UNIVERSITY OF CALIFORNIA
RIVERSIDE

A Novel Data Structure and Anomaly Detection for Time Series Data

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Sadaf Tafazoli

December 2023

ABSTRACT OF THE DISSERTATION

A Novel Data Structure and Anomaly Detection for Time Series Data

by

Sadaf Tafazoli

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2023
Dr. Eamonn Keogh, Chairperson

Time series data is one of the most extensively examined forms of data. According to a recent KDnuggets poll, 48% of analysts have engaged in time series data analysis within the past few years. This places time series data analysis second, surpassed only by relational (table) data analysis and preceding the analysis of text, images, spatial, and social network data [49]. The Matrix Profile is a data structure that enhances time series analysis by recording the Euclidean distance between each subsequence and its nearest neighbor. Leveraging the Matrix Profile, analysts can uncover numerous valuable insights about time series data, such as anomaly detection, chain discovery, motif discovery, and more. However, the Matrix Profile is limited to representing the relationship between the subsequence's *shapes*. It is known that, for some domains, useful information is conserved not in the subsequence's shapes, but in the subsequence's *features*. In recent years a new set of features for time series called catch22 has revolutionized feature-based mining of time series. Combining these two ideas seems to offer many possibilities for novel data mining applications, however, there are two difficulties in attempting this. A direct application of the Matrix Profile with the catch22 features would be prohibitively slow. Less obviously, as we will demonstrate, in almost all domains, using all twenty-two of the catch22 features produces poor results, and we must somehow select the subset appropriate for the domain. In this work we introduce novel data structure to solve both problems and demonstrate that, for most domains, the proposed C^{22}MP is a state-of-the-art anomaly detector.

Additionally, in this work, we illustrate how we can detect anomalies in multidimensional time series by framing the problem as K of N anomaly detection. The primary challenge in multidimensional time series anomaly detection appears to be that, in any N-dimensional time series, anomalies typically manifest themselves only in K of the time series, where $K < N$. This leads to a chicken-and-egg problem. If we knew which K time series exhibited the anomaly, it would be easy to discover its location. However, we do not know this in advance, and the search space is of size 2^N and not obviously amiable to greedy search. In this work we show a novel, simple algorithm that allows us to quickly find the best K of N anomaly subset for any value of K. Moreover, we show a simple metric that can rank the top anomaly subsets for all values of K from 1 to N. While our methods are mostly agnostic to the anomaly scoring model, for concreteness we use the Matrix Profile, and show that we can discover multi-dimensional anomalies that would escape detection by all current rival methods.

Table of Contents

1. Introduction

The Matrix Profile (MP) has emerged as one of the most promising general tools for time series data mining [74]. The MP is a data structure that annotates a time series by recording each subsequence's distance to its nearest neighbor. Typically, the z-normalized Euclidean distance is used, although a handful of other shape-based measures have been proposed. It has been shown that Matrix Profile allows for discovery of many regularities and structure in the original time series, including motifs, anomalies (discords), evolving patterns (chains), regimes, etc. However, the Matrix Profile has an important and underappreciated limitation: it is limited to representing the relationships between *shapes* of subsequences. For some domains useful information is conserved not in the subsequence's *shapes*, but in the subsequence's *features*. In this thesis, we introduce a novel data structure, expounded upon in Chapter 1, that adeptly captures both subsequence *shapes* and *features*.

In addition, there has been significant progress in univariate time series anomaly detection. The state-of-the-art algorithms can detect anomalies that are so subtle that they would defy discovery by careful human inspection [66]. However, efforts to generalize these successes to multi-dimensional time series have not seen similar progress. In this thesis, Chapter 2 presents our innovative algorithm designed for the detection of anomalies within multi-dimensional time series. Subsequently, we provide detailed insights into each method and elaborate on the motivations driving each project.

1.1 A novel data structure for time series data

In parallel to the explosion of interest in the MP there has been a significant breakthrough in time series feature extraction. The issue with feature extraction for time series was its ad-hoc nature. There are several thousands of named time series features in the literature, naturally including many subsets that are highly redundant. Thus, any practitioner hoping to use a feature-based approach was required to find a set of non-redundant features that were appropriate for both the domain and the task-at-hand. In [27] the authors provided a limited features set of twenty-two features, the eponymous *catch22* (C22), to make this task easier. These twenty-two features were discovered by an incredibly ambitious brute-force search through a massive set of 4,791 candidate features, evaluated on an enormous and diverse set of 147,000 time series datasets. It is no exaggeration to claim that catch22 has revolutionized feature-based mining of time series. There is a real sense in which it has democratized and deskilled time series data analytics. For example, allowing biologists to study the motion of trees in the wind without the need of data mining specialists [27].

Note that the Matrix Profile and catch22 are complementary ideas. The MP allows us to reason about the relationships between subsequences. For example, "*These two subsequences are alike*" (motifs) or "*this subsequence is different to all others*" (discords). Whereas C22 simply summarizes each individual subsequence's properties in a useful and intuitive way. As such, combing these two ideas into a single framework seems to offer many possibilities for novel data mining applications. The utility of doing this is predicated on the assumption that there can be semantic structure in time series that is better conserved in *features* than in *shapes*. In fact, this *is* the case. In Fig 1, we demonstrate this by clustering exemplars from two classes of the 50words dataset [14].

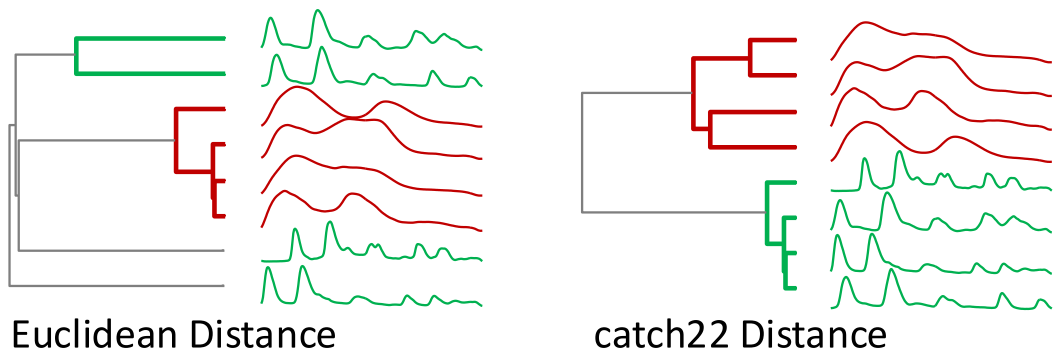


Fig 1 Eight instances from the 50words dataset clustered using Euclidean distance (*left*) and catch22 (*right*). At least for these two classes, catch22 is better able to represent the conserved class-specific structure.

However, there are two difficulties in attempting to unify these two ideas. A direct application of the Matrix Profile with the catch22 features would be intractable. The scalability of the SOTA Matrix Profile algorithm, SCRIMP[74], is due to its exploiting several unique properties of the Euclidean distance function. It does not appear that catch22 features are similarly amiable to such optimizations.

Less obviously, in almost all domains, using *all* twenty-two of the catch22 features will produce poor results. While some subset of the features will be sensitive to however an anomaly may reveal itself, it is very likely that there will also be some features that are very sensitive to irrelevant natural variations in the data, and the inclusion of these features will cause false alarms. In this work we introduce a novel representation and algorithm, C^{22}MP (Catch22 Matrix Profile) and ORR (Observed Repudiation of Regularity), to solve both problems.

1.2 Discovering Multi-Dimensional Time Series Anomalies with *K* of *N* Anomaly Detection

In the last few years, there has been significant progress in univariate time series anomaly detection. The state-of-the-art algorithms can detect anomalies that are so subtle that they would defy discovery by careful human inspection [66]. However, efforts to generalize these successes to multi-dimensional time series have not seen similar progress. The main problem appears to be that in any N-dimensional time series, the anomaly will generally only manifest itself on K of the time series with $K < N$ (typically $K \ll N$), and the inevitable small amounts of noise on the remaining N-K dimensions will tend to swamp the signal provided by the anomalous time series.

To make this concrete, consider the following simple example. We created a dataset that contains N time series, all of which are simply slightly noisy sine waves. As shown in Fig 2, we modeled a fault which induced a spike in one time series, and an unusual shape in the other, and did not affect the rest of the data. Can we detect this anomalous region?

