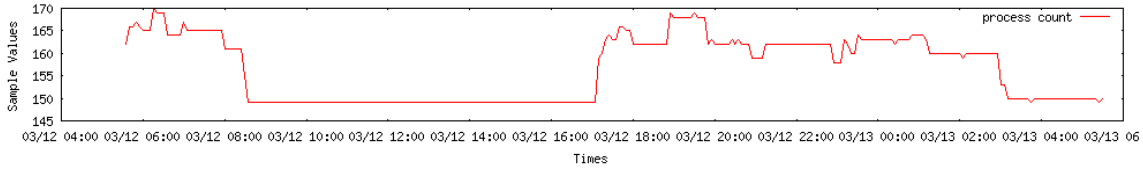

Figure 6.20: The old process count on `conglomerate`

Chapter 7 Conclusion

I find that WP analysis, although it performs well on simple synthesized time series, does not work well when applied to real data. Performing WP analysis on real data with symmetric window sizes produces an anomaly-score vector that is consistently noisy and high valued. WP analysis produces a better anomaly-score vector when the window lengths are asymmetric, but the score still does not rise sharply around anomalies. WP analysis also suffers from the lack of an absolute baseline reference.

A profile search yields good results under ideal circumstances. When I search for a specific anomaly in a clean synthesized time series, the resulting anomaly-score shows a peak surrounding the anomaly. Unfortunately, my method for profile search is not suitable for general use. It is sensitive to differences between the configuration used to build the profile and the configuration used to perform the search. In particular, if the search uses a different word length or counts subwords of a length not include in the profile, the resulting anomaly-score vector contains no useful information about were or whether the anomaly of interest occurred. In the cases where a profile search does produce useful information, the information provided by baseline analysis and WP analysis is much sharper.

Baseline analysis performs better than either WP analysis or a profile search. Baseline analysis produces an anomaly-score vector that rises sharply around anomalies in the time series. Baseline analysis works particularly well in the context of computing systems. We have a good concept of what constitutes normal behavior for computing systems, and we can easily build profiles to represent normal behavior.

My tests with the sine function demonstrate that WP and baseline analysis are both able to detect anomalous behavior that isn't a dramatic change in magnitude. Both of these methods can detect a change in frequency when configured correctly. These methods best detect such changes when the inspection window contains one entire normal cycle of behavior.

None of the three basic methods for anomaly detection is useful for automatic detection of anomalous or abnormal behavior. The anomaly-score vectors produced by these three methods vary wildly in magnitude from one series to the next. I was not able to develop a rule to tell when an anomaly score is high enough to signify an anomaly. The distance vectors produced by these three methods are only useful to show which parts of a time series are most anomalous.

Although these methods by themselves are not enough to automatically detect abnormal behavior, I was able to develop an algorithm based on these methods that does detect abnormal behavior. My method for multi-property search successfully detects abnormal behavior in a production environment. I demonstrate that by combining the anomaly-score vectors from the baseline analysis of several properties on the same machine, I am able to automatically detect whether there is any correlated anomalous behavior. Correlated anomalous behavior often in-

dicates abnormal behavior. This relationship allows me to automatically detect abnormal behavior. In addition to automatically detecting abnormal behavior, my system requires very little set up or configuration. To detect abnormal behavior, the system administrator must simply select a set of time series that display normal behavior.

RAACD does not detect abnormal behavior in a timely fashion. The delay is due primarily to the frequency at which I monitor the hosts in NodeScape. Low monitoring frequencies mean that new data come into the system infrequently, and the time difference between samples may be large. A single SAX symbol often represents at least 3 samples; for a monitoring frequency of once every 10 minutes, a single SAX symbol represents 30 minutes. I cannot detect abnormal behavior until it appears in several symbols. In some cases, this requirement means that I do not detect abnormal behavior until it has been occurring for over 1 hour.

The current implementations of NodeScape and RAACD also do not detect or repair gaps in the health data. RAACD removes gaps from the time series by concatenating the parts of the series that surround the gap. This behavior may lead to false anomalous behavior because the time series is not likely to be continuous across a gap in the samples. The gap may also disrupt natural cycles in the data. One straightforward method to repair a gap in a time series is to repeat the behavior that leads up to the gap. Repeated behavior should not cause anomalous behavior on the earlier edge of the gap; the effect of this method on the later edge of the gap is unpredictable.

RAACD is successful at reducing the amount of data presented to the system administrator. I have tested RAACD both in compute environments and general-purpose workstation/server environments. I employ my system to monitor approximately 30 machines including workstations, servers, and compute nodes. Rarely does RAACD present information from more than 5 machines at once. RAACD does sometimes present a machine that is not behaving abnormally. The system usually presents information only about machines that behave abnormally.

