Survey like paper. Gives good details of the algorithms. Less info on approximate pattern detection.

Anomaly detection and motif discovery in symbolic representations of time series

Fabio Guigou, Pierre Collet, Pierre Parrend

Anomaly detection and motif discovery in symbolic representations of time series

Fabio Guigou, Pierre Collet, Pierre Parrend

TECHNICAL REPORT N°69427/2

Thursday 13th April, 2017

4PFactory: E-Laboratory on the Factory of the Future

Anomaly detection and motif discovery in symbolic representations of time series

Fabio Guigou Pierre Collet Pierre Parrend

Thursday 13th April, 2017

Abstract

The advent of the Big Data hype and the consistent recollection of event logs and real-time data from sensors, monitoring software and machine configuration has generated a huge amount of time-varying data in about every sector of the industry. Rule-based processing of such data has ceased to be relevant in many scenarios where anomaly detection and pattern mining have to be entirely accomplished by the machine. Since the early 2000s, the de-facto standard for representing time series has been the Symbolic Aggregate approXimation (SAX). In this document, we present a few algorithms using this representation for anomaly detection and motif discovery, also known as pattern mining, in such data. We propose a benchmark of anomaly detection algorithms using data from Cloud monitoring software.

Keywords

Time series, Symbolic representation, Anomaly detection, Pattern mining

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem statement

We consider the problem of detecting anomalies and establishing baseline behaviour in time series captured by network monitoring software. Since the inception of such software, failure detection has mostly been performed by polling a number of metrics, storing them for expert analysis and applying simple decision rules based on the last few data points of each series independently, e.g. signaling a failure when all points within a short analysis window stay above a given threshold. While this approach has worked well enough to keep data-centers running, it still fails to capture many important events, such as deviations from a normal baseline or early signs of failure. While algorithms exist to perform such tasks, using them for monitoring would require significant hardware upgrades. We study alternative methods to fill the gap between crude expert systems and costly anomaly detection and pattern mining algorithms.

1.2 Change and anomaly detection in time series

1.2.1 Symbolic representations

Anomaly detection in time series is a prominent task in data-mining. However, the size and number of such series makes it extremely demanding in terms of computational power. To overcome this issue, many alternative representations have been proposed for time series: Discrete Fourier Transform, Wavelet Transform [1], Singular Value Decomposition... SAX was proposed in 2003 [2] as a simple, compact, text-based representation that reduces dimensionality and allows the use of string processing algorithms to analyze time series [3]. It has seen applications in time series indexing [4], visualization [5] and various mining tasks [6, 7, 8]. Applications have even been tried on objects that are only remotely connected to time series, such as motion detection [9].

1.2.2 On raw data

Other statistical approaches have been used on raw time series data. Change point detection, i.e. detecting when a model stops fitting the data and a new one must be derived, has been implemented in many ways: using Bayesian models [10] or sequential testing [11, 12] to perform online detection. These methods have proven computationally efficient and able to reliably detect structural changes in data stream. However, they do not offer the wide range of applications that come with symbolic representation, and especially string representation. In this report, we focus on the opportunities offered by this paradigm.

1.3 Context and goals

In the ever growing field of Cloud computing and Cloud networking, service providers face the challenge of operating a high number of devices, both physical and virtual, while maintaining a contractually defined service level, known as SLA (Service Level Agreement). This situation calls for a tight monitoring of all the infrastructure. Monitoring software typically polls devices using ICMP and SNMP, collects data such as response time, CPU load, memory usage, and compares these values to thresholds. In the event of a value crossing the alert threshold, some form of warning is sent to the administrators.

While such software "does the job", it is often imprecise, yields lots of false positive or raises alerts too late, when the device has already failed or a customer has already opened a case for performance degradation. These shortcomings indicate the need for more advanced techniques, such as behavioral anomaly detection, that can either replace or improve the current methods.

The point of this study is to determine the feasibility of real-time anomaly detection in computer network monitoring time series. The advantages of such approach are expected to be low computational costs and good accuracy. Since monitoring, even using only crude threshold methods, is already CPU-intensive, we turned to symbolic representations of time series to search for a method compatible with low-resource monitoring servers.

2 The SAX representation

SAX is a time series representation designed to vastly reduce the data dimensionality and redundancy by subsampling and quantization. Typical settings use a subsampling factor n of 8 to 10 and only $\alpha = 3$ or 4 bins for quantization. It is widely accepted that the impact of this parameter is small. The basic algorithm operates on the whole time series by applying a z-normalization (setting the mean to 0 and standard deviation to 1), replacing each non-overlapping window of n points by its average – a process known as Piecewise Aggregate Approximation (PAA) – and applying a Gaussian quantization of this PAA into α bins.

It is worth noting that, while the conversion of a time series into a character string is not a familiar operation, the subsample-and-quantize process applied is the basis of any lossy compression algorithm: JPEG for image, MPEG for video, A-law and μ-law for audio... The vector quantization, i.e. the dispatching of values into bins, depends on the original assumptions on the underlying statistical distribution. SAX [2] assumes a normal distribution of the time series values at the scale of a SAX word, as shown in Algorithm 1.