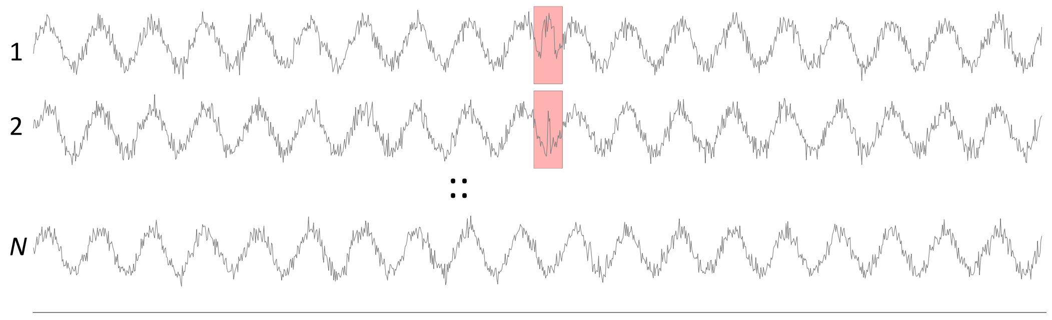


Fig 2 A set of N time series, which are slightly noisy sine waves. At one location, we have embedded anomalies in time series 1 and 2.

Given the success of anomaly scoring algorithms for the univariate case, the obvious solution to discover the anomalous region is to simply compute the anomaly scores for each time series individually, then combine those scores, in this case by adding them (although as we shall see, there are other ways). As shown in Fig 3. bottom, if $N = 2$, that is, we only consider these two-time series, then the problem is trivial.

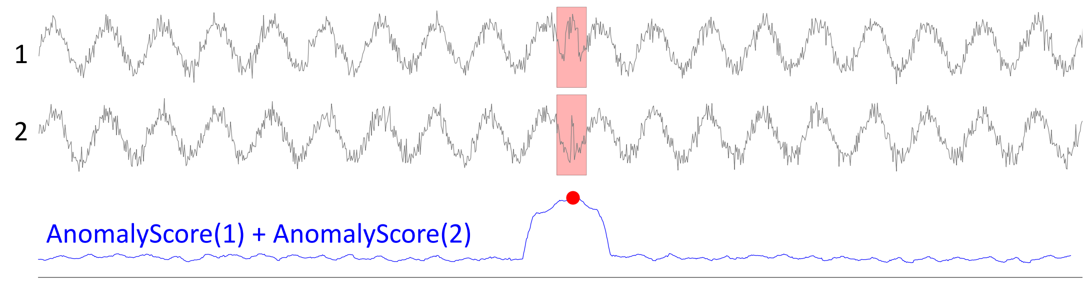


Fig 3 If $K = N$, then by simply summing the individual time series anomaly scores and finding the maximum (indicated by the red dot) we correctly predict the true anomalous region.

However, suppose that, in addition to the two above time series, we consider the others that do not contain any anomaly. Would the summed anomaly curve still reveal the correct location of the anomaly? In Fig 4 we test this question.

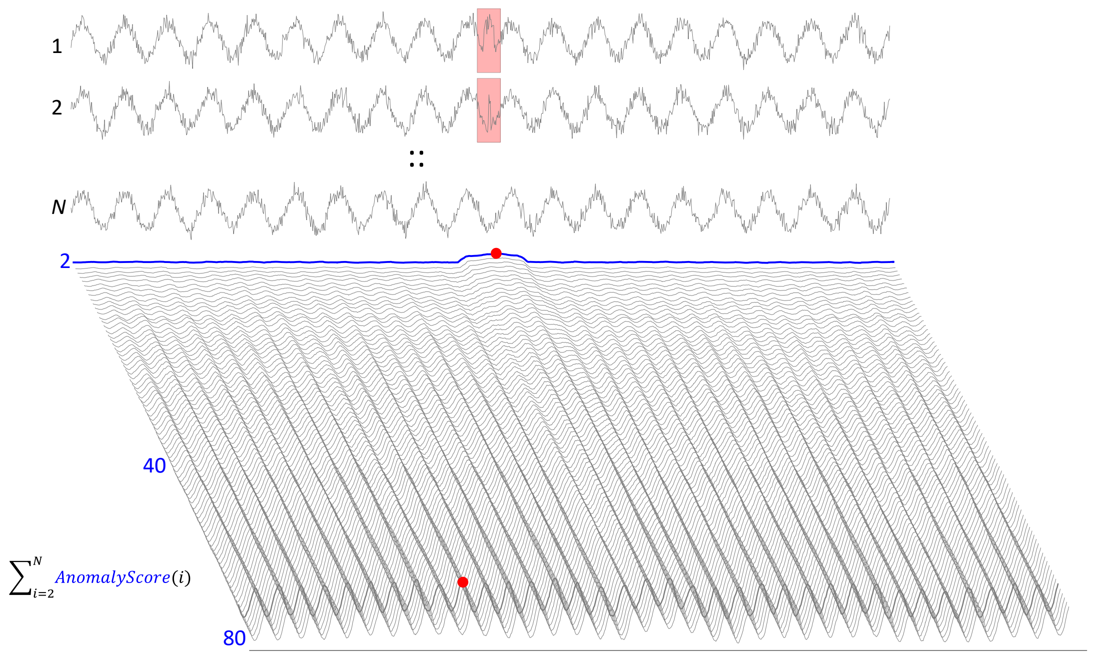


Fig 4 *top*) The time series we encountered in Fig 2 . *bottom*) The anomaly scores for increasing large sets of time series, that include the two anomalous examples. Note that by the time we have seen 75 examples, the predicted location of the anomaly has moved, and is now a false positive.

If we have, say, $N = 40$, the anomaly curve has flattened out somewhat but we can still just about detect a peak at the right place. However, by the time we have $N = 75$, the anomaly location has changed to the wrong location by the cumulative effect of small amounts of noise in the *N*-*K* normal time series.

The fact that this approach does not fail until *N* is 75 is a testament to the robustness of the underlying anomaly scoring function, here we are using the Matrix Profile [70]. However, this is a very contrived problem with short time series. For longer time series, or for real-world time series, the swamping effect will occur much earlier. In this work, we introduce a novel method to address these challenges.

2. C^{22}MP: The Marriage of catch22 and the Matrix Profile creates a Fast, Efficient and Interpretable Anomaly Detector

In this chapter, we introduce a novel representation and algorithm, C^{22}MP (Catch22 Matrix Profile) and ORR (Observed Repudiation of Regularity), designed to detect anomalies that may be preserved in either the shape or features of subsequences. We will demonstrate that our algorithm is orders of magnitude faster than a direct application of C22 to long time series, and we will introduce a framework to allow us to discover an appropriate subset of features for the domain. While the C^{22}MP can be applied to diverse problems, in this work we will confine our attention to just anomaly detection (*discord discovery* in the language of the MP [74]), reserving further generalizations to future work. We will demonstrate that the C^{22}MP can find subtle anomalies that no other SOTA algorithm can discover.

This chapter is organized as follows: in Section 2.1 we motivate the need for a feature-based anomaly detector before discussing related work in Section 2.2. Section 2.3 introduces the necessary definitions and notations. In Section 2.4 we first introduce algorithms to compute the C^{22}MP without regard to the weights, then we introduce four strategies to set the weights under different assumptions/conditions. Finally, in Section 2.5, we conduct an extensive empirical evaluation before offering conclusions in Section 2.6.

2.1 Motivation

With hundreds of proposed algorithms for Time Series Anomaly Detection (TSAD), it is reasonable to question the need for yet another one. Most of the existing TSAD algorithms found in literature only focus on the *shapes* of subsequences. This is evident in approaches such as Matrix Profile and related techniques[39][74], and appears to be the case for most deep learning approaches as well[27]. (Although many deep learning methods are opaque, even to the researchers that introduced them). Reasoning about changes in shape clearly works very well for some data types: gait, heartbeats, gestures, daily traffic patterns, etc. But many datasets may have richer and more complex behaviors, such as the insect behavior shown in Fig 5.

