SCHEDA: Lightweight euclidean-like heuristics for anomaly detection in periodic time series

Fabio Guigou, Pierre Collet, Pierre Parrend

SCHEDA: lightweight euclidean-like heuristics for anomaly detection in periodic time series

Fabio Guigou[a,b,c,*], Pierre Collet[b,c], Pierre Parrend[b,d,c]

[a] *IPLine, Caluire-et-Cuire, France*
[b] *ICube Laboratory, Université de Strasbourg, France*
[c] *Complex System Digital Campus (UNESCO Unitwin) http://unitwin-cs.org/*
[d] *ECAM Strasbourg-Europe, Schiltigheim, France*

Abstract

Detecting anomalies in time series in real time can be challenging, in particular when anomalies can manifest themselves at different time scales and need to be detected with minimal latency. The need for lightweight real-time algorithms has risen in the context of Cloud computing, where thousands of devices are monitored and deviations from normal behaviour must be detected to prevent incidents. However, this need has yet to be addressed in a way that actually scales to the size of today's network infrastructures.

Typically, time series generated by human activity often exhibit daily and weekly patterns creating long-term dependencies that are difficult to process. In such cases, the euclidean distance between subsequences of the time series, or euclidean anomaly score, can be a very effective tool to achieve good detection within constrained latency; however, this computation has a quadratic complexity and a computational footprint too high for any realistic application.

In this paper, we propose SCHEDA (Sampled Causal Heuristics for Euclidean Distance Approximation), a collection of three heuristics designed to approximate the euclidean anomaly score with a low computational footprint in time series with long-term dependencies. Our design goals are a low computational cost, the possibility of real-time operation and the absence of tuning parameters. We benchmark SCHEDA against ARIMA and the euclidean distance and show that in typical monitoring scenarios, it outperforms both at only a fraction of the computational cost.

Keywords: Time series, Anomaly detection, Real time, Cloud monitoring, Lightweight algorithms

*Corresponding author
Email addresses: fguigou@ipline.fr (Fabio Guigou), pierre.collet@unistra.fr (Pierre Collet), parrend@unistra.fr (Pierre Parrend)

1. Introduction

Anomaly detection in time series has been extensively studied across many disciplines [1, 2, 3, 4, 5]. The advent of Big Data and the systematic collection of metrics and events has provided the industry with an unprecedented volume of timestamped, unlabeled data to be mined for patterns, insights and anomalies.

Time series generated by human activity, in particular, often show very particular daily and weekly patterns that create long-term dependencies difficult to process for the usual time series analysis tools such as auto-regressive and spectral models. As anomalies can manifest themselves as sudden changes as well as modifications in an expected pattern over time, methods focusing on short-term prediction, such as auto-regressive models, are poorly suited, but simply subsampling the series to only retain the low-frequency components is also inadequate because it hides transient anomalies and delays detection.

In [5], the author defines three types of anomalies:

- Point anomaly: a single value that does not belong to the normal values of the series.
- Contextual anomaly: a single value that belongs to the normal values of the series, but is not expected given the values before and after it (*i.e.* the context in which it appears).
- Collective anomaly: a subsequence series that does not match the normal pattern of the series.

Measuring the euclidean distance between each pair of subsequences of a time series allows to detect all three types of anomalies. The resulting metric shall henceforth be referred to as *euclidean anomaly score*. However, due to the quadratic complexity of this approach, it becomes extremely expensive extremely fast and do not scale well when many time series running over long periods of time are involved. As an example, computing the euclidean anomaly score for a 1000 points time series takes 450 ms, while 10,000 points requires 13 seconds and 50.000 points 5 minutes.

In a Cloud or network monitoring scenario, a single server is expected to process tens of thousands of data points each minute. In this case, the quadratic complexity of the euclidean anomaly score generates a combinatorial explosion that requires approximate solutions instead.

To deal with scenarios where both long-term and short-term deviations need to be detected within a constrained latency, and a periodic pattern exists in the time series, we propose three algorithms under the umbrella term of Sampled Causal Heuristics for Euclidean Distance Approximation, or SCHEDA (italian for "file" or "card") that maintain most of the desirable properties of the euclidean anomaly score while drastically reducing the computational footprint. In particular, and contrary to spectral methods, it can detect changes in long-term (*i.e.* low-frequency) components without requiring a large analysis window. In particular, spectral methods have to make assumptions on the data –

namely that anomalies can appear in a specific lower dimensional space – while distance-based, or neighbour-based methods are purely data-driven [5].

The rest of this paper is organized as follows: Section 2 explicits the requirements for an anomaly detection algorithm that could be integrated in a Cloud monitoring system. Section 3 reviews the literature on the topics of anomaly detection and monitoring in light of these requirements. Section 4 presents the SCHEDA models. Section 5 explicits the methodology used to evaluate each algorithm and validate the requirements. Section 6 presents the results of the experiments, which are further discussed in Section 7. Section 8 concludes this work.

2. Requirements

Based on the issues and perspectives of network and Cloud monitoring [6, 7, 8, 9], the requirements for a working anomaly detection algorithm that would be fit in a monitoring system are:

- Low computational footprint: One of the main challenges of monitoring is scalability. Decentralized architectures and high-performance polling are the very foundation of modern supervision suites such as Shinken[1]. While commercial products do not incorporate anomaly detection, its inception should not compromise scalability. Therefore, a small CPU and memory footprint are extremely important. Typically, the time required to process a single data point should not exceed a few millisecond, so that multiple thousands of points can be processed each minute. This requirement limits the computational complexity of the model; ideally, processing a single data point should have a complexity in $O(1)$, or $O(N)$ with a very small multiplicative factor.

- Real-time: The purpose of this research is to provide complementary features to more traditional monitoring systems. These are typically real-time applications, in the sense that once a data point is acquired, it is immediately processed, as opposed to being buffered for later analysis. This requirement has no direct impact on the computational complexity, unless it becomes so high as to introduce a perceptible lag, *i.e.* of the same order as the sampling period.

- Configuration-free: As highlighted in [10], algorithms whose performance strongly depends on user-defined parameter tuning are inherently hard to scale. Since monitoring systems deal with tens to hundreds of thousands of different time series, parameter tuning is clearly not an option.

Importantly, the reason why we turned to distance-based metrics and not spectral window-based methods is the need to handle long-term dependencies

[1] http://www.shinken-monitoring.org

(in the order of thousands of points) without requiring an huge analysis window: distance-based methods consider an analysis window not by itself, but in relation to other windows; therefore, they can detect anomalies in low-frequency patterns rapidly.

3. Related work

In light of these requirements, we present a review of the literature on anomaly detection in time series along three axes: lightweight algorithms, real-time algorithms and configuration-free algorithms. A brief review of the field of Cloud and network monitoring follows.

3.1. Lightweight algorithms for anomaly detection

The need for computationally cheap anomaly detection algorithms gave rise to a number of methods based on symbolic representations of the time series, which drastically lower their dimensionality while allowing new classes of algorithms to be used.

3.1.1. Symbolic methods

The Symbolic Aggregate approXimation, or SAX, has been introduced by Eamonn Keogh and Jessica Lin in 2003 [11]. Since then, it has become the *de facto* standard for symbolic representation of time series. In this representation, a time series is z-normalized, *i.e.* normalized to a mean of 0 and a standard deviation of 1, then sliced into non-overlapping segments which are vector-quantized and assigned a single symbol, usually a letter. The process is illustrated in Figure 1. This representation, by heavily under-sampling and quantizing the time series, creates a compact representation retaining the essential features of the underlying data, while allowing to use string analysis algorithms to query it. One of the great strengths of SAX is that, though not the best fit in any possible situation, it always performs well in all benchmarks, while retaining a much lower computational cost with respect to other more complex, adaptive representations: in [12], SAX is benchmarked against ACA (Adaptive Cluster Approximation) and Persist, both of which are adaptive and more costly, and none of the three representations shows a clear superiority over the other two.

In most implementations, the whole time series is typically not converted into a single SAX word [13]. Instead, a sliding window is used to extract words and to process the time series at a local scale. However, this method assumes that the scale of observation is about the same as the size of a SAX *word*, *i.e.* a SAX-encoded subsequence of a time series. In [4], the authors use SAX to encode electrocardiogram time series, with a word size the same order of magnitude as the length of a single heartbeat.

One way to perform anomaly detection is to use SAX to speed up the search for anomalous subsequences. This is the HotSAX approach, developed in [14].