Algorithm 1 Basic SAX encoding algorithm

```
procedure SAXENCODE
input:
    $S = [s_1, s_2, ..., s_N]$                                  ▷ real-valued time series
    $\alpha = int$                                   ▷ cardinality (typical values: 3 or 4)
    $n = int$                                    ▷ symbol size (typical values: 8 to 10)
output:
    $s = string$                                              ▷ SAX representation
algo:
    ▷ GQF is the Gaussian Quartile Function
    $breakpoints = GQF(1/(\alpha-1), 2/(\alpha-1), ..., (\alpha-2)/(\alpha-1))$
    ▷ (for $\alpha = 3$, $breakpoints = [-0.43, 0.43]$; for $\alpha = 4$, $breakpoints = [-0.67, 0, 0.67]$)
    $S = (S - mean(S))/std(S)$
    $S_{aggr} = [sum(s_1, s_2, ..., s_n)/n, sum(s_{n+1}, s_{n+2}, ..., s_{2n})/n, ...]$
    $s = quantize(S_{aggr}, breakpoints)$
```

However, though this approximation is able to capture the main characteristics of a time series, it is not always appropriate for algorithms designed to operate on fixed-size sliding windows. In order to analyze long running series of thousands of points, the SAX encoding can be applied to such sliding windows as shown in Algorithm 2.

All following algorithms use this representation (list of words with full overlap) unless otherwise stated. The sliding window is often referred to as a "feature window" in the literature because the window size is tuned to the approximate size of the feature one wishes to extract from the series. This is particularly true for periodic time series, such as ECG datasets or sound analysis [13].

Algorithm 2 The most widely used SAX encoding, using a sliding window

```
procedure SAX
input:
    $S = [s_1, s_2, ..., s_N]$        ▷ long real-valued time series
    $\alpha = int$        ▷ cardinality
    $n = int$        ▷ symbol size
    $w = int$        ▷ sliding window size (feature size, usually user-specified)
output:
    $s' = [s'_1, s'_2, ..., s'_{N-w}]$        ▷ list of SAX words
algo:
    ▷ sliding window extraction
    $windows = [[s_1, s_2, ..., s_w], [s_2, s_3, ..., s_{w+1}], ..., [s_{N-w}, s_{N-w+1}, ..., s_N]]$
    $s' = SAXEncode(windows, \alpha, n)$
```

3 Anomaly detection

3.1 Hot SAX

Hot SAX [14] is not an anomaly detection algorithm *per se*. Instead, it is a heuristic based on the SAX representation of a time series to accelerate the brute-force algorithm. The "obvious" way to detect an anomaly is to search for the subsequence with the highest distance to any other subsequence, i.e. finding $argmax_i(min_j(dist(x_i, x_j))), |i-j| > n$ over the set x of all sliding windows of n points extracted from the time series t. This search involves a quadratic number of distance computations and therefore is not suitable for more than a few hundred points. Note that self-matches are excluded: the nearest neighbor of a subsequence is only searched among other subsequences having no point in common with it. The naive implementation is shown in Algorithm 3:

Algorithm 3 Brute-force anomaly detection

```
procedure BRUTE FORCE DISCOVERY
input:
    $T = [t_1, t_2, ..., t_N]$                                   ▷ time series
    $n = int$                                                    ▷ window size
output:
    $dist = float$             ▷ max Euclidean distance between any 2 subsequences
    $loc = int$                        ▷ location of most anomalous subsequence
algo:
    $dist = 0$
    $loc = null$
    for $p = 1$ to $N$ do[0]
        $nndist = inf$
        for $q = 1$ to $N - n + 1$ do
            if $|p - q| \leq n$ then
                next
            if $dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}]) < nndist$ then
                $nndist = dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}])$
        if $nndist > dist$ then
            $dist = nndist$
            $loc = p$
```

However, changing the order in which distances are computed, while not changing the end result, may speed up the search by multiple orders of magnitude (while retaining a complexity of $O(N^2)$) by allowing early terminations of the loops, as shown in Algorithm 4.

Using the SAX representation, the outer and inner heuristics index the time series by its symbolic approximation and use the simple assumptions that rare SAX words correspond to rarely occurring subsequences, and that two similar sequences will have a similar SAX representation. Therefore, the outer heuristic first searches the most anomalous subsequence among thoses represented by the rarest SAX word, and the inner

Algorithm 4 Anomaly detection, with heuristics

```
procedure HOT SAX
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $n = int$                                                   ▷ window size
output:
    $dist = float$              ▷ max Euclidean distance between any 2 subsequences
    $loc = int$                          ▷ location of most anomalous subsequence
algo:
    $dist = 0$
    $loc = null$
    for $p = 1$ to $N$ order by OuterHeuristic do
        $nndist = inf$
        for $q = 1$ to $N - n + 1$ order by InnerHeuristic do
            if $|p - q| \leq n$ then
                next
            if $dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}]) < nndist$ then
                $nndist = dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}])$
            if $dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}]) < dist$ then
                break
        if $nndist > dist$ then
            $dist = nndist$
            $loc = p$
```

heuristic searches any subsequence's neighbors among other subsequences represented by the same word as shown in Algorithm 5.

Algorithm 5 Heuristics used for anomaly detection

```
procedure OuterHeuristic
input:
    $T = [t_1, t_2, ..., t_N]$                                ▷ time series
    $n = int$                                                 ▷ window size
output:
    $P = [p_1, p_2, ..., p_N]$                                ▷ ordered indexes
    $H = \{word => [indexes]\}$                               ▷ indexed SAX representation of T
algo:
    $words = SAX(T, n)$
    $H = hash\{word => [indexes\ of\ each\ occurrence\ of\ this\ word]\}$
    $min = min(hash.values)$                                  ▷ (min = almost always 1)
    $P = [indexes\ of\ words\ occurring\ only\ min\ time] + [other\ indexes,\ shuffled]$
procedure InnerHeuristic
input:
    $T = [t_1, t_2, ..., t_N]$                                ▷ time series
    $H = \{word => [indexes]\}$                               ▷ indexed SAX representation of T
    $p = int$                                                 ▷ index of current subsequence
output:
    $Q = [q_1, q_2, ..., q_N]$                                ▷ ordered indexes
algo:
    $Q = H\{words[p]\} + [other\ indexes,\ shuffled]$
```

3.2 Sequitur

The Sequitur [15] algorithm proposed by Nevill-Manning in 1997 [16] is a dictionary-based compression algorithm. It uses the concepts of symbol, rule and digram to build a compact representation of its input data. In Sequitur vocabulary, a symbol is either an input token (e.g. a single character, or byte) or a token representing a rule; a rule is a symbol standing for a digram; and a digram is a pair of symbols. Therefore, in a string "abc", "a", "b" and "c" are symbols and "ab" and "bc" are digrams. A rule will have the form "A = ab", meaning that a string "Ac" can be read as "abc". With such a rule, "A" and "c" are symbols and "Ac" is a digram that can be itself part of another rule.