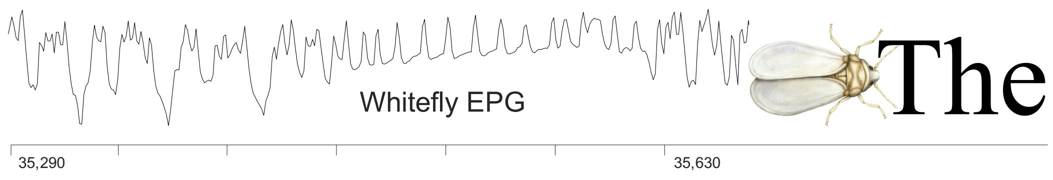


Fig 5 *left*) A snippet of the data used in the experiments in Fig 6. *right*) A whitefly (*Bemisia tabaci*) shown approximately to scale with the font used in the main body of this paper.

The insect data is clearly not random but the conservation of behavior is much more complex than simple repetition of shape, one could almost argue for a complex "grammar" of atomic behaviors.

The data is collected by gluing a tiny gold wire (about 1/100th the thickness of a human hair) to the insect and recording changes in electrical properties as it feeds on a plant. In Fig 6 we show a section of normal insect behavior that ends with a very obvious anomaly. A laboratory door was inadvertently opened, and the resulting breeze lifted the tethered insect off the plant.

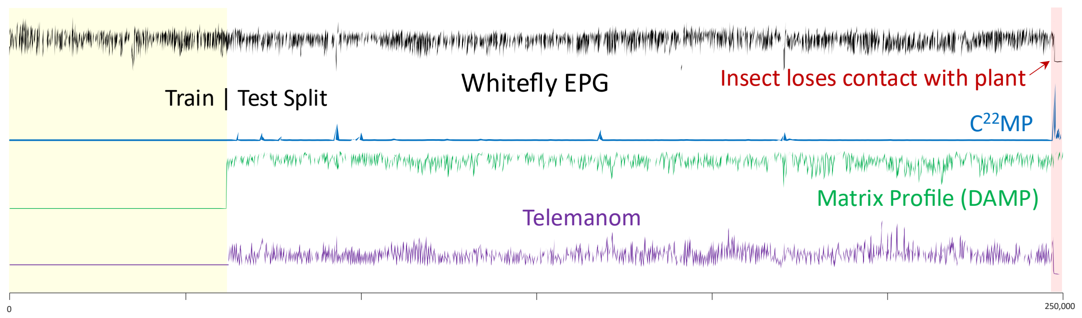


Fig 6 *top*) Forty minutes of normal insect behavior that ends with a visually obvious anomaly. *bottom*) Three TSAD algorithms, including the one proposed in this work, evaluated on this dataset. Only C^{22}MP correctly peaks at the right place (extreme right).

While the Matrix Profile [39][74] and Telemanom (a SOTA deep learning approach) [27] are unable to detect this anomaly, our proposed feature-based method, C^{22}MP, strongly peaks the moment it encounters the anomaly. As we will show in Section 2.5, there are many other domains/situations for which a feature-based approach is more suitable. It is important to note that we are not proposing feature-based methods could replace shape-based approaches, instead, we are suggesting that they can complement them.

On a side note, it worth providing a preview of some timing results here. Our proposed C^{22}MP took a total of 5.5 minutes, about 7.2 times faster than real time (Telemanom took a total of 14.6 minutes). This scalability is important, Wu & Keogh have argued that progress in TSAD algorithms has been hampered by the community's insistence on confining their attention to tiny datasets[1] [66][66].

[1] The two most cited datasets for evaluating TSAD algorithms are tiny: NY-Taxi (length 10,320) and Yahoo! Webscope (mean length 1,415) [66].

2.2 Background and Related Work

2.2.1 The Problem we are Solving

Prior to discussing related work, we wish to clarify exactly *what* problem we are attempting to solve here. A complete anomaly detection system can be described as consisting of two components:

1. Point to the *K* locations that are most likely to contain a subsequence that will be considered anomalous.
2. Make *K* decisions on whether to sound an alarm.

In certain TSAD systems, these two tasks are inseparably entwined. Nevertheless, we strongly believe that they should be completely divorced. The reason is that task ‘2’ can potentially avail of additional knowledge and context. For example, in some manufacturing scenarios, it is common to get false alarms at 12am, 8am and 4pm. These are caused by the minor disruptions of operator shift changes. If a user knows this, she can have a higher threshold for anomalies around these times. Moreover, the binary nature of task ‘2’ is often a false dichotomy. Instead of triggering a single possible action, i.e. (`red-alert`), in practice we may want a hierarchical response (`beep`, `log`, $\texttt{alert}_{\texttt{manager}}$, $\texttt{alert}_{\texttt{engineer}}$,…, `red-alert`). Finally, echoing the caution expressed by Wu and Keogh [66] and other recent papers, if we are not careful evaluating the results of task ‘2’ only can yield a highly deceptive notion of our actual performance.

With that context, here we are solving task ‘1’, pointing to the *K* locations most likely to contain an anomaly. Note that this ability can be used by other downstream algorithms such as attention focusing and summarization.

2.2.2 Related Work

Anomaly detection methods vary depending on the type of data being analyzed. In the case of non-temporal data, such as spatial data, anomalies can be detected by measuring the deviation of abnormal points from the rest of the data or by clustering the dataset and identifying points in less dense regions as anomalies. Spatial data is assumed to consist of independent data points. On the other hand, time-series data is different because the data points are not completely independent, and the latest data points in the sequence influence their following timestamps. Consider the following example to understand the limitations of point-wise analysis in detecting anomalies. Let us say we have a time series that records the temperature of an equipment, with values as follows: 40, 44, 41, 44, 42, 43, 44, 44, 95, 97, 96, 97, 98, 98, 96, 99. If we analyze these values independently, most anomaly detection methods would fail to detect the significant step change from 44 to 95 in the data. Instead, they would simply detect two equally distributed clusters. However, by incorporating the temporal dependencies of samples in a time series, anomaly detection models can capture the context and *shape* of regular patterns in the data and identify any anomalies. As a result, many techniques define anomalies as the subsequences that are maximally far from their nearest neighbor(s), or the time series *discord*[39][68][74].

For instance, in [39] the Discord Aware Matrix Profile (DAMP) is introduced. This fast throughput algorithm (exceeding 300,000Hz) is simple and only requires a single parameter. Unlike Matrix Profile [74], DAMP is not confused by repeated anomalies (twin-freaks) and can handle concept drift in time series. However, this algorithm is limited to detecting shape-based anomalies only.

There are several other machine learning methods[30][42][67], which try to detect shape-based anomalies using clustering methods on time series subsequence. For instance, in [42], K-Means is used to cluster subsequences of time series extracted by a moving window. In these methods to detect anomalies, the desired number of clusters, k, is defined, and the K-Means algorithm is executed until convergence, producing k centroids. The distance between each subsequence and its nearest centroid is then computed to identify anomalies. Any subsequence that exceeds a predefined threshold from its nearest centroid is classified as an anomaly. In their work, Keogh and Lin [25] have demonstrated that using sub-sequences of time-series data for clustering algorithms is meaningless. They found that the cluster centers obtained through multiple runs of the K-means algorithm on the same dataset are no more similar to each other than those of a random walk dataset. This means that the resulting centroids of a dataset could be those of a random walk, and it would be impossible to differentiate between them. Keogh and Lin also tried other algorithms such as hierarchical clustering, but they received the same results. They tested various distance measures, including Manhattan, L-infinity, and Mahalanobis distance, and applied K-means with k=3 and w=128 on the CBF dataset. They found that the resulting centroids were sinus waves that were completely different from the instances in the CBF dataset. Many authors have attempted to analyze this behavior mathematically, and others have tried to solve this problem, or at least show time series patterns that would work with Subsequence time-series Clustering (STSC). However, these problems generally remain unsolved.

In contrast to such <u>*shape*</u>-based methods, deep learning approaches typically "*model complex nonlinear <u>feature</u> interactions*" [1][27][44]. Although these models can identify feature-based anomalies, the features themselves are generally opaque to the user, and overfitting remains a significant challenge. Additionally, these models involve a substantial number of hyperparameters that must be managed, including:

- The number of hidden layers in the network (depth of the network)
- The number of nodes in each layer (width of the network)
- The length of the window,
- The learning rate
- The optimization function

If Convolutional Neural Networks are used, another set of parameters must also be handled, such as:

- The architecture of the CNN, such as using Batch Normalization, Dropout, or Max Pooling layers
- The number of kernels in each convolution layer
- The size of the kernel
- The depth of the Convolution Layer

Given the complexity and possibility of overfitting (especially when the number of samples in the data is limited), we adopt a more straightforward approach to detect feature-based anomalies.