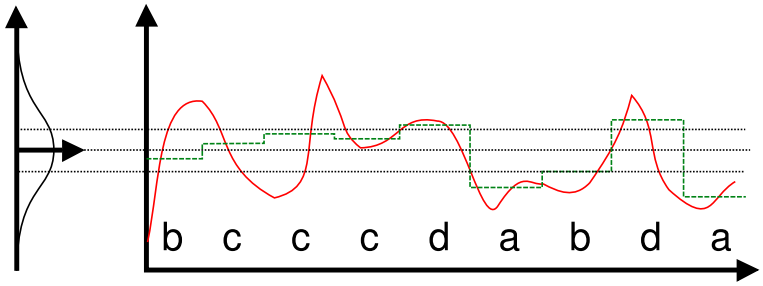


Figure 1: SAX encoding of a time series (red) into the word 'bcccdabda'. The piecewise approximation is shown in green and the normal distribution quartiles in black.

The complexity remains quadratic, using nested loops to compute all the distances between each possible pair of subsequences, but computations are pruned by early abandon heuristics in both loops, based on the SAX representation of the subsequences. Unfortunately, even with this speed-up, the computational cost remains too high for large volumes of time series, as the authors themselves acknowledge.

More efficient methods can be developed by using only the SAX representation. In [4], SAX subsequences are mapped to "bitmaps" (square matrices that be easily be represented by images) using the Chaos Game representation, initially designed for DNA sequences [15]. These bitmaps represent the number of occurrences of each possible n-letter sequence in a SAX word, n being typically small (3 or 4).

The anomaly score for any point is computed as the distance between the bitmap representing the left hand side of the point and the bitmap representing the right hand side. The lookback and lookahead windows are bounded, thus the complexity is linear. However, the anomaly computation does not take into account the whole series; only a neighbourhood is used. This may lead to a number of failure modes, as we show in the next section.

Another approach [16] uses Sequitur, a dictionary-based compression algorithm [17], to approximate the Kolmogorov complexity of the time series, *i.e.* the minimal number of bits required to represent it. Because Sequitur is dictionary-based, at any point in the series, it takes into account all the past information. This method is linear in time, causal, and generally robust, which makes it a good basis for online anomaly detection. It has, however, some known failure modes, which we develop in the next section.

In [18], the authors propose VizTree, a SAX-based method for visually exploring periodic time series and automatically detecting anomalies. A time series is converted into a tree based on its SAX representation, and successive trees representing different positions in the time series can be compared by building a diff-tree, *i.e.* a tree representing the differences between SAX-encoded time series. This method has been shown to yield good results on simple anomalies. It is not, however, a real-time algorithm, as its ability to detect structure mostly depends on the granularity of the underlying SAX representation: good quality

detection requires the size of a SAX word to match the length of a typical pattern, which can be a full day. A lag of one day is appropriate for trend detection or post-mortem analysis, but unsuitable for real-time anomaly detection.

3.1.2. Shortcomings

SAX-based methods are extremely convenient because of their speed and simplicity. However, none of them is perfect: they come with a number of trade-offs and failure modes. HotSAX [14] uses the SAX representation of a time series to prune computations when searching for the most anomalous subsequence in a series, in the sense of the euclidean anomaly score; however it will only find a *single* result. It is an exact algorithm: the subsequence it detects is the farthest from its closest match in the whole series. This precision comes at the cost of a quadratic complexity, and therefore a very long running time for long series.

The Chaos Game bitmap algorithm brings a classic trade-off in anomaly detection: precision versus lag. It can be extremely precise, but requires a long lookahead window, typically two or more occurrences of the normal pattern [4]. When the typical periodicity of the signal is a few seconds, a lag of less than a minute is perfectly acceptable. When dealing with daily, or even weekly periodicity, which is typical of resource usage in computing or electric power usage, which show peaks during working hours and flatlines during the nights and the week-ends [1], being informed of an anomaly a few *weeks* after it has happened is utterly useless. The problem of reducing the lag is that the algorithm becomes increasingly sensitive to periodicity and concept drift, and eventually classifies the pattern itself as an anomaly.

The SAX/Sequitur algorithm, on the other hand, deals well with periodic signals, low frequency variations, and does not require much lookahead (a new symbol is generally compressed after one or two others have been received). However, its fundamental assumption that anomalies are always less compressible than normal signal, though apparently sound, does not apply to all data. Examples can be found in [19]. A typical failure mode is depicted in Figure 2: when the CPU usage measured in the time series rises suddenly and becomes stuck at 100%, the ARIMA prediction error and euclidean anomaly score clearly show a peak at the beginning of the anomaly, while the SAX/Sequitur score drops immediately, thus signalling no anomaly at all.

It is worth noting that the shortcomings of both the Chaos Game representation and VizTree are also shared by spectral methods such as the Short-Term Fourier Transform: because they operate on each subsequence, or window, independantly, they require the window size to be larger than the largest pattern in the time series in order to detect deviations.

The euclidean score, by contrast, does not consider a subsequence as an independant entity, but rather quantifies a relation between a subsequence and the rest of the series. This alleviates the need for long windows: low-frequency patterns are implicitly taken into account because the distance between two subsequences separated by the exact period of a pattern is always small (or even null in the absence of noise, as can be seen in Figure 6). Meanwhile,

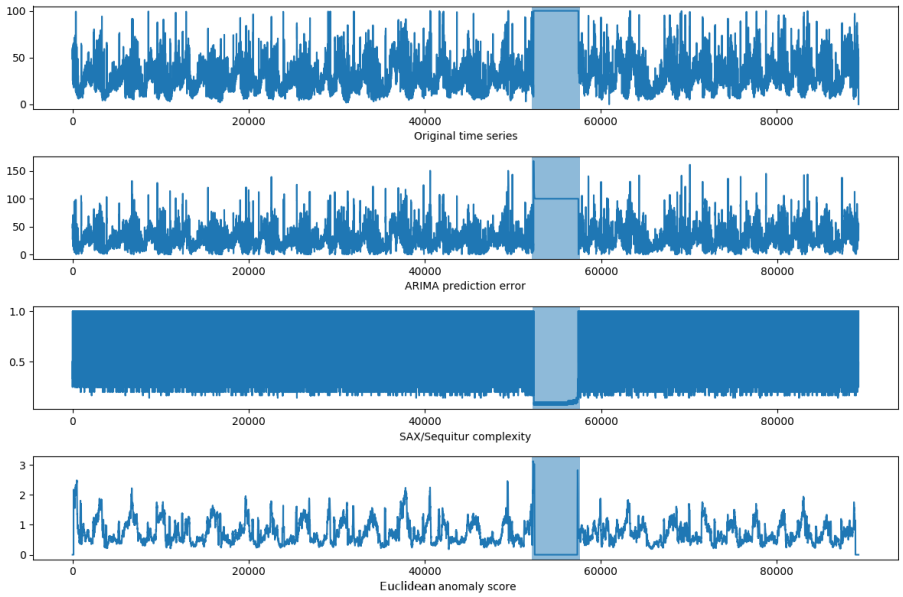


Figure 2: A comparison of the SAX/Sequitur algorithm, ARIMA and the euclidean score for anomaly detection in a fairly simple case

anomalies characterized by high-frequency events yield a high distance because they are only seen once in the series.

3.1.3. Exploiting periodicity

Information about the periodicity of a time series can be available offline, *e.g.* from domain knowledge, and considerably simplify the problem of anomaly detection. Using the known periodicity of a time series to reduce the computational cost of an analysis method is a very old technique; it was already used in the 1960s, *e.g.* to perform auto-regressive prediction of time series [20]. Short-time prediction and profiling in the energy sector has also successfully used the daily and weekly periodicity of the power demand [1]. This type of application is close to the requirements of cloud supervision: the signals have a known period, and analysis has to happen in real-time. However, short-term prediction and anomaly detection are not completely aligned, and the use of auto-regression with exogenous variables makes the method parameter-heavy. More importantly, long-term dependencies are hard to model, resulting in very high computational costs for auto-regressive models.

In the context of network monitoring, Burgess *et. al.* have developped approaches for adaptive computer systems [21, 22] that include time series anomaly detection. In particular, they model time as a cylinder, and univariate time series as two-dimensional, each point having a n coordinate modelling the period to which it belongs and a τ coordinate corresponding to its offset within a

period. Thus time is expressed as:

$$t = nP + \tau \tag{1}$$

This representation allows a rolling estimate of the mean and standard deviation to be computed at constant τ, thus modelling the variation interval of each point within a period. Typically, a period is chosen to represent the equivalent of a week. These estimates are updated in such way as to discount older samples and keep the model up-to-date with changes in the time series.

While this approach is robust and lightweight, it has to maintain a model (mean and standard deviation) of each time series over a full period. More importantly, it only exploits a *single* period in the time series. To deal with events happenning at shorter time scales, additional tests have to be used, such as the Leap-Detection Test (LDT) [23], which builds a *local* model of the time series and detects changes at the scale of about 10 time steps. Using two different models also means having two different, independent sources of alerting in a monitoring system.

We hypothesize that a model-free, distance-based system can reduce the storage space requirement while implicitely discounting older data. A distance-based, model-free nearest-neighbour search should also be capable of operating at any time scale without modification.