Since Sequitur builds a compact, context-free generative grammar for any sequence, it has been used for various analytical tasks, such as program trace analysis [17], query of compressed XML databases [18] or structure inference in DNA and musical pieces [19]. The algorithm has a complexity of $O(N)$, which makes it suitable even for large series.

Sequitur transforms any input sequence into a compact representation where two essentials constraints are met: no digram appears more than once in the output sequence (digram uniqueness), and no rule is used (either in the output sequence or in other rules) less than twice (rule utility). It is argued in [15] that the number of rules used to represent a given point in a Sequitur-compressed time series, being proportional to its compressibility, is a good approximation of the series' Kolmogorov complexity, or algorithmic complexity (i.e. the size of the smallest program capable of generating the series) at that point. The assumption is that an anomaly is very likely to correspond to a rising complexity, or a lowering compressibility.

Sequitur builds a grammar tree in which the depth of a leaf is directly related to its frequency in the original string. Higher level, non-terminal nodes in the tree correspond to frequent patterns in the data. Conversely, shallow branches denote rare patterns, which is of interest in anomaly detection. With the analytic decompression routine that follows, the rule density, i.e. the depth in the grammar tree, can be associated to any point in the series from the Sequitur representation as shown in Algorithm 6.

Algorithm 6 Analytic unwrapping procedure

```
unwrap(token, depth, rules) =
  if token not in rules then (token, depth)
  else [unwrap(rules{token}[1], depth + 1, rules),
    unwrap(rules{token}[2], depth + 1, rules)]

unwrap(blob, rules) =
  unwrap(token, 0, rules) for each token in blob
```

The anomaly detection algorithm proposed uses as Sequitur symbols the SAX words representing subsequences (i.e. a basic symbol could be 'aabacd') as shown in Algorithm 7.

Algorithm 7 Anomaly detection using the Sequitur compression algorithm

```
procedure SequiturAnomaly
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $n = int$                                                  ▷ window size
output:
    ▷ Sequitur rule density @ each point of the series (inverse anomaly score)
    $density = [d_1, d_2, ..., d_N]$
algo:
    $words = SAX(T, n)$
    ▷ SAX words [e.g. 'abbcab'] are used as elementary alphabet for Sequitur
    $tokens, rules = Sequitur(words)$
    $tokens$ = $list\ of\ Sequitur\ symbols$
    $rules$ = $hash\ \{symbol \rightarrow [symbol|letter, symbol|letter]\}$
    $unwrapped = []$
    for all $token$ in $tokens$ do
        $unwrapped << unwrap(token, 0, rules)$       ▷ $unwrapped$ is [(SAX word, depth)]
    for $i = 1$ to $N$ do
        $density = sum(unwrapped[max(i-n, 0), max(i-n+1, 0), ..., i])$
```

3.3 Chaos game representation

The Chaos Game representation [20, 21] is a way of generating a bitmap from a DNA fragment, i.e. a string generated by an alphabet of four symbols (A, C, T, G). It recursively splits the two-dimensional space into pixels representing the number of occurrences of specific strings, adding one character at each level (e.g. at level one, it generates a 4-pixel image with the total count of A, C, G, and T bases. At level 2, the image is 4-pixel wide, with pixel corresponding to length-2 strings such as AA, AC, AT, ..., TG, TT). The proposed algorithm uses the SAX representation of a time series, with an alphabet size of 4, to build such bitmaps [22], and then compare the bitmap of a detection window (after the currently analyzed point) and a lag window (before the point). Typically the lag window is 2 or 3 times longer than the detection window, therefore the bitmap must first be scaled.

Besides the graphical aspect of the bitmap generation, this algorithm is a simple histogram comparison: the frequency distribution of all possible N symbols strings is computed to the left and right of each point in the time series and the two distributions are compared to yield an anomaly score. Since the number of bins in the histograms grows exponentially with the analysis level, the SAX sliding window will typically be short. The authors use a level 3 analysis, i.e. 64-bin histograms.

The anomaly score at any given point t_i of the time series is given by $d_i = \sum_{j=1}^{N}(H_{i-lead,i} - H_{i,i+lag})^2$, where $H_{i,j}$ is the histogram computed between points t_i and t_j, as shown in Algorithm 8. Results presented in [20] show that even subtle structural anomalies in periodic series can be detected.

Algorithm 8 Anomaly detection using the Chaos Game sequence representation

```
procedure ChaosGameAnomaly
input:
    $T = [t_1, t_2, ..., t_N]$                    ▷ time series
    $n = int$                                     ▷ symbol size
    $w = int$                                     ▷ window size
    $D = int$        ▷ detection window length (in number of SAX windows)
    $L = int$              ▷ lag window length (in number of SAX windows)
output:
    $score = [s_1, s_2, ..., s_N]$                ▷ anomaly score
algo:
    words = SAX(T, 4, n, w)
    for $i = 1 + L$ to $N - D$ do
        $det\_map = ChaosGame(words[i, i+n, ..., i+n*(D-1)])/D$
        $lag\_map = ChaosGame(words[i-L*n, i-(L-1)*n, ..., i-n])/L$
        $score_i = EuclidianDistance(det\_map, lag\_map)$
```

4 Motif discovery

4.1 Minimal Description Length

While not directly using the SAX representation, the Minimal Description Length algorithm [23] transforms the time series in a symbolic representation by means of a multiscale Gaussian blur called scale-space image; at each level, segments bounded by the zero-crossings of the first derivative of the smoothed time series are extracted. These segments are compactly described by two coordinates: their length and the difference between their first and last point. This description is then quantized via a k-means clustering step. The time series at each scale can be processed as a string.