Our work exploits the success of the catch22 feature set for time series [1][39]. This set of features was discovered through a brute-force search through thousands of candidate features, averaged over hundreds of datasets from diverse domains, using classification accuracy as an objective function. These features are designed to be a complete and (largely) independent universal feature set. Since its introduction there has been a flurry of activity in using catch22 for tasks such as time series classification and clustering. However, to the best of our knowledge, there are only a handful of efforts to exploit catch22 for anomaly detection [1]. In every case, they simply use catch22 features as inputs to deep learning algorithms. However, this means that we inherit the complexity, opaqueness, and sloth of deep learning algorithms. In contrast, we propose to directly use the catch22 features to produce a simple, transparent and fast algorithm.

The field of TSAD (Time Series Anomaly Detection) is highly active [27][39][66][74], with numerous methods proposed [27][51][52][64]. However, reviewing all of these different techniques is beyond the scope of this paper. Interested readers are directed to [66] and its references for a more comprehensive review.

2.3 Definition and Notation

Below we introduce all the necessary notations and definitions to frame our contributions. We begin by defining the key terms used in this work. The data we are addressing is *time series*.

Definition 1: A *time series* T is a sequence of real-valued numbers t_i: $T = [t_1, t_2, \dots, t_n]$ where *n* is the length of *T*.

Fig 7 demonstrates this notation on a running example dataset toy dataset we use as a running example. This dataset has a local distortion (in this case, *noise*) which *may* be an anomaly, depending on the context.

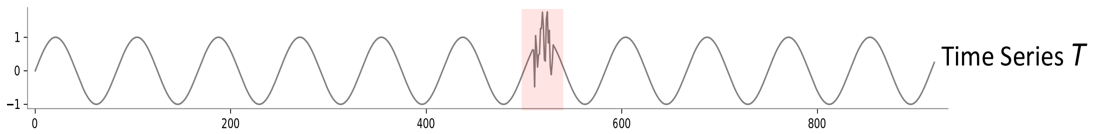


Fig 7 Our running example has a local burst of noise, which *may* be an anomaly, depending on the context.

We are interested in local *subsequences* of the times series.

Definition 2: A *subsequence* $T_{i,m}$ of a time series *T* is a continuous subset of data points from *T* of length m starting at position *i*. $T_{i,m} = [t_i, t_{i+1}, \ldots, t_{i+m-1}]$, $1 \leq i \leq n - m + 1$.

The length of the subsequence is typically set by the user based on domain knowledge. For example, for human or animal activities there is a cycle of a daily pattern so a subsequence of twenty-four hours (the circadian rhythm) might be appropriate.

In our proposed algorithm we need to calculate *all-pairs-similarity* of *all* subsequences of a given time series. The notion of all-subsequences set is purely for notational purposes. In our implementation, we do not actually extract the subsequences in this form as it would require significant time and space overhead. We define an *all-subsequences set* of a given time series as a set that contains all possible subsequences from the time series:

Definition 3: An *all-subsequences set A* of a time series *T* is an ordered set of all possible subsequences of *T* obtained by sliding a window of length *m* across T: $A = \{T_{1,m}, T_{2,m}, \ldots, T_{n-m+1,m}\}$, where m is the user defined subsequence length. We use $A[i]$ to denote $T_{i,m}$.

We consider two subsequences to be similar if they have similar features; more specifically if they have similar *catch22 features*. We calculate the catch22 features of each subsequence in the all-subsequences set and generate the *feature profiles*:

Definition 4: The *feature profiles* $FP \in \mathbb{R}^{(n-m+1)\times 22}$ is an array containing the catch22 features of all the subsequences in an all-subsequences set. We use $FP[i,:]$ to denote the subsequence $T_{i,m}'s$ features. Each column in the *FP* array corresponds to one of the twenty-two features' profile.

Fig 8 illustrates this concept, with each row representing the feature profile corresponding to one of the catch22 features computed with a sliding window of length m = 80. Note that:

- Some of the features (i.e., '1' and '2') seem to be invariant to the burst of noise.
- Some features *increase* in the presence of the distortion.
- Other features *decrease* in the presence of the distortion.
- Some features (i.e., '3' and '16') have variability that seems unconnected to the presence of the distortion.

The subset of features that are sensitive or invariant to a given distortion depends both on the type of distortion and the data itself. This observation correctly suggests that the choice of the subset of features to use will be critical for the success of any TSAD algorithm based on C22 features.

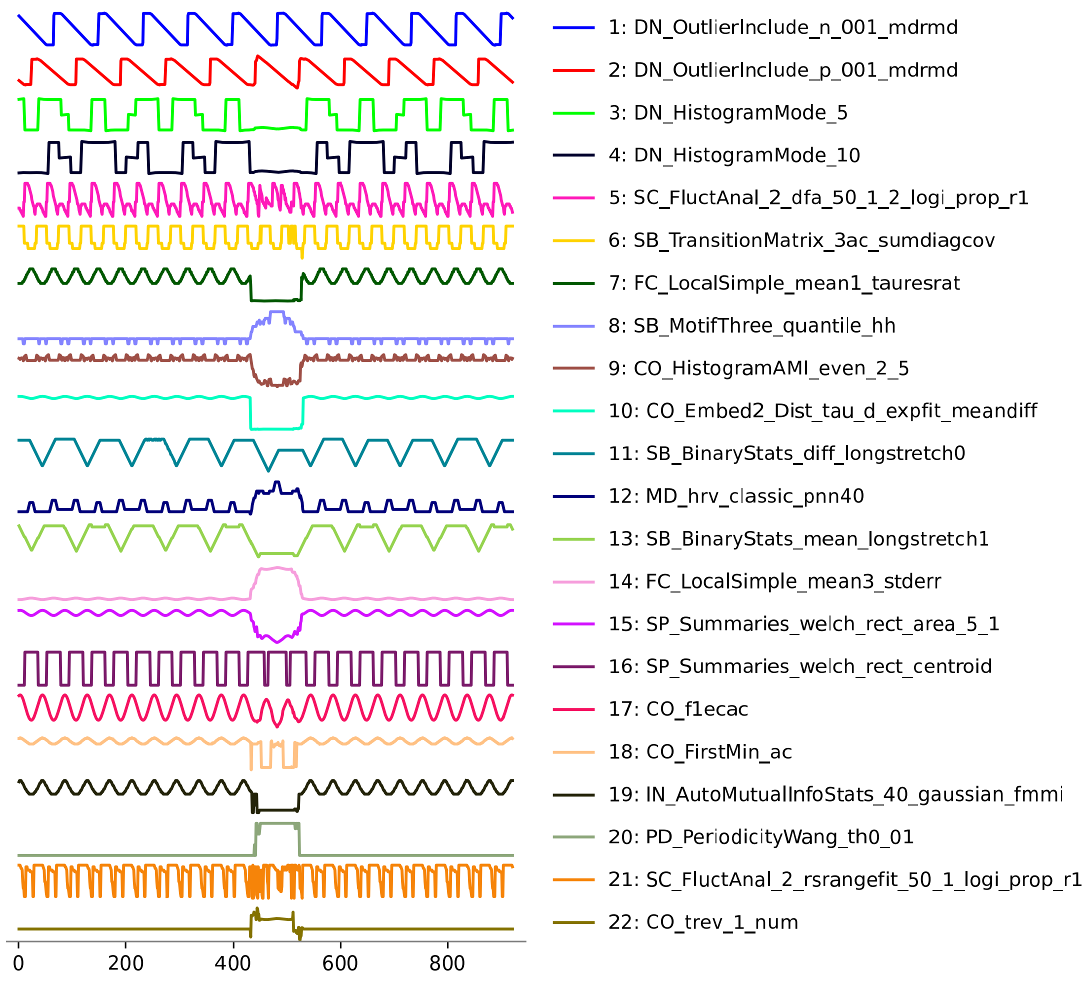


Fig 8 C22 Feature profiles for the running example introduced in Fig 7.

If we take any subsequence $T_{i,m}$ as a query, calculate its feature-based distance from all subsequences in the time series T and store the distances in an array in order, we obtain a *distance profile*.

Definition 5: *Distance profile* D_i for time series T refers to an ordered array of feature-based distances between the query subsequence $T_{i,m}$ and all subsequences of T. Formally, $D_i = d_{i,1}, d_{i,2}, \dots, d_{i,n-m+1}$,where $d_{i,j}$ $(1 \leq i, j \leq n - m + 1)$ is the feature-based distance between $T_{i,m}$ and $T_{j,m}$.