3.2. Real-time anomaly detection

Many online anomaly detection methods are not real-time, since they operate with a non-zero lag [4, 16]. The problem with that class of algorithm is that in general, the accuracy increases with the lag, *i.e.* a high lag is necessary to obtain a good detection rate.

The Hierarchical Temporal Memory (HTM) model [25] operates in real time and shows promising results, though the performance metric used to demonstrate it in [25] is an aggregate metric, and no direct measure of accuracy, true positive rate or false positive rate is given. However, its computational cost is high even on small data files. The reported average time to process a single data point is 11.3 ms, higher than all competing algorithms cited in [25] – with the exception of the Skyline model[2], which is unmaintained and has never been published; Skyline requires 414.2 ms of processing per data point. By contrast, the most efficient of the proposed SCHEDA heuristics has an average processing time per point of only 2.7 μs. In [26], a generic framework is proposed for real-time anomaly detection, addressing scalability and accuracy issues, but with little focus on the algorithms themselves. Once trained, recurrent neural networks such as long short-term memory networks can operate in real time [27], though the training phase has to be performed offline.

Statistical approaches based on autoregressive moving average (ARMA and its derivatives, ARIMA, SARIMA...) work completely online but require careful

[2] `https://github.com/etsy/skyline`

parameter tuning and do not take into account seasonality (*i.e.* what we refer to as "periodicity") [28, 29]. Typically, such methods are designed to work with a very low sampling rate (*e.g.* a point per month for revenue growth estimation) or with very low autocorrelation (*e.g.* financial time series), *i.e.* no long-term dependencies.

3.3. Configuration-free algorithms

The need for parameter-free clustering algorithms was already expressed in the early 2000s. In [10], the authors highlight the danger of incorrect tuning generating invalid results from which invalid conclusions would be drawn on the data. Instead, they propose that the local complexity of data, as approximated by compression methods, should be used. This work led to the development of more complexity-based clustering algorithms, such as SAX/Sequitur [16].

In [30], the parameter-free clustering algorithm CENTREx is proposed as an alternative to K-means, which requires the user to provide the number of clusters in advance, or DB-SCAN, which depends on a matching threshold. In [31], an Extreme Learning Machine (ELM) is used to determine the ideal number of clusters for image segmentation, effectively making the SLIC framework [32] parameter-free.

The first method has two advantages: it only computes the clusters once, as opposed to trying multiple different clusterings to find the optimum; and it does not require the additional cost of a neural network.

These methods, however, only handle clustering: anomaly – or outlier – detection has to be performed in a second step.

In [33], a fast nearest-neighbour search for time series is proposed. A considerable speed-up is obtained by using the fast Fourier transform to compute the dot products of any subsequence to all the series in a single operation. The algorithm is then extended to detect patterns and discords, *i.e.* anomalies, online. However, it maintains a quadratic complexity (even $O(N^2 logN)$ because of the Fourier transform); our tests required over a second to process a point, which is clearly way too high for our applications, where thousands of points are received and must be processed each minute. In fact, it was designed with the goal of maintaining a profile of a *single* time series for a very long time (*i.e.* multiple years), rather than processing multiple time series in parallel.

4. SCHEDA: Sampled Causal Heuristics for Euclidean Distance Approximation

We can finally present the models we derived from the requirements and the literature. We designed three algorithms to approximate the euclidean anomaly score, which we grouped under the name of SCHEDA, for Sampled Causal Heuristics for Euclidean Distance Approximation. "Sampled" refers to the fact that only a handful of distances are computed; "Causal" implied that the anomaly score for any given point is computed before the next point is sampled. This is a requirement for real-time processing. The three algorithms

are named Periodic Euclidean Sampling (PES), Random Euclidean Sampling (RES) and Sparse Euclidean Sampling (SES). They are presented in order of decreasing computational cost.

The choice of a distance-based approach is grounded on the following notions:

- Discounting of old data is not necessary, as changes of behaviour in a time series will naturally mean that the smallest distance will be found close to the considered subsequence and not in the distant past.
- The multiple time scales observed in monitoring time series are a difficulty when trying to build a model, but an opportunity with distance-based methods, as they provide candidates for the nearest neighbour.

In the following, the anomaly score will be denoted by A, the time series by T, its length by N and the size of a subsequence by n.

4.1. Overview

Figure 3 describes the process of computing the anomaly score as the acquisition of an image M, where each pixel corresponds to the distance between two subsequences, followed by the reduction of each column to a single value *via* the *min* operator:

$$M_{i,j} = \sqrt{\sum_{k=1}^{n}(T_{i+k} - T_{j+k})^2} \tag{2}$$

$$A_i = min_{j=1}^{N} M_{i,j} \tag{3}$$

This model of anomaly detection has a number of advantages: contrary to auto-regressive methods, it is global and long-term, and not limited to a short lag. In contrast with machine learning techniques such as the Hierarchical Temporal Memory [25], it is model-free and does not require training. Finally, because it is not based on a model, it does not require updating and discounting of old data as models such as Burgess' two-dimensional time [21] does. Its drawback is the computational cost of trying each possible subsequence to find the minimum of Equation 3.

The image described by Equation 2 is obviously symmetric, as $M_{i,j} = M_{j,i}$. This allows half of the distance computations to be discarded. The diagonal band defined by $|i-j| \leq n$ is also discarded as it defines an area of self matches, *i.e.* distances between a subsequence and a slightly shifted version of itself, which will always yield a distance of almost zero, irrelevant for anomaly detection [14].

The window size is a trade-off between context sensitivity and lag: the limit case of a very short window is a single point, which does not allow for any kind of structural anomaly detection as it captures *no* structure at all. A very long window, on the other hand, averages the effect of an anomaly over a long time, thus hiding short anomalies and introducing a longer detection lag. Empirically,

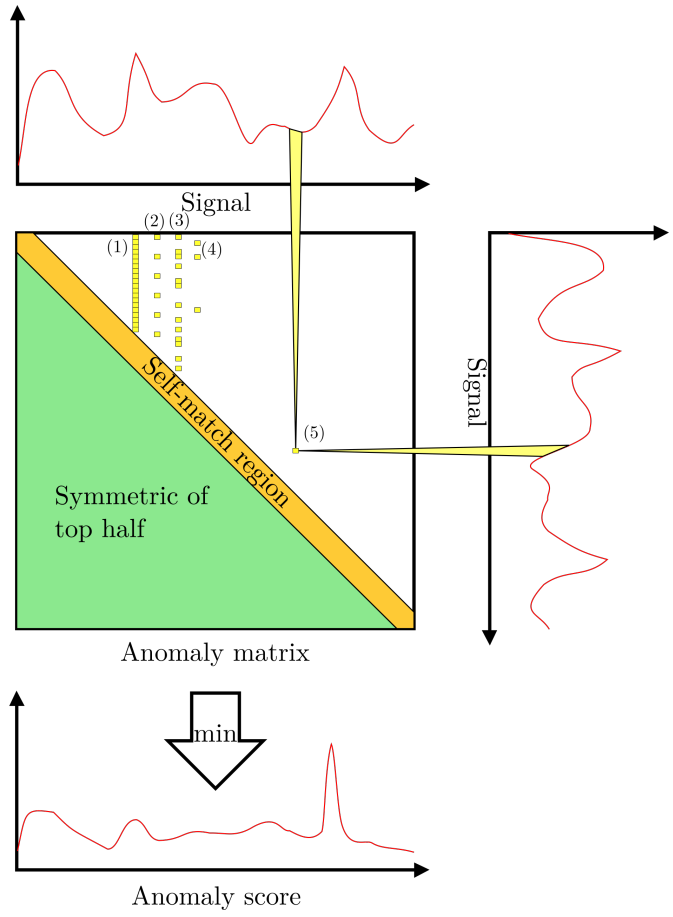


Figure 3: Anomaly detection as image sensing

we found the size of 120 points, or 2 hours of data, to be suitable for the use case of Cloud monitoring. It also reflects the time scale on which events happen in an IT system; as such, the ideal window size depends on context.

Each pixel of the image depicted in Figure 3 is defined as the distance between a subsequence of a time series (signal on top) and another subsequence of the same series (signal on right side). An example is provided by the `(5)` caption.

The pixels on the lower left half of the image are a mirror of the top right part, while the pixels in the top-left to bottom-right diagonal are not computed. The captions `(1)`, `(2)`, `(3)` and `(4)` illustrate columns acquired using respectively the euclidean anomaly score (all pixels acquired), Periodic Euclidean Sampling (1 pixel out of N is acquired), Random Euclidean Sampling (a fixed number of random samples is acquired) and Sparse Euclidean Sampling (a handful of pixels at specified offsets are acquired). In this particular example, Periodic

Euclidean Sampling acquires one complete *row* of the image out of 5 (vertical downsampling), while Sparse Euclidean Sampling acquires 3 rows.