The Minimal Description Length (MDL) framework is an approach in which a motif is selected if it increases the compression ratio of the time series, i.e. it allows for a shorter description. Given the complexity of it exact computation (which would require the extraction and evaluation of all possible sets of motifs), a heuristic is used: only one motif is extracted at each scale. Typically, 8 scales are used. This algorithm, used on complex and noisy data sets such as vibration measured on a bridge, is able to isolate multiscale overlapping patterns, such as the passing of a single car on the aforementioned bridge as shown in Algorithm 9.

Algorithm 9 Motif detection using the Minimal Description Length framework

```
procedure MDLMotif
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $n = int$                                                  ▷ decomposition level
output:
    $M = (m_1, m_2, ..., m_N)$                                  ▷ set of motifs
algo:
    $M = ()$
    for $i = 1$ to $n$ do
        $detail_i = T \star h_i$                                ▷ $h_i$ gaussian kernel of size $2^i$
        $deriv_i = detail_{i,j+1} - detail_{i,j}, j \in \{1,2,...,N-1\}$ 1st        ▷ derivative
        ▷ Computation of zero-crossings of the derivative
        $zC = \{j | deriv_{i,j} = 0\}$
        $segments_i = (\{zC_{k+1} - zC_k, detail_{i,zC_{k+1}} - detail_{i,zC_{k+1}}\}), k \in \{1,2,...,|zC|-1\}$
        $motifs_i$ = set of repeating strings in $segments_i$
        $best\_motif_i = argmin_{m \in motifs_i} L(m) + L(T|m)$    ▷ best predictor for $T$ at scale $i$
        $M = M \cup best\_motif_i$
```

4.2 Grammar inference

As shown in the previous section, grammar inference [24] can be used not only to detect anomalies but, more naturally, to extract motifs. In this approach, the time series is SAX-encoded with a sliding window and converted into its Sequitur representation in the exact same way as for anomaly detection. The rules created during the compression are then mapped back to time series segments. Contrary to anomaly detection, the points with the highest rule density, i.e. the longest branches in the Sequitur grammar tree, are selected. A post-processing step is required to refine the results by:

- eliminating self-matches (motif matching itself within a very small offset);
- selecting the longest patterns, which are more likely to convey useful information;
- removing “obvious” patterns, e.g. monotonically increasing or constant ones; and
- merging overlapping patterns if necessary

4.3 MK algorithm

The MK (Mueen-Keogh) [25] algorithm is much related to HOTSAX, in that it uses a heuristic to prune an otherwise exact search for the most similar (instead of most different) pair of subsequences inside a time series. However, this algorithm is the only one in our collection operating on the raw time series data instead of a symbolic representation. We chose to include it in this report because of the similarity it bears to HOTSAX.

Instead of a SAX pre-processing to lower the number of distance evaluations, the algorithm uses reference points: subsequences chosen at random in the time series, and orders the other subsequences by their distance to the reference. The heuristic is based on the assumption that subsequences that are close to each other will also be close in this projection as shown in Algorithm 10.

Note that the algorithm can use multiple reference points, using only one to perform the subsequence reordering and all of them to compute *best_dist*. This further increases the convergence. As with HOTSAX, the overall complexity of the algorithm is not different from brute force search ($O(N^2)$), but the efficient pruning of distance computations makes the actual runtime orders of magnitude faster.

4.4 Motif Tracking algorithm

The Motif Tracking algorithm [26], shown in Algorithm 11, is an attempt to bring the Artificial Immune Systems (AIS) class of algorithms that operate almost only on strings into the field of time series processing. The SAX representation is used to produce the antibodies/antigens. It uses a subsequence length equal to the word size, therefore each subsequence is represented by a single symbol. Trackers are used as memory cells; they contain the string representation of a subsequence and a match count. Contrary to usual AIS algorithm, this one is deterministic.

Since this algorithm detects exact motifs (without any mutation between any two occurrences) when they appear at least twice, it is logically equivalent to the grammar induction algorithm described earlier.

4.5 Mining approximate motifs

The ‘Mining approximate motifs’ algorithm [27], given in Algorithm 12, proposes a number of innovations: it uses aggregative clustering to detect motifs and Pearson’s r instead of Euclidean distance to match potential motif occurrences. Matches are encoded as 2-clusters that are later merged and extended. Note that the original algorithm is proposed for a database of time series; we show an adapted version for a single long time series.

The original algorithm proposes a way to extend the motif instances; however, our use of SAX sliding windows makes this step inconvenient and mostly useless: the motif length is an initial parameter. In this case, it is also possible to use SAX’s lower-bound distance or Hamming distance as a similarity measure.