For the distance profile D_i of query $T_{i,m}$, the i^{th} position represents the feature-based distance between the query and itself, so the value must be 0. The values before and after position i are also close to 0, because the corresponding subsequences have overlap with query. Our algorithm neglects these matches of the query and itself, and instead focuses on *non-self match*.

Definition 6: *Non-self match*: Given a time series *T* containing a subsequence $T_{p,m}$ of length m starting at position p and a matching subsequence $T_{q,m}$ starting at q, $T_{p,m}$ is a *non-self match* to $T_{q,m}$ with distance $d_{p,q}$ if $|p - q| \geq m$. A non-self matching set M_p of $T_{p,m}$, contains all such non-self matching subsequences.

While the word *discord* has become something of a synonym for *anomaly* in recent years, for clarity, we reserve the word *discord* for its original meaning [39][74]. For our proposed variant, the subsequences with the *feature* vector whose distance to its nearest neighbor feature vector is maximum, we use *discordia* (the Latin origin of *discord*). We can now define *time series discordia.*

Definition 7: *Time series discordia*: Given a time series *T*, the subsequence $T_{r,m}$ of length *m* beginning at position *r* is said to be a *discordia* of *T* if the feature-based distance between $T_{r,m}$ and its nearest non-self match is maximum. That is, ∀ subsequences $T_{c,m}$ of *T*, non-self matching set M_r of $T_{r,m}$, and non-self matching set M_c of $T_{c,m}$, $min(d_{r,M_r}) > min(d_{c,M_c})$.

Intuitively the difference between time series discord and time series discordia is that a *discord* is a subsequence that is unique in its *shape*, and a discordia is a subsequence that is unique in its *features*.

Exploiting the closeness of the concepts of "discord" and "discordia," we adopt the idea of Matrix Profile to locate time series discordia. It has been shown that Matrix Profile is one of the most effective ways to find discords [74]. The Matrix Profile (MP) is a data structure that annotates a time series by recording each subsequence's *Z-normalized Euclidean distance* to its nearest neighbor.

Note that our framing of our contributions as a variant of the Matrix Profile is very deliberate. There is a large and growing body of literature on the MP and a diverse community that uses and extends the MP for diverse tasks [6][39][45][74]. Such users will find our contributions familiar and initiative and should be able to "plug" our algorithm into their frameworks with ease. We introduce the *Catch22 Matrix Profile* (C^{22}MP) a data structure that annotates a time series by recording each subsequence's *feature-based distance* to its nearest neighbor.

Definition 8: A C^{22}MP of a time series *T* is a vector storing the feature-based distance between each subsequence and its nearest non-self match. Formally, $C^{22}MP = [min(D_1), min(D_2), \dots, min(D_{n-m+1})]$, where D_i $(1 \leq i \leq n-m+1)$ is the distance profile of query $T_{i,m}$ in time series *T*. It is easy to see that the highest value of the C^{22}MP is the time series discordia.

As we will explain below, we can compute a special C^{22}MP which only looks to the past. We call it the *left*C^{22}MP.

Definition 9: A *left*C^{22}MP of a time series *T* is a vector that stores the feature-based distance between each subsequence and the nearest non-self match appearing before that subsequence. Formally, given a query subsequence $T_{i,m}$, let $D_i^L = d_{i,1}, d_{i,2}, \dots, d_{i,i-m+1}$ be a special distance profile that records only the distance between the query subsequence and all subsequences that occur before the query, then we have $C^{22}MP^L = [min(D_1^L), min(D_2^L), \dots, min(D_{n-m+1}^L)]$.

As mentioned earlier, in almost all domains, using *all* twenty-two of the catch22 features will produce poor results. To address this issue we can use a weight vector that indicates the importance of each feature for the given domain:

Definition 10: A weight vector, *w* is a list representing the importance of each feature, where the sum of the total weights is equal to one. Formally, $\sum_{i=1}^{22} w_i = 1, w_i \in (0,1)$.

Note that the weight vector can be real-valued. However, in many cases we may just want to use *f* equal-weight features, typically with $f \ll 22$. In this case we simply set the weight of each desired feature to 1/*f*.

In Section 2.4.3, we will present several different methods to set these weights, depending on the user's access to labeled data, domain expertise or ability to obtain feedback from a user.

2.4 The C^{22}MP

With the essential notations and definitions established, we are now in a position to explain how to calculate and accelerate the computation of the C^{22}MP, and how to set the feature weights.

*2.4.1 The left*C^{22}*MP*

In Table 1, we outline a brute force algorithm to compute the *left*C^{22}MP, assuming the feature weight vector *w* has already been determined and the user has chosen *m*. For clarity we only show the top-1 case. The generalization to the top-*K* case is trivial.

In line 1 we use the subroutine in Table 2 to obtain the feature profiles of the time series *T*. We scan all possible subsequences in test data in line 4. For each subsequence, we search to the left for its nearest neighbor. To consider the worst-case scenario, in line 5, we initialize the candidate subsequence to its leftmost nearest neighbor to infinity in line 5, we set the distance of the candidate subsequence to its leftmost nearest neighbor to infinity. In lines 6 to 10, we iterate through all the subsequences that are before the candidate subsequence to check which one of them has the lowest distance to the candidate subsequence.

Table 1 Brute Force Computation of *left*C^{22}MP.

```
Function: Brute_Force_Left_C22MP(T, m, w, s_idx)
Input:
  T: Time series
  m: Subsequence length
  w: Feature's importance weights
  s_idx: Location of split point between training and test data
Output:
  left_c22mp: leftC22MP

1    FP = get_feature_profies(T, m, w) //get feature profiles (def. 4)
2    left_c22mp = zeros((len(T)-m+1,))
3    //Scan all subsequences in test data and calculate left_c22mp (def. 9)
4    for i in range(s_idx, len(T) – m + 1):
5        lbsf = inf
6        for j in range(i-1-exclude_zone, -1, -1):
7            d = distance(FP[i], FP[j])
8            if d < lbsf:
9                lbsf = d
10       left_c22mp[i] = lbsf
11   return left_c22mp
```

We skip the subsequences in the exclusion zone to avoid trivial matches[74]. Finally, in line 10 we use the distance of the candidate subsequence and its leftmost nearest neighbor (which is saved in *lbsf*) to fill the *left_c22mp* array. Fig 9 demonstrate the output of this algorithm on the running example introduced in Fig 7.

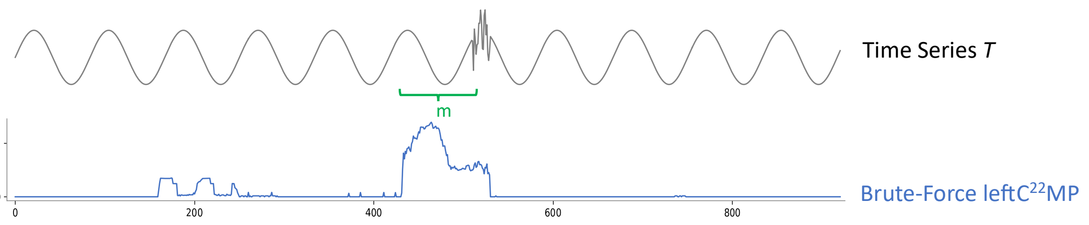


Fig 9 Top-1 *left*C^{22}MP computed on our running example.

To calculate the feature profiles, we use the subroutine in Table 2 as follows: we iterate through all possible subsequences of length m in T and calculate their catch22 features in lines 3 to 6. Also, we apply the weights to each feature value respectively. If a feature has a weight of zero, it means the feature is not used, so to speed up the feature profiles' computation we can omit this feature. Finally, in line 7 we scale the feature profiles to make them unitless and commensurate.