Chapter 8 Future Work

The work I present in this thesis is the beginning of our research on how to automatically detect abnormal behavior in computing systems. Further work in this area includes novel areas of research as well as possible improvements to the methods and tools that I have developed.

Multi-host detection

I search for abnormal behavior by analyzing multiple properties for a single machine. In many cases, the cause of abnormal behavior actually affects multiple machines. For example, a failure of the cooling system would cause CPU temperatures on all of the computers in the machine room to rise. We would like to develop a method to analyze multiple machines for simultaneous anomalies. Implementing this method would require more configuration because the tool would need to be told how to group machines. We may also benefit from combining multi-host detection with multi-property detection.

More timely detection

I believe RAACD may be able to detect abnormal behavior more quickly. I can achieve more timely detection in two ways. First, I can modify NodeScape to collect measurements of each property more frequently. I can also modify NodeScape to collect more information than just an instantaneous measurement of each property. I can modify the monitor so that it collects the minimum, maximum, and mean values for the gaps between samples. This extra information would allow me to interpolate values between samples. Second, I can modify the multi-property detection method to detect partial patterns of abnormal behavior. Once I have detected a particular abnormal behavior several times, I may be able to characterize the pattern that leads up to that abnormal behavior. I may be able to detect abnormal behavior before observing the entire pattern of abnormal behavior.

Improvements to multi-property detection

My implementation of multi-property detection is crude. There may be more than one expected profile for baseline behavior on a machine. Currently, `check-prop` does not allow multiple baseline profiles for a single property on a single machine. Allowing multiple baselines for a property would require a more sophisticated analysis to keep properties with multiple baselines from carrying more weight that properties with only a single baseline.

A different approach to pre-computing profiles

The deficiencies in my method for searching for a particular anomaly in a time series are partially due to the way that I pre-compute the profile for an anomaly. I compute the subword histogram for all of the words in the series. There may be better ways to build a profile for an anomaly that more precisely encode the characteristics of the anomaly.

One idea is to represent a profile as a sequence of subword histograms, where each histogram represents a different inspection window from the profile series. I would search for this type of profile by looking for a similar sequence as I slide the inspection window across the time series.

Improvements to `check-prop`

I use `check-prop` to test my methods on real data, but the tool is far from production quality. The code is not well organized, I have no process for deployment, and there is no documentation. I do not have any central configuration interface. The baseline profiles that check-prop uses are built using a separate process and separate tools. `Check-prop` also has performance problems. Database queries take a very long time, and some of the algorithms could be changed to increase performance. In addition to the problems with `check-prop` itself, NodeScape v2, which check-prop depends on for collection and storage of data, is not ready for release. I have not thought much about security for either `check-prop` or NodeScape v2. To release `check-prop`, I at least need to write documentation and installation instructions for `check-prop` and must add some security features to NodeScape v2.

Integration into other monitoring tools

I have implemented multi-property detection in RAACD as a front-end for NodeScape, but I may be able to adapt multi-property detection to work with other infrastructure monitoring packages. Specifically, I may be able to integrate multi-property detection into Pulsar and Nagios. To integrate multi-property detection into Pulsar, I would implement my method as a monitor. I would use a discomfort level of 0 for hosts that behave normally and a much higher discomfort level, perhaps 20, for hosts that behave abnormally. I could vary the discomfort level based on the number of properties exhibiting anomalous behavior.

There are two ways that I might integrate multi-property detection into Nagios. Instead of applying multi-property detection to a set of anomaly scores, I would use the results of service checks run by Nagios. I would implement multi-property detection as a new service check. The service check would return WARNING if several of the service checks that it monitors also return WARNING. I would return CRITICAL if several of the service checks return CRITICAL. This approach is similar to on-demand checks in Nagios; on-demand checks allow Nagios to query the state of the host when a service running on that host changes state. The second

way that I might implement multi-property detection in Nagios is also as a service check. The service check would return WARNING if multiple properties exhibit anomalous behavior at once and would return CRITICAL if many properties exhibit anomalous behavior at once.

Study and implementation of other anomaly detection schemes

I would like to study other schemes for detecting anomalous and abnormal behavior. I briefly considered frequency-domain analysis and comparison with standard deviation as approaches to detect anomalies. I did not find either method useful, but a more thorough analysis of these methods may prove otherwise. I present methods in sections 3.2 and 3.3 that may also be useful for analyzing computer health information. I can substitute the immunology-based method for baseline analysis with very few changes to the structure of my implementation. One weakness of my approach is that I do not detect long-running trends. I may be able to use TSX to build trend analysis into RAACD. Finally, there are other distance functions for measuring distance between two SAX representations. I can apply any of the above methods in place of comparing subword histograms.