Algorithm 10 Mueen-Keogh algorithm

```
procedure MKMOTIF
input:
    $T = [t_1, t_2, ..., t_N]$                                    ▷ time series
    $n = int$                                                     ▷ motif length
output:
    $S_1, S_2$                        ▷ pair of subsequences with smallest distance
algo:
    $best\_dist = \infty$
    $ref = [T_r, T_{r+1}, ..., T_{r+n-1}]$, $r$ random
    for $i = 1$ to $N - n$ do
        $dist_i = EuclidianDistance(ref, [T_i, T_{i+1}, ..., T_{i+n-1}])$
        if $dist_i < best\_dist$ then
            $best\_dist = dist_i$
            $S_1 = ref$
            $S_2 = [T_i, T_{i+1}, ..., T_{i+n-1}]$
    Order subsequences $s_1, s_2, ..., s_{N-n}$ by their corresponding distances with function $I_j$ mapping
to their original positions
    $offset = 0$
    $abandon = 0$
    while not abandon do
        $offset = offset + 1$
        $abandon = 1$
        for $j = 1$ to $N - n$ do
            if $dist_{I_j} - dist_{I_{j+offset}} < best\_dist$ then
                $abandon = 0$
                if $EuclidianDistance(s_{I_j}, s_{I_{j+offset}}) < best\_dist$ then
                    $best\_dist = EuclidianDistance(s_{I_j}, s_{I_{j+offset}})$
                    $S_1 = s_{I_j}$
                    $S_2 = s_{I_{j+offset}}$
```

Algorithm 11 The motif tracking algorithm

```
procedure MotifTracking
input:
    $T = [t_1, t_2, ..., t_N]$                              ▷ first differential of time series
    $\alpha = int$                                           ▷ alphabet size
    $s = 1$                                                  ▷ subsequence length
    $r)float$                                                ▷ matching threshold (Euclidian distance)
output:
    $M = (m_1, m_2, ..., m_N)$                               ▷ set of motifs
algo:
    ▷ Trackers are SAX alphabet symbols with associated scores and indexes of occurrences
    $trackers = ([a,0,],[b,0,],[c,0,],...)$
    $symbols = [u_1, u_2, ..., u_{N-s}] = SAX(T, \alpha, s, s)$
    $trackers\_change = 0$
    $l = 1$                                                  ▷ motif length
    repeat
        for $i = 1toN - s - l$ do
            for all $tracker, score, indexes \in trackers$ do
                if $[u_i, u_{i+1}, ..., u_{i+l}] = tracker$ then
                    if $\forall j \in indexes, dist([t_i, t_{i+1}, ..., t_{i+s}], [t_j, t_{j+1}, ..., t_{j+s}])/s < r$ then
                        $score += 1$
                        $indexes = indexes \cup i$
        for all $tracker, score \in trackers$ do
            if $score < 2$ then
                Remove $tracker$ from $trackers$
            else
                for all $symbol \in alphabet$ do
                    Add $[tracker + symbol, 0]$ to $trackers$
        $s += 1$
    until $trackers\_change = 0$
    $M = trackers$
```

Algorithm 12 Mining approximate motifs with aggregative clustering

```
procedure AggregativeClusteringMotif
input:
    $T = [t_1, t_2, ..., t_N]$                                    ▷ time series
    $R_{min} = float$                                             ▷ similarity threshold
    $\alpha, word, window$                                        ▷ SAX parameters
output:
    $M = (m_1, m_2, ..., m_N)$                                    ▷ set of motifs
algo:
    $D = SAX(T, \alpha, word, window)$
    $clusters = \{\}$
    for $i = 1$ to $N$ do
        for $j = i + window$ to $N - s$ do
            if $|r(D_i, D_j)| \geq R_{min}$ then
                $clusters = clusters \cup < i, j >$
    repeat
        $updates = 0$
        for all $X = [x_1, x_2, ..., x_{|X|}] \in clusters$ do
            for all $Y = [y_1, y_2, ..., y_{|Y|}] \in clusters \setminus X$ do
                if $\forall x \in X \forall y \in Y |r(D_x, D_y)| \geq R_{min}$ then
                    $clusters = clusters \setminus X \setminus Y \cup < x_1, x_2, ..., y_1, y_2, ... >$
                    $updates += 1$
    until $updates = 0$
```

5 Experimental evaluation

5.1 Input data

To benchmark these algorithms, we use a collection of 14 time series spanning over two months, representing CPU load, memory usage, process count and active TCP sessions of 3 production servers and firewalls. The metrics are acquired with a temporal resolution of one point per minute. These series have been selected because they contain both identifiable motives and anomalies. In particular, they display various failure modes, some of which invalidate the basic assumptions of certain algorithms and therefore are not detected.

Anomalies such as depicted in figure 5.1 are easy to detect even for conventional, commercial monitoring software. However, more subtle changes (figure 5.1) that can be symptomatic of a critical component crashing are harder to detect and generally overlooked; instead, the crash might be detected later when it causes other symptoms and impact on other parts of the system. Detecting such types of anomalies is key to faster correction, smaller impact and easier root cause analysis.

In the context of IT network monitoring, awareness of motives can simplify analysis by two means: on the one hand, having an expert associate a label to a motif makes it possible to recognize a possibly complex and rare event in real time. On the other hand, periodicity makes traditional, threshold-crossing based methods unreliable because of the potentially large variation of the signal in its normal range. Being able to filter out large low-frequency components is a simple way to improve anomaly detection.

5.2 Anomaly detection benchmark

Since the very concept of an anomaly is hard to quantify (i.e. classifying data into "normal" or "abnormal" in time series can be somehow arbitrary, outside of the obvious transitions), we will perform a qualitative evaluation of these algorithms, taking into account execution speed (i.e. CPU load), failure modes (i.e. types of anomalies that cannot be detected), detection delay (i.e. number of data points required to spot an anomaly) and sensitivity to periodic input (i.e. precision loss due to the signal being periodic).