The final anomaly score is obtained by taking the minimal computed value in each column.

4.2. Periodic Euclidean Sampling (PES)

The first SCHEDA heuristic we present is the Periodic Euclidean Sampling, or PES. The principle of PES is that the distances between a subsequence and two subsequences that are very close to each other (and therefore considerably overlapping) should be very close. As such, it may be enough to compute only one distance in five, or ten, and thus to reduce the runtime cost by an order of magnitude or more. It is computed as:

$$A_i = min_{l=1}^{\frac{i}{s}} \sqrt{\sum_{k=1}^{n} (T_{i+k} - T_{i-l*s+k})^2} \tag{4}$$

where s would be a pruning factor, *i.e.* a (potentially large) integer determining the sparsity of the estimation. It is worth noting that, while faster than the naive computation of all the distances, this method still has a quadratic complexity in $O(\frac{N}{2s})$.

4.3. Random Euclidean Sampling (RES)

The Random Euclidean Sampling heuristic computes a set of *random* distances; its only parameter is the number of distances to be computed. Because of its random nature, it can be used on any signal, without stationarity assumption. It is defined as:

$$A_i = min_{l=1}^{m} \sqrt{\sum_{k=1}^{n} (T_{i+k} - T_{i-R+k})^2} \tag{5}$$

where R is a random integer generated for each computation.

4.4. Sparse Euclidean Sampling (SES)

Finally, we propose Sparse Euclidean Sampling as a very efficient heuristic for time series of stationary periodicity. SES only samples a few distances at very specific lags, where the minimum euclidean distance is expected to be found. (our benchmarks show that 5 lags can be sufficient – see Section 6. It requires the periodicity to be known ahead of time (which is possible for many cases, *e.g.* for Cloud monitoring) or estimated using some other method. It is defined as:

$$A_i = min_{l=1}^{m} \sqrt{\sum_{k=1}^{n} (T_{i+k} - T_{i-L_l+k})^2} \tag{6}$$

where $L_1, L_2, ..., L_m$ are a set of predefined lags at which the distance is to be sampled. This is ideal in the case of a signal with a known and stationary periodicity: the minimal distance will always be between subsequences separated by an integer number of periods. Typically, signals related to human activity, such as power demand [1] or server resource (disk and CPU activity) display a daily and a weekly periodicity; therefore, sound values for $L_1, L_2, ...$ should correspond to a day, a week, and multiples thereof. For instance, with a sampling rate of one point per minute, a minimal lag vector could be $L = (1440, 10080)$: the distance is only computed with a lag of one day and one week. However, with these settings, anomalies risk generating an echo: the return to the baseline behaviour can be detected as an anomaly. Therefore, it is preferable to also add the double of each lag, so that $L = (1440, 2880, 10080, 20160)$. More lags can be added for increased precision.

These lags can be defined either by domain knowledge (which is the case in this research) or by frequency analysis of the signal using a Fourier transform or cepstral analysis [34]. The only assumption is that the periodicity of the signal remains constant, or that a change in frequency should be detected as an anomaly.

4.5. Matching threshold estimation

In [25], the anomaly threshold is established on the probability for a prediction error to come from the predicted model *vs.* the data. The best threshold is a probability of 10^{-5} which, for a normally distributed error, corresponds to about 5 standard deviations. Therefore, we might consider any value of the anomaly score superior to its mean by more than 5 standard deviations to indicate an anomaly.

However, the true standard deviation of the series cannot be known at runtime: only a running estimate can be used. Instead of a global, or true, mean and standard deviation, we can only rely on a *local* mean and deviation. Empirically, we found that a five-sigma rule works well when the true distribution is known, *i.e.* when it can be computed from the complete series, but leads to many false positives when applied on local estimations. Instead, we use an eight-sigma rule: we detect an anomaly if the score $A_i \geq \hat{\mu} + 8\hat{\sigma}$. This process is described in Algorithm 1. The lines beginning with `PES:` , `RES:` or `SES:` correspond to the parts specific to one or the other heuristic. The algorithm should be read by picking one of the three and ignoring the lines specific to the other two. For ease of reading, the heuristics have been colored differently (red for PES, olive for RES and blue for SES).

5. Evaluation

In this section, we present the evaluation methodology and the particular context (Cloud monitoring) for the SCHEDA algorithms. The goal of these evaluations is to compare the performance of the heuristics to each other and to other methods and to validate the requirements expressed in Section 2.

Algorithm 1 SCHEDA Anomaly Detection

procedure AnoDet(S, L)
input:
$S = [s_1, s_2, ..., s_N]$ ▷ real-valued time series
PES: $P : integer$ ▷ pruning factor
RES: $R : integer$ ▷ number of random samples
SES: $L = [l_1, l_2, ..., l_m]$ ▷ lags for nearest neighbour search
$w : integer$ ▷ window size for distance computation
$r : integer$ ▷ length of the mean/deviation estimation
algorithm:
$A := [a_1, a_2, ..., a_N] = [0, 0, ..., 0]$ ▷ anomaly score
PES: $i_0 := P + w$
RES: $i_0 := R + w$
SES: $i_0 := max(L) + w$
for $i := i_0$ **to** N **do**
PES: $a_i := min_{k=1}^{i/P} \sqrt{\sum_{j=0}^{w-1} (s_i - s_{i-kP})^2}$
RES: $r_1, r_2, ..., r_R = rand(w, i - w)$
RES: $a_i := min_{k=1}^{R} \sqrt{\sum_{j=0}^{w-1} (s_i - s_{i-r_k})^2}$
SES: $a_i := min_{k=1}^{m} \sqrt{\sum_{j=0}^{w-1} (s_i - s_{i-l_k})^2}$
if $i < i_0 + r$ **then**
next
$\hat{\mu} = mean(A_{i-r:i})$
$\hat{\sigma} = std(A_{i-r:i})$
if $a_i \geq \hat{\mu} + 8\hat{\sigma}$ **then**
signal_anomaly()

5.1. Use case: Cloud and network monitoring

In the world of IT infrastructure, "monitoring" can have different meanings for different communities. In particular, monitoring has spawned research communities in the fields of Cloud and network monitoring, with very different goals and methods.

Still, many common points are to be found between these two very different contexts. In [8], the authors acknowledge the importance of monitoring and the fact that, despite this very importance, the area is still under-researched. They define monitoring as a 3-step process:

- Collection: data is first polled from devices using standard tools: SNMP[3] or standard system utilities (`top` to monitor processes, `df` to monitor storage, `uptime` to detect reboots...)
- Analysis: from simplest to most complex:
  - Graphing: the collected data is inserted into a database, so that a history may be retrieved and viewed for debugging and post-mortem analysis.
  - Thresholding: alarms are automatically raised when a metric becomes too high (or too low).
  - Behavioural analysis: alarms are raised when a metric deviates from its normal behaviour. This is the current use case for SCHEDA.
- Decision: generally limited to alerting the user. Monitoring systems capable of automatic fault mitigation and correction in non-trivial contexts remain an open problem.

5.2. Data used

The data used to benchmark our algorithm is extracted from a supervision system deployed in a datacentre. It consists of 31 time series representing the CPU load, memory usage, number of concurrent sessions and other indicators measured on networking devices, some of which containing anomalies while others only reflect normal behaviour. Each series runs over 1 to 2 months, with a sampling rate of one point per minute.

This dataset has been partially labeled: each series is hand-annotated with regions that either contain anomalies (and where detections are expected) or are strictly normal (and detections are errors). A region is simply a starting point and an end point. An anomalous region may cover multiple individual anomalies and some normal behaviour, and parts of the series may not be covered by any region. Detection in such areas (where the observed behaviour cannot be clearly classified as normal or abnormal) is not taken into account in the final evaluation.

[3] Simple Network Management Protocol, often used to capture metrics such as CPU and memory usage, disk status, uptime...