Table 2 Computation of *feature profiles.*

```
Function: get_feature_profies(T, m, w)
Input:
  T: Time series
  m: Subsequence length
  w: Feature's importance weights
Output:
  FP: weighted feature profiles

1  FP = zeros((len(T)-m+1,))
2  //calculate the catch22 features for each subsequence (def. 4)
3  for i in range(len(T)-m+1):
4      subsequence = T[i:i+m] //get the subsequent (def .2)
5      //pass w to apply weights while calculating catch22 features (def. 10)
6      FP[i] = cal_catch22_feature(subsequence, w)
7  FP = scale(FP) // scale feature profiles
8  return FP
```

We noted that in Table 2, in line 7 that we scale the feature profiles to make them unitless and commensurate. We use min-max normalization to rescale the range of all the features in the range of [0,1]. Note with this scaling for offline analysis, we can guarantee that the scaled sample will be in the desired range, however, that is not true for *online* prediction. It is possible that new incoming data will be outside the bounds of the minimum and maximum values, therefore the new scaled data will exceed the range of 0 and 1. Consequently, when using ORR in an online setting, we need to check for these observations prior to making predictions and either remove them from the dataset or limit them to the pre-defined maximum or minimum values.

2.4.2 Fast computation of leftC^{22}MP

In The brute force algorithm shown in Table 1 correctly computes the *left*C^{22}MP but is too slow to be practical for large datasets. Here we will show how to accelerate its computation. The key insight is that, to extract the top-*K* discordia, we only need to have exact values for the *K largest* values in the *left*C^{22}MP. For all other values, it suffices to know any upper bound that is less than the top-*K* value.

For example, suppose we are computing the discordia value for the *j*th subsequence, and further suppose, that for a previously seen subsequence at *i,* we have recorded the *best-so-far* discordia value of 4.5. As we begin to compute the value of the *j*th subsequence, we must start by initializing a variable to infinity, and then we need to update its value every time we encounter a closer match than previously encountered. Thus, the value of the variable will monotonically decrease: ∞, 7.2, 5.3, 4.2, 2.9, 2.6. 2.2,… However, once we encountered the value of 4.2, we could have admissibly abandoned our search, as its value is less than our current *best-so-far* of 4.5.

The utility of this early abandoning idea depends on how quickly we find any neighbor that is closer than our *best-so-far*. In the best case, it could be after examining a *single* neighbor, but in the worst case it could require a search through all the data. Nearest neighbor searches can often be accelerated by indexing, however it would be essentially impossible to build an index in real time on a fast moving stream. However, there is a simple strategy we can use to accelerate our early-abandoning search. Instead of a search *forward* from *time zero* to *now* (present time), we can search *backward* from *now* to *time zero*. The reason why this can be expected to be more efficient is simply that there is generally significant autocorrelation in the feature profiles, and much less correlation between current feature vectors and feature vectors in the distant past. This is because almost all systems slowly drift over time.

This means that all things being equal, the more recent data will be more likely to be similar to current data.

In Table 3, we formalize these observations with the ORR (Observed Repudiation of Regularity) algorithm.

Table 3 ORR: Early-Abandoning Computation of *left*C^{22}MP

```
Function: OOR(T, m, w, s_idx)
Input:
  T: Time series
  m: Subsequence length
  w: Feature's importance weights
  s_idx: Location of split point between training and test data
Output:
  a_left_c22mp: approximate leftC^22MP

FP = get_feature_profies(T, m, w) //get feature profiles (def. 4)
left_c22mp = zeros((len(T)-m+1,))
bsf = 0   // The current best discordia score
//Scan all subsequences in test data and calculate approximate left_c22mp
for i in range(s_idx, len(T) – m + 1):
   lbsf = inf
   for j in range(i-1-exclude_zone, -1, -1):
      d = distance(FP[i], FP[j])
      if d < lbsf:
         lbsf = d
      if d < bsf and j > 0:
         Break
      elif d >= bsf and j > 0:
         continue
      elif d >= bsf and j == 0:
         bsf = lbsf
   a_left_c22mp[i] = lbsf
return a_left_c22mp
```

In line 1 we obtain the feature profiles of the given time series *T*. In lines 5 to 17 we iterate through all possible subsequences in test data and find their approximate left nearest neighbor as follows: for each subsequence (skipping the exclusion zone, as per Definition 6) we go back till we find a neighbor that has a distance less than *best-so-far*. We use that distance to fill the *a_left_c22mp*. If we go back so far that we reach the beginning, this means that we have a new best discordia. In that case we not only use the nearest neighbor distance to fill the *a_left_c22mp* but we also use it to update the *best-so-far*.

Fig 10 provides visual evidence of the correctness of ORR.

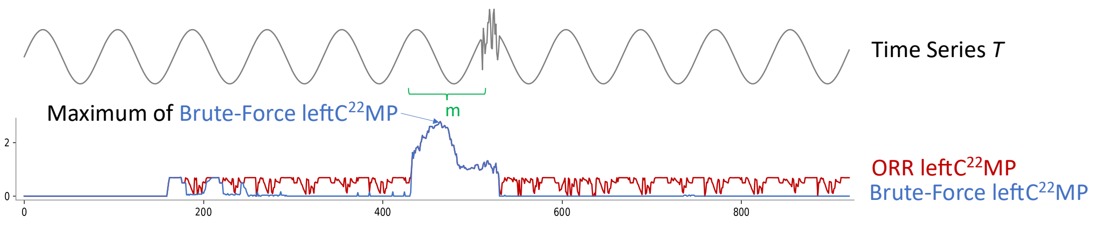


Fig 10 A comparison of the output of the top-1 brute-force and the ORR algorithm on a small toy dataset.

Note that while the output of the brute force algorithm and our swifter ORR algorithm can diverge, the following properties are true:

- The two outputs *exactly* agree at the location of the maximum of the brute force algorithm.
- No value in *left*C^{22}MP is greater than the maximum of the brute force algorithm.

These properties combined mean that, just as with the brute force algorithm, we can report the *K* highest values of the output of the ORR algorithm as the top-*K* anomalous regions.

As we will show in Section 2.5.10, the early-abandoning algorithm is several orders of magnitude faster than brute-force search and allows us to address datasets with billions of datapoints.

2.4.3 Setting the Feature Weights for $C^{22}MP$

The ORR algorithm introduced in Table 3 allows the efficient discovery of *discordia*, but if we use all twenty-two of the C22 features we may be doomed to poor performance. In particular, we may be condemned to reporting many false positives if we use features that are irrelevant to the anomaly detection task. To see this, we can revisit the example shown in Fig 1 To enhance the flow of the introduction, we originally presented the better clustering simply as catch22. However, as made clear in Fig 11.*right*, this clustering was obtained by using a *subset* of catch22 features. Had we used all the features, we would have obtained the clustering shown in Fig 11.*left*. Note that this clustering is still better than using Euclidean distance, but it becomes evident that a judicious selection of features can greatly benefit catch22.

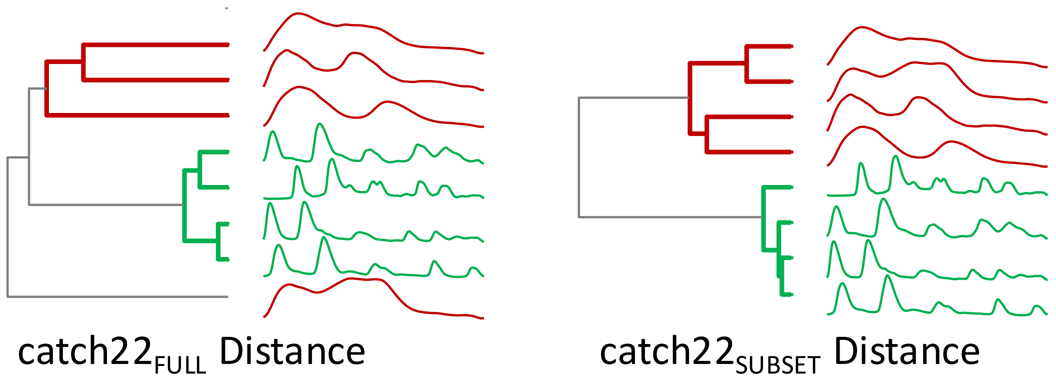


Fig 11 Instances from the 50words dataset clustered using the full set of catch22 features (*left*) and a subset of catch22 features (*right*). The full set produces a reasonable clustering, but incorrectly considers a red object to be an outlier (cf. Fig 1).

While this suggests that a good choice of features weights will improve the sensitivity and specificity of the *left*C^{22}MP, we think it is very unlikely that there exists a single best strategy to find such weights. An anomaly detection algorithm can be deployed in diverse settings, spanning the space of no-labeled-data vs. copious-labeled-data, no-domain-knowledge vs. rich-domain-knowledge etc. Instead, in this section we will consider a handful of techniques to discover the appropriate weights for the C22 features.