5.2.1 Note on figure reading

The blue graph is the raw time series, the green one the anomaly score and the red vertical lines indicate anomaly detection (anomaly score crossing a tunable threshold, here five standard deviations). The anomaly

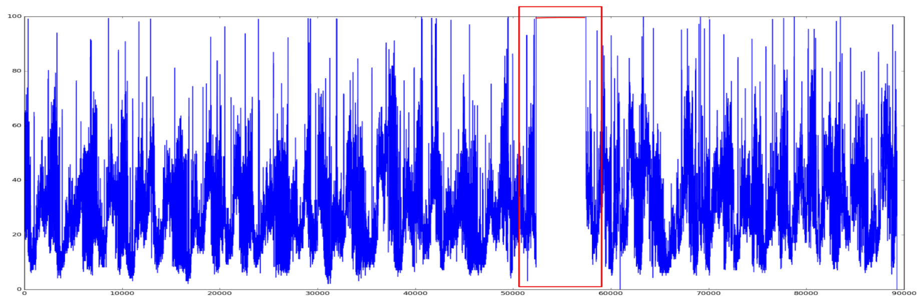

Figure 5.1: Anomaly in CPU load stuck at 100%

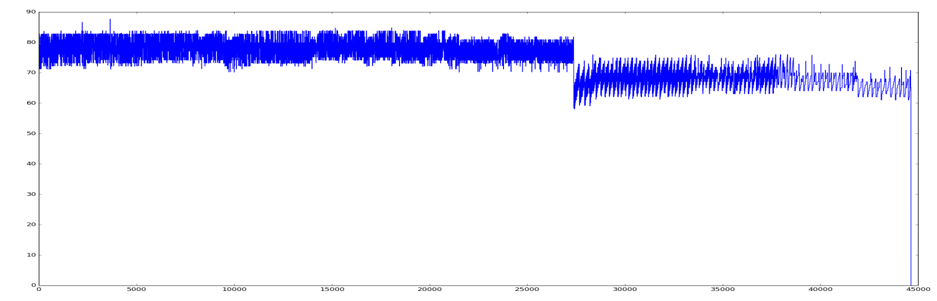

Figure 5.2: Transition in RAM usage (high and aperiodic to low and periodic)

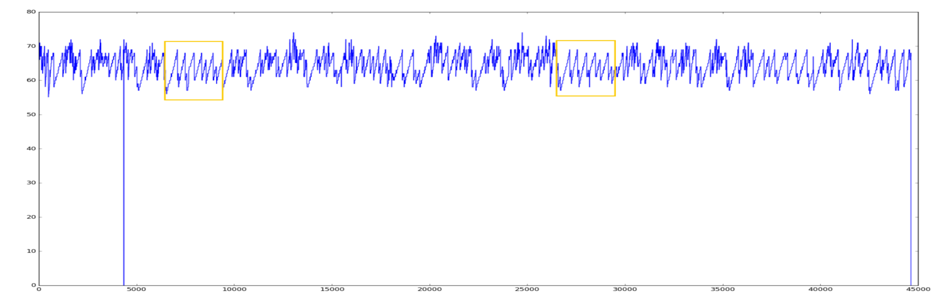

Figure 5.3: Small motif embedded in an apparently random RAM usage

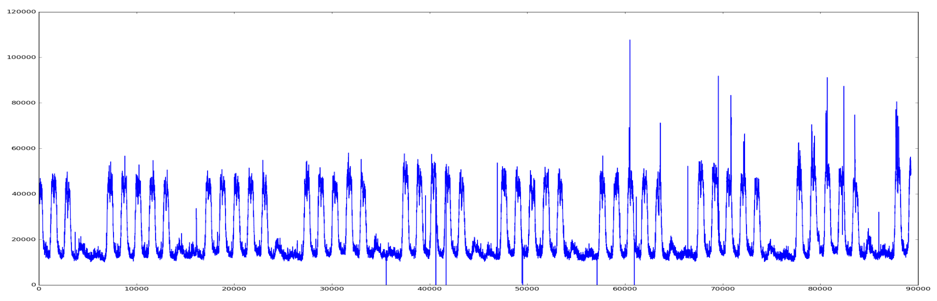

Figure 5.4: Obviously periodic TCP session count

Table 5.1: Runtime comparison of anomaly detection algorithms (mean time (s) / standard deviation)

Nb of points	Chaos Game	Sequitur	Hot SAX
44k	6.66/0.24	6.62/3.03	651/?
88k	12.67/0.45	12.07/5.10	?/?

score does not necessarily begin at the same point as the series: it is delayed by buffering in the case of Chaos Game, which implies that the alarm is only raised after that buffering time.

5.2.2 Performance figures

We first study the running time of the algorithms. Since we expect to monitor thousands of time series on each monitoring server, a low running time is of paramount importance. Results show that Sequitur and Chaos Game yield about the same running time, between 6 and 7 seconds per month of data. The running time, as predicted, is linear with the size of the series. Hot SAX, on the other hand, displays a much larger, and quadratic, running time; we only ran it once on one month of data, and therefore have no estimate as to the variability of our result of almost 11 minutes. On a time series of two months, the algorithm ran for more than an hour without producing any result. Given that it definitely eliminates it as a viable candidate for IT monitoring, we did not try to let it complete.

5.2.3 On SAX parameters

As stated in almost all papers by Keogh et.al., the parameters used for SAX encoding (cardinality and word size) have little impact on the behaviour of any algorithm. At most, they can change the sensitivity and the running time, e.g. trade off accuracy for faster execution, or move along the precision-recall curve. Window size, however, must generally be adapted to the scale of any feature in the series, i.e. the size of the anomaly we try to detect, or the period of the signal.

This is one of the major shortcomings we observed in all SAX-based algorithms: with mostly weekly periodic signals, the best settings would delay any analysis until after at least 7 *days* after a data point is acquired. This is due to the mismatch between the scale of the observed phenomena and the desired detection time. In most use cases, SAX is used to study phenomena of the scale of a second, with a few dozen points per occurrence. We study weekly patterns with thousands of points per occurrence; therefore, the periodic signal is mostly perceived as concept drift from the SAX perspective. However, even with this limitation, most anomalies can still be detected.