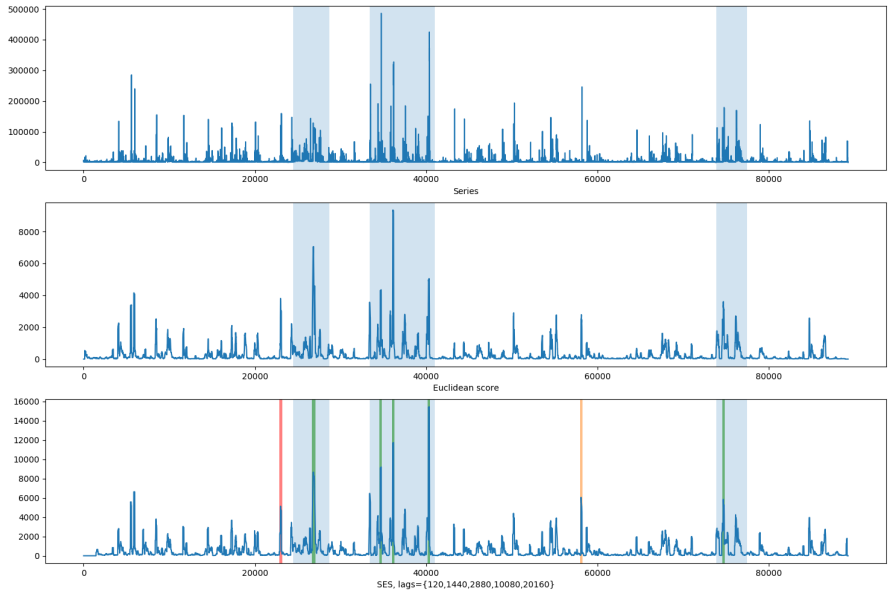


Figure 4: Example time series, euclidean anomaly score and SES score

A typical time series, with the associated anomaly score and detected anomalies, is depicted in Figure 4. This series (top graph) represents the CPU usage of a server measured during a period of two months, with one data point per minute. The anomalies are highlighted in clear blue. The graph in the middle shows the corresponding euclidean anomaly score, while the bottom one represents the approximation of the euclidean score by the SES heuristic, using 5 time lags. The highlights in green correspond to anomalies correctly detected, the red to false positives and the yellow to anomalies detected in an unlabelled area, *i.e.* that cannot be accurately classified by the expert. The graph shows 5 true positives, one false positive, one unlabelled detection and no false negative (anomalous region without any anomaly detected).

The periodicity of such time series, even if not visible at first glance, is made obvious by a simple autocorrelation plot. Figure 5 shows two time series, one of the CPU load of a server and the other of the session count of a firewall. The autocorrelation plots below show a clear periodicity of 1440 points, which corresponds to a day at a sampling rate of one point per minute.

5.3. Computational footprint

The total CPU and memory usage are recorded using the standard UNIX `time` command. In particular, the cumulated CPU time and peak resident memory are recorded for the process of:

- Opening and reading the time series

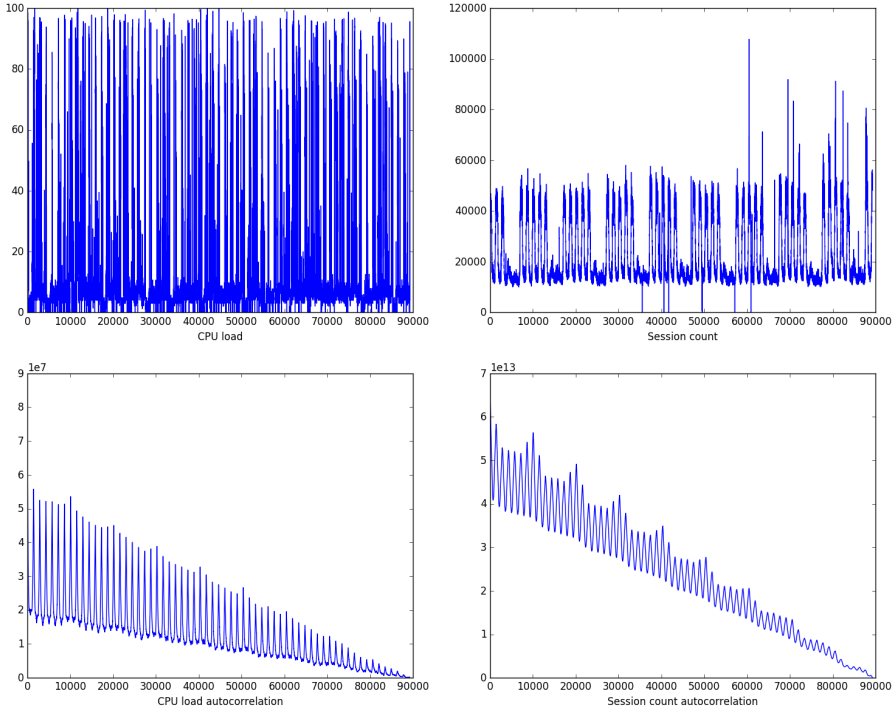


Figure 5: Two IT monitoring time series, with the associated autocorrelation plots

- Computing the anomaly score of each point
- Generating the boolean detection time series (1 if an anomaly is detected, 0 otherwise)

The CPU time is divided by the number of points in the series to yield a processing time per point.

From these two metrics, we can determine how many time series can be analyzed on a single server before it runs out of memory or becomes unable to process new data points as fast as they are collected.

5.4. Euclidean distance approximation

It is not trivial to find a metric to compare two time series. The intuitive euclidean distance does not provide the critical *shape* information: any transformation on the euclidean anomaly score that preserves *shape* is equivalent, in terms of anomaly detection potential, to the actual score. Most efforts in shape matching use dynamic programming and try to find an optimal alignment for the time series [35, 36]. This process is long, complex and not required in our case, since the times series we compare (the euclidean anomaly score and its approximation) overlap perfectly without time distortion.

Instead, we use two metrics from geometry and information theory, respectively: the dot product and the Kullback-Leibler divergence (KLD). The intuition behind this choice is the following: the dot product measures the degree to which two vectors are *aligned*. In the context of vectors representing time series, this means that relative highs and lows in both series match – if not in terms of value, at least in their temporal location.

The KLD evaluates the quality of a statistical distribution as a predictor for another. In this case, we use it to quantify how well the proposed heuristic approximates the real euclidean anomaly score. The KLD is computed between the statistical distribution histograms of the euclidean score and its approximation. This method has been used for clustering in [37].

To summarize, our metrics are the following, for two time series T_1 and T_2 of length N:

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{N} T_{1,i} - T_{2,i}}{N}} \tag{7}$$

$$T_1 \cdot T_2 = \frac{\sum_{i=1}^{N} T_{1,i} T_{2,i}}{||T_1||.||T_2||} \tag{8}$$

$$D_{KL}(T_1||T_2) = \sum_{j=1}^{m} H_{1,m} log \frac{H_{1,m}}{H_{2,m}} \tag{9}$$

where H_1 and H_2 are the m-bins value histograms of series T_1 and T_2.

These metrics are used not only for their absolute values but also to rank various tunings of the heuristic.

5.5. Anomaly detection

Unsupervised learning can be difficult to evaluate in the absence of a ground truth. When no previous information is available about the data distribution, only generic metrics can be applied, e.g. cluster compactness, distance between cluster members, or mutual entropy. While these metrics allow to compare different algorithms or different settings, they do not correlate well to the actual relevance of the clustering itself. These metrics are known as intrinsic metrics.

In order to correctly assess the quality of the anomaly detection with respect to a real use case, it is necessary to label at least part of the data. This allows for the computation of true and a false positive rates, which define a Receiver Operating Characteristic (ROC) curve, *i.e.* the plot of the true positive rate against the false positive rate. This curve represents the trade-off between false alerts and missed anomalies. The Area Under the (ROC) Curve (AUC) defines the cost of that trade-off, *i.e.* how many anomalies are missed at a given error rate (ideally none), or how many false detections occur if all the anomalies are caught (again, ideally none). We use this metric to assess the quality of our anomaly detection algorithm under various settings. A perfect detector would have an AUC of 1, while a random detector has an AUC of 0.5. These metrics are known as *extrinsic* metrics [38]. We use the formula from [39] to estimate it.

Since the ground truth, *i.e.* what is really an anomaly, is hard to define even for an expert, we do not label each point individually; instead, we define *anomalous* areas in which we expect anomalies to be detected and *normal* areas in which we expect no detection. Anomalies detected in normal areas are considered false positives, while the absence of detection in anomalous areas generates a false negative.

6. Results

This section presents the results of the evaluations of PES, RES, SES, the euclidean anomaly score and ARIMA. The quality of PES, RES and SES as approximations of the euclidean anomaly score is evaluated, as well as the performance of PES, RES, SES, the euclidean anomaly score and ARIMA for anomaly detection on real data, based on the ROC curves, false alarm and missed alarm rates. Where relevant, computational cost and quality as an approximation of the euclidean anomaly score are also measured.