2.4.3.1 Hand Coded Subsets of Features

In certain instances, practitioners can leverage their domain expertise to manually design a suitable subset of features. For example, suppose that we know that for our domain it is possible that the typical patterns might be observed flipped upside down, and that this is *not* to be considered anomalous. However, if the patterns are ever observed flipped left-to-right, that is indicative of an anomaly.

Knowing this, we can use features that are not sensitive to data appearing flipped upside down. These include features {5, 6, 7, 8, 9}. However, we want to avoid using features that are sensitive data appearing flipped upside down, these include features {1, 2, 5, 21}. In order to help such practitioners, we have created a detailed key of all the invariances of catch22 [12][2]. This is an attractive solution for users that have both a familiarity with c22 features and a strong initiative understanding of their domain of interest. Since such users may be rare, below we consider other possibilities.

[2] This contrived example is not as implausible as it may seem. Suppose we are monitoring the accelerometer time series from a smartphone in a user's pocket. If the user takes a call, and then returns the phone to her pocket upside down, the Y-axis time series will flip upside down, but will not be flipped backwards.

2.4.3.2 *Feature Search with Copious Training Data*

In a handful of situations, we may have copious training data to learn an appropriate feature weighting. Here we must ward off a possible confusion. There are papers in the literature that claim to be doing *anomaly detection*, but are arguably doing *classification*, with "anomaly" simply being the rarer class. One could critique such efforts by noting that if we have examples of the anomaly in advance, this is simply classification or pattern matching. The normal definition of anomaly suggests an unknown/unexpected pattern, precluding the possibility we have examples in our training data.

However, here we are not suggesting we learn the anomaly *patterns,* we are simply learning which *features* are sensitive to disturbances in the domain of interest. We propose to learn these features by treating the training data as a classification problem and optimizing the classification error-rate by a simple greedy forward search. Table 4 demonstrate the pseudocode for this idea.

Table 4 Pseudocode for feature search with labeled data

```
Function: feature_search(T, y)
Input:
  T: a set of normal and anomaly time series
  y: labels for each sample
Output:
  w: Feature's importance weights
1   transfomed_T = []
2   for t in T:
3       transformed_T.append(cal_catch22_features(t))
4   model = Classifier()
5   model.fit(transfomed_T, y)
6   w = model.feature_importances
7   return FP
```

In lines 1 to 3 we iterate through all samples in the labeled data and calculate the catch22 features for each time series. We create a classification model in line 4, then train this model to classify normal and anomaly samples in line 5. Finally, we extract the feature importance learned through classification in line 6. Note we can use any explainable classification model in this method. For example, we can use a Decision Tree Classifier, in which the feature importance scores are estimated by calculating Gini gain (the amount of Gini impurity that was eliminated at each branch of the decision tree) or a logistic regression model in which the feature importance scores are estimated by using the learned coefficient value for each feature [50].

We can also use the leave-one-out mechanism with any classification model. Where in each iteration we use one of the twenty-two features to classify the samples. Subsequently, we select the feature set that yields the highest classification accuracy.

2.4.3.3 Feature Search by Feature Profiles

In scenarios where we have limited labeled anomalies (maybe just one or two anomaly samples), the method presented in section 2.4.3.2 would not be suitable. In such cases, an alternative approach is to analyze the feature profiles in order to learn the weight of each feature. Intuitively, we expect to see a feature in the feature set if its feature profile "responds" to the anomaly of our interest. For example, in our toy example in Fig 7, we do not expect to have features 1 and 2 in the feature set, as their feature profiles seem to vary independently of the anomaly. This suggests that we can use regression analysis to learn the feature set. We create a regression model to estimate the relationships between the time series (dependent variable) and feature profiles (independent variables), then we extract the learned relationships (the coefficients). The coefficients can be used as the feature importance score. In general, feature importance refers to how useful a feature is at predicting the target variable, which, for our case, is the time series and the anomaly within it. Table 5 formalizes this idea.

Table 5 Pseudocode for feature search with feature profiles.

```
Function: feature_search(T, FP)
Input:
  T: Time series
  FP: feature profiles
Output:
  w: Feature's importance weights
1  model = Regression()
2  model.fit(FP, T)
3  w = abs(model.coefficients)
4  return w
```

In line 1 we create a linear regression model. To learn the relationship between the time series and its feature profiles, we fit the model to the time series and its feature profiles in line 2. Finally, we extract the coefficient learned during regression as feature importance in line 3. Similar to using a classification model to learn feature's weights, we can use any explainable regression model in this approach such as linear regression or a Decision Tree Regressor [38].

2.4.3.4 Human in the Loop Feature Search

In Section 2.4.3.1, we proposed to use domain expert's knowledge to choose features. In such cases, we assume that the user has detailed domain knowledge that will help them choose the right features. For example, in the Oil&Gas production domain, conservation laws (i.e., conservation of mass, energy) constrain the values that can be seen in normal telemetry, and features can be chosen to be sensitive to deviations from these [3]. In this section we propose to exploit less formal and explicit human knowledge. Our only assumption is that a user could recognize an anomaly if she saw one. Given that assumption, could we generate artificial, but plausible anomalies?

If so, our difficulties are over, we could simply revert to one of the feature learning algorithms in the Section 2.4.3.2 and 2.4.3.3.

Here we are inspired by an observation by Sir D'Arcy Thompson. Fig 12 shows some images from his 1917 book *On Growth and Form* [51], which shows it is possible to reproduce almost all the morphological diversity of fishes, with just a few samples and a small set of geometric transformations.

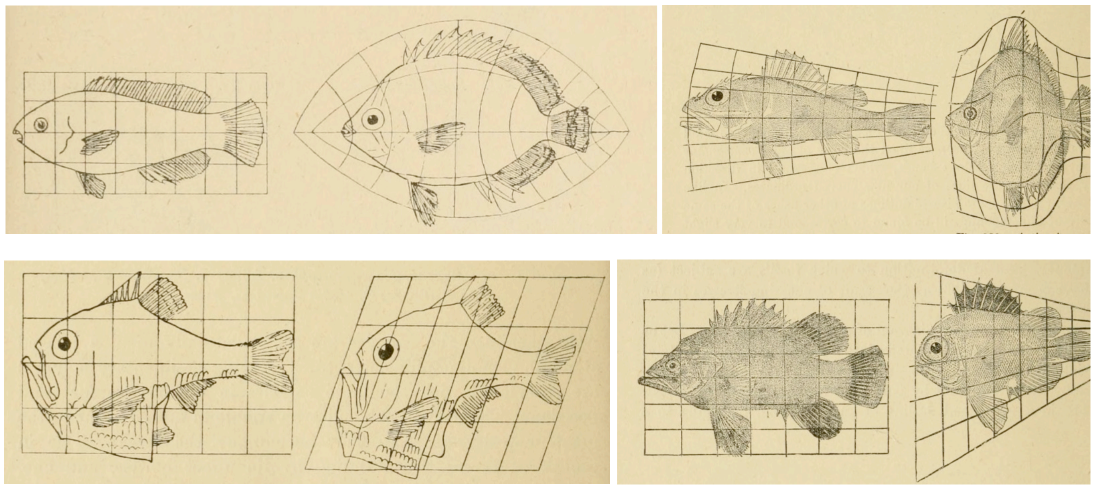


Fig 12 A sequence of figures from [51] suggests that we can reproduce almost all the diversity of fishes, using just a handful of "prototype" fish, and small number of linear (bottom row) and non-linear (top row) distortions. For example, the figure in bottom left shows that we can take a known fish, here the Olfer's Hatchetfish (*left*), and apply a simple geometric transformation to obtain another fish that really exists, here the Diaphanous Hatchetfish (*right*).

We believe that we can similarly create plausible synthetic anomalies [35][53]. Consider the two datasets shown in Fig 13.