5.2.4 Hot SAX

As described above, Hot SAX is the only algorithm that does not generate an “anomaly score”, or any kind of distance measure. It only returns the single most anomalous point in a series. While this point, in all our tests, always corresponded to the real anomaly (or *one* of the anomalies), the algorithm remains extremely slow and its result is of little use in any real-time settings. In fact, our tests had trouble even completing, taking over 10 minutes for one single month of data (while the other algorithms only needed half a dozen seconds).

As Hot SAX, by its design, always outputs the mathematically defined worst anomaly, it has inherently no failure mode. In fact, it could be used in IT monitoring, with some restrictions:

- The data must be undersampled.
- The search window must be limited.

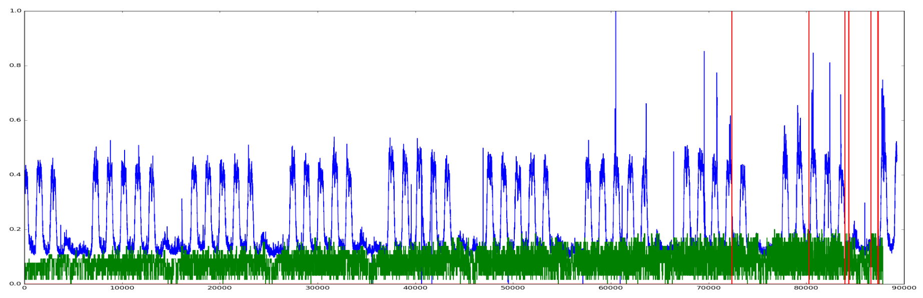

Figure 5.5: Weekly pattern ignored, anomalies detected when average value runs too high (Sequitur)

- Hot SAX must only be applied to some critical metrics.

If the time series in which the search is conducted can be maintained within a few thousands of points, and Hot SAX is not used on each and every series, it can be a very precise tool for advanced analytics. Its CPU cost is the main obstacle to its adoption as a standard algorithm for anomaly detection.

5.2.5 Sequitur

While fast and capable of catching most anomalies, as well as immune to cyclic phenomena, Sequitur relies on the low compressibility of anomalies to spot them. Therefore, it completely fails to detect any anomaly that results in a *simplified* pattern, like the one in figure 5.1. An interesting feature is that the running time of the algorithm depends more on the complexity (i.e. the number of rules generated by Sequitur) than the number of data points. We found it to be, performance-wise and in terms of detection speed, the best algorithm, requiring little to no look-ahead (in contrast to Chaos Game) and insensitive to cyclic variations (compare correct detection in figure 5.2.5 with errors in figure 5.2.6). However, the failure mode we observed is not compatible with a real production environment, in which it can indicate a serious failure or an attack.

Therefore, if Sequitur is to be used, it must be coupled with another algorithm able to detect such failures. It is worth noting that these stationary patterns are only dangerous when they happen at *high* values, e.g. 100% CPU. Thus a simple threshold-based failure detector would complement Sequitur in a satisfactory way.

5.2.6 Chaos Game

The Chaos Game algorithm is the most precise anomaly detector with acceptable runtime performance. We found no failure mode in our dataset. The running time is strictly proportional to the length of the time series, and therefore predictable. We found, however, that the look-ahead required to correctly perform anomaly detection (at least twice the feature window) is unacceptably long, and a short feature window makes the algorithm very sensitive to cyclic patterns. See figure 5.2.6 for an example of false positive (correctly handled by Sequitur in figure 5.2.5).

We could not find a correct tuning capable of ignoring these low-frequency variations without lengthening the window (and hence the time between an anomaly happening and it being detected) to at least a day. The only way to use the Chaos Game algorithm would be to couple it with a filter capable of rejecting the false positive after detection, or to remove the low-frequency components before analysis, which could potentially lead to a high false negative rate.

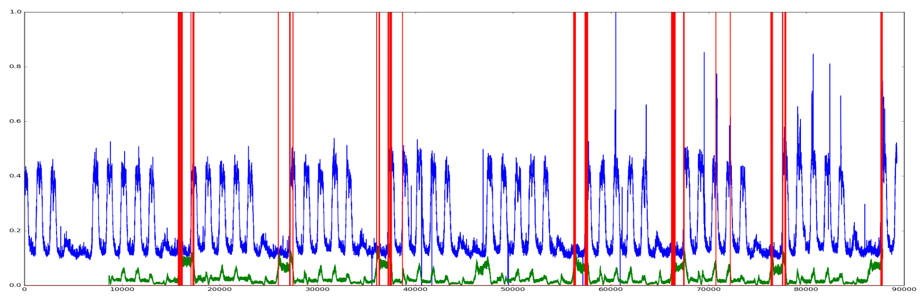

Figure 5.6: Wrongly detected anomalies due to a weekly cycling pattern (Chaos Game)

6 Conclusion and perspectives

After implementing and comparing various anomaly detection algorithms based on the SAX representation, we found that none is a perfect fit for real-time detection. This is due partly to the representation itself, which needs time to take a new point into account (due to its heavy downsampling and sliding window analysis). However, tuning the parameters for a good detection led to the conclusion that the feature window is bound to be too large for a practical analysis: for typical weekly patterns found in network monitoring, a good feature window is about one day long. This in turn leads to huge SAX words, with a single symbol representing an hour of data. We conclude that, while extremely useful for fast, “post-mortem” analysis, algorithms based on SAX are mostly unsuitable for the kind of real-time analysis we want to perform.

In light of these findings, we did not perform a benchmark of pattern mining algorithms, as the same limitations are bound to apply to them as well. However, given the low computational cost and acceptable results in our anomaly detection experiments, we suggest that hybrid algorithms, working on long time scales with symbolic representation and in real-time with the raw data, could be created to address the overly long detection delay. Conversely, hybrid representations containing both aggregate and raw (or quantized) data could improve detection times.