Study other SAX alphabet sizes

I use a 4-symbol alphabet for SAX conversion, but SAX works with an alphabet of any size. Pouget et al. apply SAX with a 7-symbol alphabet. One advantage of a larger alphabet is that it preserves more information from the original series. In particular, alphabets with an odd number of symbols provide a center symbol. Because I only use 4 symbols, SAX converts a time series with only 1 value into a series of words containing only the letter ”c”. SAX uses the symbol ”c” because the average value of every normalized segment is 0, which is the center breakpoint for an alphabet with an even number of symbols. The choice of the symbol ”c” for the value 0 is arbitrary; I could have instead chosen to use ”b”. An alphabet with an odd number of symbols eliminates this arbitrary decision. I would like to study whether alphabets with odd cardinality provide better encodings than alphabets with even cardinality.

Bibliography

[1] Cacti - The Complete RRDTool-based Graphing Solution. http://www.cacti.net.2013.

[2] Ganglia Monitoring System. http://ganglia.info.2013.

[3] Munin. http://munin-monitoring.org.2013.

[4] Nagios - The Industry Standard in IT Infrastructure Monitoring. http://www.nagios.org.

[5] RRDtool - About RRDtool. http://oss.oetiker.ch/rrdtool/.2013.

[6] Dipankar Dasgupta and Stephanie Forrest. Novelty detection in time series data using ideas from immunology. In *In Proceedings of The International Conference on Intelligent Systems*, 1995.

[7] H. Dietz. The Aggregate's nodescape Utility. http://www.aggregate.org/NODESCAPE, 2011.

[8] Raphael A. Finkel. Pulsar: an extensible tool for monitoring large Unix sites. *Software: Practice and Experience*, 27(10):1163–1176, 1997.

[9] Eamonn Keogh, Stefano Lonardi, and Bill 'Yuan-chi' Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 550–556, New York, NY, USA, 2002. ACM.

[10] Guiling Li, Liping Zhang, and Linquan Yang. Tsx: A novel symbolic representation for financial time series. In Patricia Anthony, Mitsuru Ishizuka, and Dickson Lukose, editors, *PRICAI 2012: Trends in Artificial Intelligence*, volume 7458 of *Lecture Notes in Computer Science*, pages 262–273. Springer Berlin Heidelberg, 2012.

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

[12] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel. Finding motifs in time series. pages 53–68, 2002.

[13] Fabien Pouget, Guillaume Urvoy Keller, and Marc Dacier. Time signatures to detect multi-headed stealthy attack tools. In *18th Annual FIRST Conference, June 25-30, 2006, Baltimore, USA*, Baltimore, UNITED STATES, 06 2006.

[14] Xuan Shang, Ke Chen, Lidan Shou, Gang Chen, and Tianlei Hu. (k,p)-anonymity: towards pattern-preserving anonymity of time-series data. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 1333–1336, New York, NY, USA, 2010. ACM.

[15] Latanya Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

[16] Li Wei, Nitin Kumar, Venkata Lolla, Eamonn J. Keogh, Stefano Lonardi, and Chotirat Ratanamahatana. Assumption-free anomaly detection in time series. In *Proceedings of the 17th international conference on Scientific and statistical database management*, SSDBM'2005, pages 237–240, Berkeley, CA, US, 2005. Lawrence Berkeley Laboratory.

Vita

J. Frank Roberts

Education:

Bachelor of Science in Computer Engineering May 2011
University of Kentucky

Experience:

Teaching Assistant August 2012 - Present
University of Kentucky, Department of Computer Science

Graduate Research Assistant June 2011 - August 2012
University of Kentucky, Aggregate.org research consortium

IT Assistant March 2007 - Present
Bell Engineering

Honors and Awards:

Eagle Scout, 2006
Kentucky Governor's Scholar, 2006
Completed UK Honors Program (Undergraduate)

Projects:

CCSVLIB Winter 2013
Role: sole developer
CCSVLIB is a C library for reading, manipulating, and writing comma-separated-values files. The library presents a small, simple API. CCSVLIB is released under the BSD license.
`http://www.jafrro.net/software`

KOAP: Kentucky OpenCL Application Preprocessor Fall 2011
Role: sole developer
KOAP allows the programmer to simplify and aggregate calls to the OpenCL C API.
`http://aggregate.org/KOAP`

NodeScape Summer 2011 - Present
Role: team member
NodeScape uses analysis and presentation techniques to solve some of the scalability issues inherent in monitoring large computing systems.
`http://aggregate.org/NODESCAPE`