6.1. Computational footprint

The CPU and RAM costs are measured on a specially crafted artificial series running for 3 years, *i.e.* about 1.5 million points. All algorithms are implemented in C and read the full series in memory before carrying out the computations. The results are written in a file and not kept in memory. The CPU used is a 12 core Intel Xeon clocked at 2.2 GHz. The euclidean score, PES and RES implementations use OpenMP for parallelism, while SES runs sequentially. The running times reported in Table 1 are both expressed in wall-clock

Method	Clock time	CPU time	RAM	CPU time/pt
Eucl. dist.	5 h 37 min.	67 hours	13.5 MB	166.4 ms
PES ($s = 100$)	212 s	42:15	13.2 MB	1.7 ms
RES ($s = 500$)	95 s	18:48	13.3 MB	$777\mu s$
RES ($s = 100$)	25 s	4:41	13.0 MB	$194\mu s$
SES (5 lags)	4 s	4 s	12.8 MB	$2.7\mu s$

Table 1: Runtime cost for all algorithms

Method	Max. monitors count
Euclidean distance	1446
PES ($s = 100$)	141,176
RES ($s = 500$)	308,880
RES ($s = 100$)	1,237,113
SES (5 lags)	88,888,888

Table 2: Maximal number of monitors per server

time and CPU time, the latter being the most relevant here as it quantifies the actual cost of the algorithm, while the wall-clock time only reflects the degree of parallelism of one particular implementation on one particular CPU. Typically, the wall-clock time can be reduced by adding more cores to the machine, while CPU time can only be reduced by algorithmic of implementation improvements.

The results in 1 quite clearly demonstrate the primary quality of SES: its low computational cost. While all methods use about the same amount of memory (13 MB), their CPU cost varies wildly, from 2.7 microseconds per data point for SES to almost a second for the euclidean distance. Keep in mind, however, that both the euclidean score and PES run in quadratic time, therefore the CPU time per point increases with the length of the series. Obviously, if any real life application were to use one of these metrics, the search range would have to be limited somewhat (*e.g.* only compute the distance with respect to the data of the previous N months).

To assess the relative costs of RES and SES, let us imagine a scenario where a server would be running each one and receiving one point per minute for each series it monitors. Then the maximal number of series that could be monitored is given by $n = k * 60/t$, t being the processing time per point and k the number of cores on the server. This disregards the time required to actually *acquire* this point, which, if not optimized, may well be by far the most expensive operation. The results of this simulation are given in Table 2 for a 4-core server.

However, the RAM would be more limiting than the CPU. Considering a server with 8 GB of memory, and disregarding the memory used by the operating system and various services running on the machine, only about 630 monitors could run in parallel.

While this figure is somewhat disappointing, it fails to account for the fact that in a real scenario, the executable file and shared libraries would only be

Method	Complexity	Dot	KLD	NRMSE
PES	$O(N^2/1024)$	0.93	0.024	1.31
PES	$O(N^2/1200)$	0.90	0.029	1.70
RES	$O(500N)$	0.97	0.012	0.63
RES	$O(50N)$	0.93	0.026	1.29
RES	$O(20N)$	0.89	0.039	1.81
SES	$O(3N)$	0.91	0.033	1.60
SES	$O(5N)$	0.92	0.029	1.40

Table 3: Complexity of various methods for equivalent approximation power

loaded *once*, and thus need to be factored out. Given that all methods are equivalent as far as RAM usage is concerned (about 13 MB), using series of different lengths, we ran multiple SES instances to build a linear model of the memory cost and found the initial cost of loading the executable and libraries to be 1.7 MB, with a linear cost of 7.7 bytes per point of the series.

For the exact or the pruned euclidean score as well as RES, the whole series, or at least a long history (months to years of data) need to be kept in memory, as the distance computations range across all the previous data. Assuming a full year of data is kept in memory, this allows an 8 GB server to run a little over 2,000 monitors.

However, SES can overcome this limitation by only maintaining a circular buffer of the same size as its longest lag. In this example, keeping two weeks of data instead of a year means that the same server could run over 55,000 monitors simultaneously. Note that this would only require 0.06% of CPU time.

6.2. Euclidean distance approximation

The quality of each approximation is measured by the dot product between the approximation and the exact euclidean score, their Kullback-Leibler divergence and the normalized root mean square error (closer to 0 is best for both, closer to 1 is best for the dot product).

The results in Table 3 show that both SES and RES perform well as euclidean score approximations. It is interesting to note that approximations computed by sampling every 30^{th} distance, 50 random distances or 5 well-chosen distances are about equivalent, whichever similarity metric (*i.e.* Kullback-Leibler divergence, dot product or mean square error) is considered.

These results have the following implications:

- A dot product close to 1 implies that the SES/RES series "follows" well the euclidean score shapewise: the peaks and valleys match and their height with respect to the noise floor are about the same. Preserving shape is key to anomaly detection, where the absolute value of the anomaly score is of little importance but its contrast, *e.g.* how well the peaks are distinct from the surrounding clutter or noise, is paramount.

Method	Complexity	AUC	FDR	MAR
Euclidean score	$O(N^2/2)$	0.69	0.14	0.41
ARIMA	$O(N)$	0.56	0.20	0.46
PES (60)	$O(N^2/120)$	0.74	0.06	0.41
RES (100)	$O(100N)$	0.71	0.07	0.38
SES (1440, 2880)	$O(2N)$	0.71	0.10	0.46

Table 4: Comparison of various euclidean approximations for anomaly detection on real data

- A low mean-square error means that a point in the approximation is close to its corresponding point in the reference series. This is a very generic measure to quantify the quality of prediction or compression algorithms. While hard to interpret by itself, its variations are more important than those of other metrics and illustrate well the convergence of RES and SES towards the true euclidean score as more distances are computed, and the divergence of PES when the pruning factor varies.
- A low Kullback-Leibler divergence indicates that the statistical distributions of the euclidean score and its approximations are close. This is another important aspect of anomaly detection: since the detection threshold is directly derived from the statistical distribution of the anomaly score, it should remain as close as possible to the euclidean score.

These three metrics indicate that RES and SES are good approximations of the euclidean score.

6.3. Anomaly detection

Table 4 summarizes the performance of each approximation for anomaly detection. The euclidean score is included as a reference. The metrics used are the area under the ROC curve (AUC), the false discovery rate (FDR), *i.e.* the proportion of false alarms, and the missed alarm rate (MAR), *i.e.* the proportions of anomalies that did not result in an alarm. The AUC is computed using the Pareto front, *i.e.* using the optimal parameters for any given false positive rate. The actual ROC curves are depicted in Figure 8. The false discovery rate and missed alarm rate are computed using the eight sigma rule and reflect the performance of SCHEDA with a fixed threshold.

RES and SES are qualitatively tested on artificial data, so as to demonstrate the upsides and downsides of each flavour. These results are shown in Figures 6 and 7. Figure 6 pictures a toy example (neither the series nor the anomaly are likely to be encountered in any realistic scenario) of a structural, or collective anomaly: one occurrence of a pattern is shifted in time with respect to the others. The pattern *in itself* is normal, as are the individual values, but it happens too early. In this case, the euclidean score is unable to detect anything, while SES readily picks up the anomaly. Note that such shifts are likely to happen twice a year in countries implementing daylight saving time: a pattern caused by human activity and expected to repeat every 24 hours will be shifted

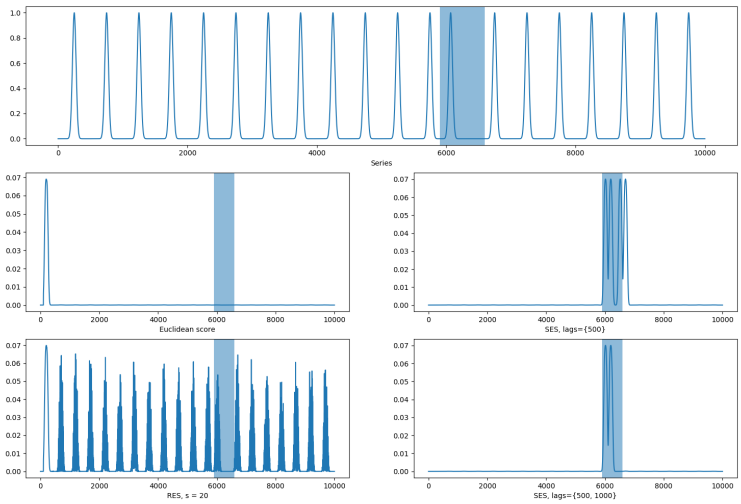


Figure 6: Dirac train-like series with structural anomaly

an hour because Monday, 9 a.m. will occur 47 (or 49) hours after Saturday, 9 a.m...

Figure 7 is a bit more realistic and contains four distinct anomalies that can represent real-life scenarios, namely:

- A high peak of activity during an active period (point anomaly) that could indicate a server overload or an attack
- A pattern of low activity in a normally active period (collective anomaly) that could indicate a network failure or simply depict the unexpected drop in activity during a holiday
- A very short peak of high activity in a normally active period (point anomaly) that could indicate an overload or an attack
- A short peak of activity in a normally inactive period (contextual anomaly) that could indicate a server failure or an attack

The results in Table 4 show that as far as anomaly detection is concerned, SES is considerably superior to euclidean distance, while RES remains close to euclidean and lags behind on the most important metric, the false discovery rate.