7 Scientific validation

This work has been reviewed and accepted for publication as a technical paper of the Complex System-Digital Campus by:

- Ismaila Diouf, McF, Université Cheikh Antar Diop de Dakar (Sénégal)
- Dominique Pastor, Professeur des Universités, Télécom Bretagne (France)

Bibliography

[1] Jessica Lin et al. “Iterative incremental clustering of time series”. In: *Advances in Database Technology-EDBT 2004*. Springer, 2004, pp. 106–122.

[2] Jessica Lin et al. “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*. ACM. 2003, pp. 2–11.

[3] Jessica Lin et al. “Experiencing SAX: a novel symbolic representation of time series”. In: *Data Mining and knowledge discovery* 15.2 (2007), pp. 107–144.

[4] Alessandro Camerra et al. “iSAX 2.0: Indexing and mining one billion time series”. In: *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE. 2010, pp. 58–67.

[5] Yi Gu and Chaoli Wang. “itree: Exploring time-varying data using indexable tree”. In: *Visualization Symposium (PacificVis), 2013 IEEE Pacific*. IEEE. 2013, pp. 137–144.

[6] Nguyen Quoc Viet Hung and Duong Tuan Anh. “Combining SAX and piecewise linear approximation to improve similarity search on financial time series”. In: *Information Technology Convergence, 2007. ISITC 2007. International Symposium on*. IEEE. 2007, pp. 58–62.

[7] Thanawin Rakthanmanon and Eamonn Keogh. “Fast shapelets: A scalable algorithm for discovering time series shapelets”. In: *Proceedings of the thirteenth SIAM conference on data mining (SDM)*. SIAM. 2013, pp. 668–676.

[8] Pavel Senin and Sergey Malinchik. “Sax-vsm: Interpretable time series classification using sax and vector space model”. In: *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE. 2013, pp. 1175–1180.

[9] Imran N Junejo, Khurrum Nazir Junejo, and Zaher Al Aghbari. “Silhouette-based human action recognition using SAX-Shapes”. In: *The Visual Computer* 30.3 (2014), pp. 259–269.

[10] Ryan Prescott Adams and David JC MacKay. “Bayesian online changepoint detection”. In: *arXiv preprint arXiv:0710.3742* (2007).

[11] Rudolf B Blazek et al. “A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point detection methods”. In: *Proceedings of IEEE systems, man and cybernetics information assurance workshop*. Citeseer. 2001, pp. 220–226.

[12] Alexander G Tartakovsky and Venugopal V Veeravalli. “Change-point detection in multichannel and distributed systems”. In: *Applied Sequential Methodologies: Real-World Examples with Data Analysis* 173 (2004), pp. 339–370.

[13] Shashwati Kasetty et al. “Real-time classification of streaming sensor data”. In: *Tools with Artificial Intelligence, 2008. ICTAI’08. 20th IEEE International Conference on*. Vol. 1. IEEE. 2008, pp. 149–156.

[14] Eamonn Keogh, Jessica Lin, and Ada Fu. “Hot sax: Efficiently finding the most unusual time series subsequence”. In: *Data mining, fifth IEEE international conference on*. IEEE. 2005, 8–pp.

[15] Pavel Senin et al. “Time series anomaly discovery with grammar-based compression.” In: *EDBT*. 2015, pp. 481–492.

[16] Craig G. Nevill-Manning and Ian H. Witten. “Identifying hierarchical strcture in sequences: A linear-time algorithm”. In: *J. Artif. Intell. Res.(JAIR)* 7 (1997), pp. 67–82.

[17] Neil Walkinshaw, Sheeva Afshan, and Phil McMinn. “Using compression algorithms to support the comprehension of program traces”. In: *Proceedings of the Eighth International Workshop on Dynamic Analysis*. ACM. 2010, pp. 8–13.

[18] Yongjing Lin et al. “Supporting efficient query processing on compressed XML files”. In: *Proceedings of the 2005 ACM symposium on Applied computing*. ACM. 2005, pp. 660–665.

[19] Erin Earl and Richard E Ladner. “Enhanced sequitur for finding structure in data”. In: *Data Compression Conference, 2003. Proceedings. DCC 2003*. IEEE. 2003, p. 425.

[20] Li Wei et al. “Assumption-Free Anomaly Detection in Time Series.” In: *SSDBM*. Vol. 5. 2005, pp. 237–242.

[21] MF Barnsley. *Fractals Everywhere, 1993.*

[22] Nitin Kumar et al. “Time-series Bitmaps: a Practical Visualization Tool for Working with Large Time Series Databases.” In: *SDM*. SIAM. 2005, pp. 531–535.

[23] Ugo Vespier, Siegfried Nijssen, and Arno Knobbe. “Mining characteristic multi-scale motifs in sensor-based time series”. In: *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM. 2013, pp. 2393–2398.

[24] Yuan Li and Jessica Lin. “Approximate variable-length time series motif discovery using grammar inference”. In: *Proceedings of the Tenth International Workshop on Multimedia Data Mining*. ACM. 2010, p. 10.

[25] Abdullah Mueen et al. “Exact Discovery of Time Series Motifs.” In: *SDM*. Vol. 9. SIAM. 2009, pp. 473–484.

[26] William Wilson, Phil Birkin, and Uwe Aickelin. “The motif tracking algorithm”. In: *International Journal of Automation and Computing* 5.1 (2008), pp. 32–44.

[27] Pedro G Ferreira et al. “Mining approximate motifs in time series”. In: *Discovery Science*. Springer. 2006, pp. 89–101.

Publisher
Complex System Digital Campus
UNITWIN UNESCO
http://unitwin-cs.org

ARK unitwin-cs.org/ark:/69427/02