Interesting qualitative results can be drawn from Figures 6 and 7. The first thing we notice is the total inability of the euclidean score to detect the structural anomaly in the Dirac train (Figure 6). This is an effect of the window

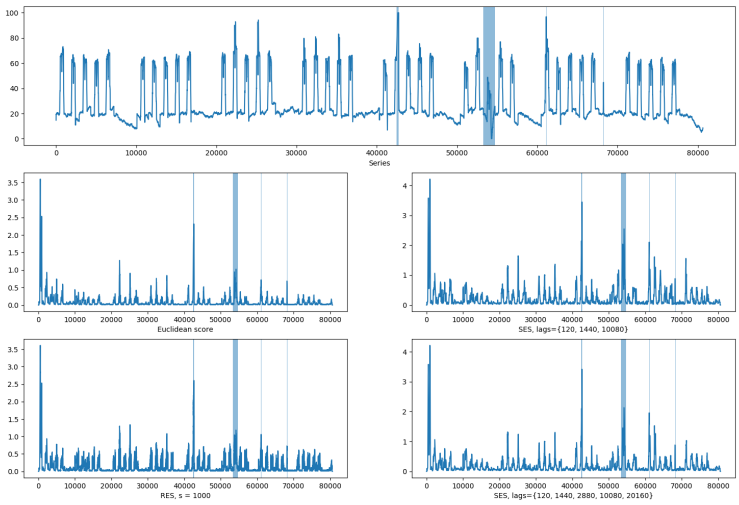


Figure 7: Human activity-like series with 4 typical anomalies. SES, RES, PES and the euclidean score are demonstrated.

size: the analysis window being smaller than the distance betweek two peaks, no single window contains the anomaly. This is an example of *structural* anomaly, where each single data point is normal in itself, but positioned in a place where it should not be. Longer windows, however, average out point anomalies, and therefore make short, transient errors harder to detect.

RES, as an approximation of the euclidean score, displays the same behaviour. Note that a median filter is applied to remove most of the noise: due to its random nature, RES tends to "jump around", with some points sampling good matches and others missing them. Both experiments show that it offers less contrast than the euclidean score: anomalies tend to be closer to the noise floor.

SES, on the other hand, has a higher contrast but suffers from the echo phenomenon we discussed in 4.1. If we consider the Dirac train experiment, we can readily observe the difference between the single-lag (top right) and the double-lag (bottom right) version: with a single lag, the return to a normal behaviour is detected as an anomaly. This can also be seen in the second experiment: out of four anomalies, the first two, which happen in the middle of a week, are correctly detected.

The last two on the other hand, taking place just after and during a weekend, where two of the reference lags (one day before and two days before) are much different from what is observed, are much more blurry and bleed further in time. The third anomaly, on a Monday, makes the Tuesday anomalous. The

last Monday, which shows a high anomaly score, is a very interesting case: of its 4 reference lags, 3 are useless: the previous 2 days are a week-end, which means a totally different statistical distribution, and the previous Monday contains an anomaly. Therefore, the anomaly score is estimated from a single reliable point, and is much higher than it should be. This, in turn, generates a false positive. It illustrates the most prominent failure mode of SES.

Figure 4 is given as an example of the typical shape of the time series and anomaly scores. Marks on the lowest time series correspond to detections, with green marks being real positives, red marks false positives and yellow marks detections in areas that have not been labeled as definitely normal or abnormal.

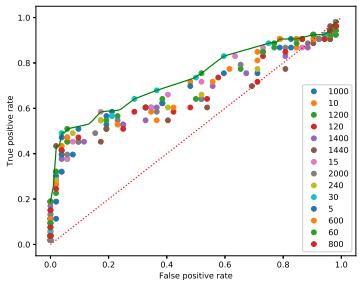
6.4. Failure modes

We mentioned symbolic algorithms in Section 3, and how they might be inappropriate for our applications. The results also indicate quite clearly that the prediction error of forecasting methods such as ARIMA is not a good indicator of structural anomalies. Figure 2 illustrates the failure modes of both of these algorithms, in particular how ARIMA fails to extract an obvious anomaly from the noise floor and how Sequitur detects it as the *least* anomalous part of the series. The anomaly being a plateau, it is easily compressible and has a very low entropy, defeating the basic assumption of using Sequitur for anomaly detection. The euclidean distance, by contrast, shows peaks at the location of the anomaly.

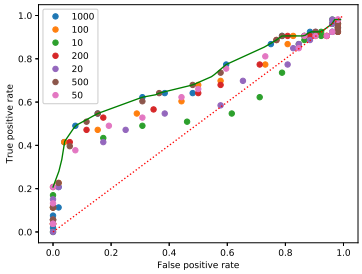
7. Discussion

In this section, we review the results of our experiments in the light of the requirements expressed in Section 2:

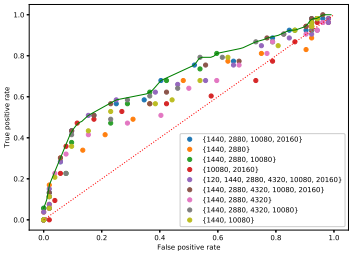
- Low computational footprint: Interestingly, the real limitation to the number of time series that can be processed is the available memory, and not the available CPU power. This can be mitigated by either using smaller float representations, compressing the data in memory or retrieving the relevant points from a time series database on disk. However, even without such optimizations, the SCHEDA heuristics, and especially SES, are scalable enough to be a valuable add-on to any monitoring system. The computational cost of SES is over 60,000 times lower than that of the euclidean anomaly score, yet its performance is superior when it comes to anomaly detection.
- Real-time: All three heuristics are real-time by design, as they process data points without buffering.
- Configuration-free: Even with a fixed detection threshold, all three algorithms perform correctly.



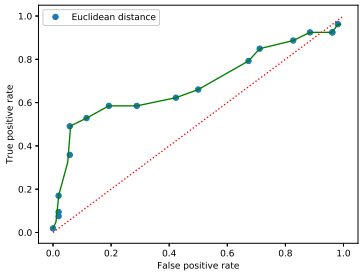
(a) PES with different pruning factors



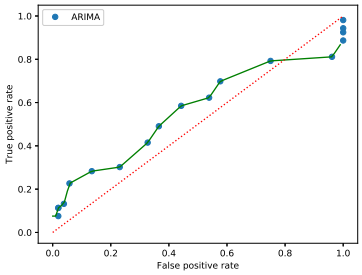
(b) RES with different sample counts



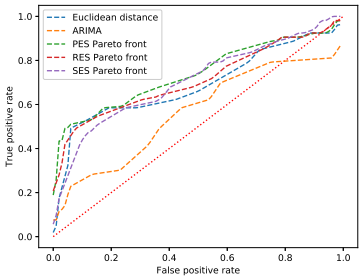
(c) SES with different lags



(d) Reference euclidean anomaly score



(e) ARIMA (1, 1, 1)



(f) Pareto front for SES, RES, PES, the euclidean score and ARIMA

Figure 8: ROC curves for PES, RES, SES, Euclidean score and ARIMA. For PES, RES and SES, the green line represents the Pareto front, *i.e.* the ROC curve using the optimal settings for each detection threshold

One interesting outcome of this experiment is that the best approximations of the euclidean anomaly score do *not* perform the best in anomaly detection. We also found that the euclidean anomaly score itself is outperformed by PES, RES and SES.

The ROC curves themselves are not typical of what is usually considered "good": an area barely above 0.7 is rarely considered a good outcome. Outlier detection in [40] typically exhibits AUC over 0.85. It is worth remembering though, that dealing with *temporal* data is harder than *static* data, as only partial information is available at any time.

A better way to understand these results is to look at the false discovery (or false alarm) and missed alarm rate. In Table 4, we can see that, for instance, RES, using the eight-sigma rule, the false alarm rate is 7%, thus one in 14 detections is a false alarm, while the missed alarm rate is 38%, meaning that nearly two thirds of the anomalies are detected. By comparison, SES, with the exact same AUC, has a 10% false alarm rate and a 46% missed alarm rate, both worse than RES.

Now we can put these results in the context of monitoring: the state of the industry is such that current monitoring systems are unable to detect anything but point anomalies; and, even then, generate more than 10% of false alarms[4]. Therefore, SCHEDA offers a significant improvement over traditional monitoring solutions, while keeping the operational cost (the false alarms sent to the support teams) acceptably low.

8. Conclusion and perspectives

We have presented three heuristics designed to approximate the euclidean anomaly score for time series anomaly detection. All three outperform the euclidean anomaly score for anomaly detection. In particular, we have shown that Sparse Euclidean Sampling (SES), a linear-time approximation of the euclidean score based on prior knowledge of the time series seasonality, outperforms the euclidean distance for a computational cost 60,000 times smaller (2.7 microseconds per point against 166 milliseconds, or 4 seconds for three years of data againt 67 hours). Random Euclidean Sampling (RES), though more expensive, also shows similar performance. Periodic Euclidean Sampling (PES) is the best performing heuristic but retains the quadratic complexity of the euclidean score.

SES and RES both validate the original requirements for Cloud monitoring: low computational cost, real-time analysis and absence of parameter tuning. Furthermore, all three methods have shown an ability to detect collective anomalies efficiently, even more that the euclidean score.

Despite the promising results demonstrated in this work, the proposed models can still be improved. In their current state, the SCHEDA heuristics work well on time series with a known, or at least guessable period, that could be extracted by peak detection or autocorrelation analysis. However, they could

[4]Personal experience of the author

potentially be extended to more unpredictable signals with only an *approximate* period, *e.g.* biological signals. This could be achieved using elastic distances such as Dynamic Time Warping (DTW, [41]) and longer time windows. The burden of aligning the subsequences would be shifted from the user (in the case of SES, by precisely choosing the lags) to the distance function itself. The computational overhead of using more complex distance metrics has been studied in [42]; the authors present many optimization techniques to manage this overhead. They even show that the DTW can be faster than the euclidean search.

In the context of Cloud monitoring, we have shown that a simple automatic threshold estimation allows tens of thousands of time series to be monitored in real time from a single server, without any user intervention, *i.e.* without specifying *any* parameter. It is our hope and belief that Sparse and Random Euclidean Sampling could be integrated into monitoring platforms to help operations and support teams anticipate failures by picking up the earliest signs of anomalies.

Acknowledgements

The work presented here has been funded by IPLine SAS, by the French ANRT in the frame of CIFRE contract 2015/0079, and by the French Banque Publique d'Investissement (BPI) under program FUI-AAP-19 in the frame of the HuMa project.

References

[1] M. Espinoza, C. Joye, R. Belmans, B. De Moor, Short-term load forecasting, profile identification, and customer segmentation: a methodology based on periodic time series, IEEE Transactions on Power Systems 20 (3) (2005) 1622–1630 (2005).

[2] D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, S. T. Edwards, Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data, arXiv preprint arXiv:1708.03665 (2017).

[3] A. Lavin, S. Ahmad, Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark, in: Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on, IEEE, 2015, pp. 38–44 (2015).

[4] L. Wei, N. Kumar, V. N. Lolla, E. J. Keogh, S. Lonardi, C. A. Ratanamahatana, Assumption-free anomaly detection in time series., in: SSDBM, Vol. 5, 2005, pp. 237–242 (2005).

[5] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (3) (2009) 15 (2009).

[6] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, V. Bhatnagar, An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art, Computing 97 (4) (2015) 357–377 (2015).

[7] S. R. Chowdhury, M. F. Bari, R. Ahmed, R. Boutaba, Payless: A low cost network monitoring framework for software defined networks, in: Network Operations and Management Symposium (NOMS), 2014 IEEE, IEEE, 2014, pp. 1–9 (2014).

[8] J. S. Ward, A. Barker, Observing the clouds: a survey and taxonomy of cloud monitoring, Journal of Cloud Computing 3 (1) (2014) 24 (2014).

[9] S. A. De Chaves, R. B. Uriarte, C. B. Westphall, Toward an architecture for monitoring private clouds, IEEE Communications Magazine 49 (12) (2011) 130–137 (2011).

[10] E. Keogh, S. Lonardi, C. A. Ratanamahatana, Towards parameter-free data mining, in: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2004, pp. 206–215 (2004).

[11] J. Lin, E. Keogh, S. Lonardi, B. 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, ACM, 2003, pp. 2–11 (2003).

[12] A. Sant'Anna, N. Wickström, Symbolization of time-series: An evaluation of sax, persist, and aca, in: Image and Signal Processing (CISP), 2011 4th International Congress on, Vol. 4, IEEE, 2011, pp. 2223–2228 (2011).

[13] J. Lin, E. Keogh, L. Wei, S. Lonardi, Experiencing sax: a novel symbolic representation of time series, Data Mining and knowledge discovery 15 (2) (2007) 107–144 (2007).

[14] E. Keogh, J. Lin, A. Fu, Hot sax: Efficiently finding the most unusual time series subsequence, in: Data mining, fifth IEEE international conference on, IEEE, 2005, pp. 8–pp (2005).

[15] M. F. Barnsley, Fractals everywhere, Academic press, 2014 (2014).

[16] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, S. Frankenstein, Time series anomaly discovery with grammar-based compression., in: EDBT, 2015, pp. 481–492 (2015).

[17] C. G. Nevill-Manning, I. H. Witten, Identifying hierarchical strcture in sequences: A linear-time algorithm, J. Artif. Intell. Res.(JAIR) 7 (1997) 67–82 (1997).

[18] J. Lin, E. Keogh, S. Lonardi, J. P. Lankford, D. M. Nystrom, Visually mining and monitoring massive time series, in: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2004, pp. 460–469 (2004).

[19] F. Guigou, P. Collet, P. Parrend, Anomaly detection and motif discovery in symbolic representations of time series, arXiv preprint arXiv:1704.05325 (2017).

[20] R. H. Jones, W. M. Brelsford, Time series with periodic structure, Biometrika 54 (3-4) (1967) 403–408 (1967).

[21] M. Burgess, Two dimensional time-series for anomaly detection and regulation in adaptive systems, in: International Workshop on Distributed Systems: Operations and Management, Springer, 2002, pp. 169–180 (2002).

[22] M. Burgess, Probabilistic anomaly detection in distributed computer networks, Science of Computer Programming 60 (1) (2006) 1–26 (2006).

[23] K. Begnum, M. Burgess, Improving anomaly detection event analysis using the eventrank algorithm, in: IFIP International Conference on Autonomous Infrastructure, Management and Security, Springer, 2007, pp. 145–155 (2007).

[24] J. Hawkins, S. Ahmad, D. Dubinsky, The science of anomaly detection, Tech. rep., Online technical report]. Redwood City, CA: Numenta, Inc. Available: http://numenta. com/# technology (2014).

[25] S. Ahmad, A. Lavin, S. Purdy, Z. Agha, Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017).

[26] N. Laptev, S. Amizadeh, I. Flint, Generic and scalable framework for automated time-series anomaly detection, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1939–1947 (2015).

[27] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, in: Proceedings, Presses universitaires de Louvain, 2015, p. 89 (2015).

[28] S. S. Pappas, L. Ekonomou, D. C. Karamousantas, G. Chatzarakis, S. Katsikas, P. Liatsis, Electricity demand loads modeling using autoregressive moving average (arma) models, Energy 33 (9) (2008) 1353–1360 (2008).

[29] H. Z. Moayedi, M. Masnadi-Shirazi, Arima model for network traffic prediction and anomaly detection, in: Information Technology, 2008. ITSim 2008. International Symposium on, Vol. 4, IEEE, 2008, pp. 1–6 (2008).

[30] D. Pastor, E. Dupraz, F.-X. Socheleau, Decentralized clustering based on robust estimation and hypothesis testing, arXiv preprint arXiv:1706.03537 (2017).

[31] F. Boemer, E. Ratner, A. Lendasse, Parameter-free image segmentation with slic, Neurocomputing (2017).

[32] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, Slic superpixels compared to state-of-the-art superpixel methods, IEEE transactions on pattern analysis and machine intelligence 34 (11) (2012) 2274–2282 (2012).

[33] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, E. Keogh, Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets, in: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, 2016, pp. 1317–1322 (2016).

[34] R. B. Randall, A history of cepstrum analysis and its application to mechanical problems, Mechanical Systems and Signal Processing (2016).

[35] M. D. Morse, J. M. Patel, An efficient and accurate method for evaluating time series similarity, in: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, ACM, 2007, pp. 569–580 (2007).

[36] T. Nakamura, K. Taki, H. Nomiya, K. Seki, K. Uehara, A shape-based similarity measure for time series data with ensemble learning, Pattern Analysis and Applications 16 (4) (2013) 535–548 (2013).

[37] T. Lee, Y. Xiao, X. Meng, D. Duling, Clustering time series based on forecast distributions using kullback-leibler divergence, Web, 2014 (2014).

[38] H. Aidos, R. P. Duin, A. L. Fred, The area under the roc curve as a criterion for clustering evaluation., in: ICPRAM, 2013, pp. 276–280 (2013).

[39] T. Katostaras, N. Katostara, Area of the roc curve when one point is available., in: ICIMTH, 2013, pp. 219–221 (2013).

[40] A. Lazarevic, V. Kumar, Feature bagging for outlier detection (2005).

[41] I. Oregi, A. Pérez, J. Del Ser, J. A. Lozano, On-line dynamic time warping for streaming time series, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2017, pp. 591–605 (2017).

[42] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Searching and mining trillions of time series subsequences under dynamic time warping, in: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2012, pp. 262–270 (2012).