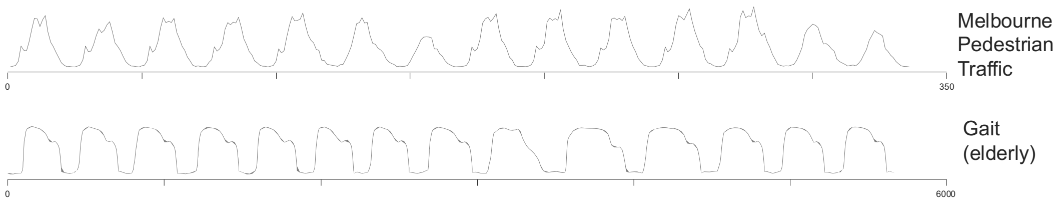


Fig 13 Two datasets with about 14 cycles shown. Both are periodic, but Melbourne has almost no temporal variability, the rigidity of the 9 to 5 business cycles acts to keep the days in synch. In contrast, the gait cycle shows the natural variability of walking in an indoor environment with obstacles, doorways, changes of direction etc.

Assuming that we are seeing normal data, how should we weigh features that are sensitive to time warping of the data? This is a difficult question, even if we know which features are sensitive to such distortions (recall we have built a dictionary to help build such intuitions [12]). Moreover, the question depends on the user's task and domain knowledge, which may be intuitive and hard to elicit and represent. However, consider Fig 14, where we added the *same* amount of warping to the middle section of each time series.

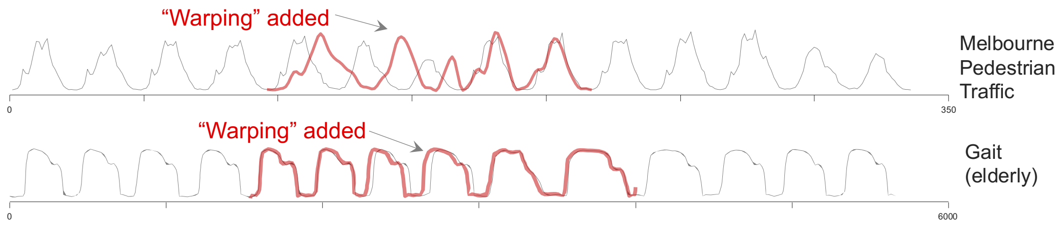


Fig 14 The two datasets shown in Fig 13 with the same amount of warping added to each's middle section. The addition of warping looks natural on the gait data but is visually jarring for the Melbourne pedestrian dataset. This domain induced distinction offers a clue as to how we can learn the best set of features for each domain.

For the Melbourne Pedestrian dataset, there are a lot of variabilities, but it is mostly manifest in local changes in amplitude (especially weekends vs. weekdays). Adding this amount of time warping creates a visually obvious anomaly. In contrast, for the Gait dataset, the added warping is subsumed within the existing natural variability.

This example exemplifies our human-in-the-loop approach. For each distortion of interest (including *noise*, *spike*, *warping*, *linear scaling* etc. [11][12][65][71]), we show the user four randomly chosen snippets from their dataset. The plot has an interactive slider that allows the user to increase or decrease the amount of distortion. The user is invited to move the slider to indicate the maximum amount of distortion she would accept before declaring at least one of the four snippets an anomaly.

Once the user has annotated the data in this way, we can simply avail of the feature search algorithm discussed in the previous section to learn the appropriate features. There are several research efforts on time series generation, almost all of which use deep learning [2][72]. However, we found that for each distortion we could make high-quality examples with just a few lines of code (see [12]). In [12], we show a video of a typical interaction with a user. Note that the entire process only takes a few minutes.

2.5 Empirical Evaluation

To make certain that our experiments are reproducible, we have built a website [12] that contains all the data/code used in this work. All experiments were conducted on an Intel® Core i7-9700 CPU at 3.00GHz with 32 GB of main memory, unless otherwise stated.

2.5.1 How Should we Evaluate TSAD Algorithms?

A recent series of papers from various research groups have cast significant doubt on both the common TSAD benchmarks and evaluation metrics [66]. While we have neither the space nor the inclination to weigh in on this debate, we need to at least briefly explain some of the issues to justify how we do *our* evaluation.

Consider Fig 15, which shows an excerpt of one dimension (FIT401) of the 51-dimensional SWAT benchmark [20]. This is one of the most cited and used benchmarks in the literature, appearing in at least 200 papers [7][16][36][57]. We ran the fixed weigh *left*C^{22}MP algorithm on this dataset. Unsurprisingly, it strongly peaks at both the transition from normal data to the anomaly, and from the anomaly to normal data.

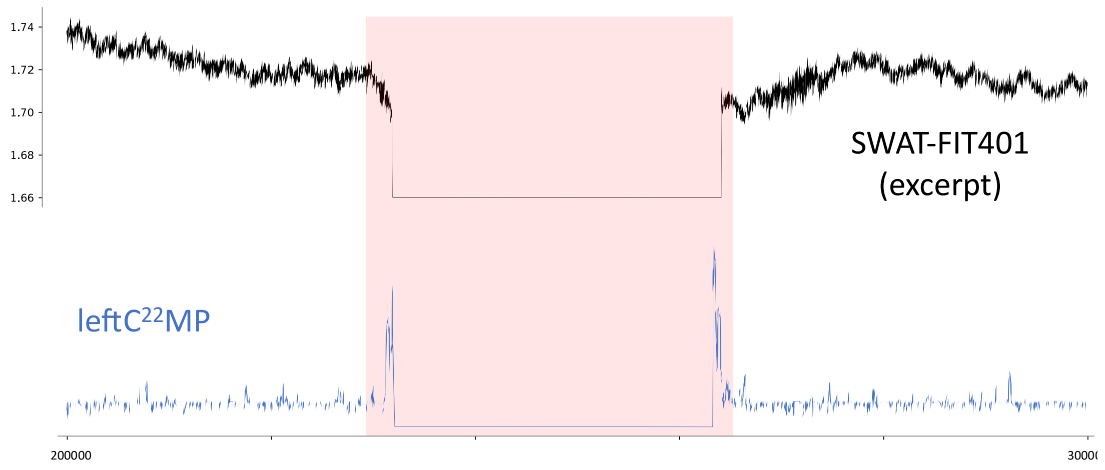


Fig 15 An excerpt from the SWAT-FIT401 dataset. The ground truth anomaly is highlighted in red. Unsurprisingly *left*C^{22}MP can easily find this anomaly. The constant region that forms the anomaly actually has a value of exactly zero, we added a constant to it to make the variability of the normal data clearer.

This example highlights the problems the community has noted:

1. Many benchmark problems are *much* too simple to warrant claims of a successful algorithm. Note that while we did use *left*C^{22}MP to find the anomaly, we could have found it with a single line of MATLAB [66]: `isAnomaly = (FIT401==0);`
2. We would argue that here that detecting this anomaly should count as a single *binary* success. However almost all papers reason like this: *The anomaly is marked as 35,800 datapoints long in a dataset of 449,921 datapoints, so we should report detecting this with four significant digits!* In our view, this is misleading spurious precision; it implies precise measurements supported by thousands of *independent* datapoints, not a single "blip". We think this argument to be forceful, but to our surprise, some have pushed back at it[3]. We have found the following analogy to be fruitful (with apologies for the graphic imagery).

[3] In blog forums, private conversations, openreview.net etc.

Suppose we suspect that John, a 100 kg peace worker was killed by a landmine. Two forensic anthropologists are sent to investigate. Alice finds a large skull fragment, does a DNA test, and concludes that **John is dead**. Bob finds a leg and a hand, which in total weight 15 kg, does a DNA test, and concludes that **John is 15% dead**.

We believe that this story is a perfect analogy for the example in Fig 15. Once we have seen the value of the time series plunge to zero, this sensor is unambiguously "dead". Any scoring function that *continues* to reward you for finding additional sections of constant region time series is just like suggesting that Alice would be surer that John is dead is she later found a toe fragment. Likewise, if Bob reweights his fragments with a more sensitive scale and then reports **John is 15.49% dead**, we would think him quite naïve. With this understanding, the dozens of papers that report four significant digits on this dataset do seem somewhat naïve. In [16] the authors summarize the results of twenty-seven approaches to SWAT, that appeared in eight unique papers. Sixteen of the approach are reported with four significant digits, and the rest with three significant digits. It is hard to reconcile such precision with even a casual inspection of this dataset.

3. Problem '2' above is compounded by poor labeling. The begin—end locations for this anomaly are given as 227900—263700. The fact that these are multiples of 100 should tell us that these are the rough guesses by the annotator. In fact, careful visual inspection strongly suggests that these labels have an uncertainty *greater than* ±100. This uncertainty would not change the *binary* success we advocate for but would change at least two of the four significant digits that most papers report[16].

In summary, we have sympathy for the claims that both the common benchmarks and the evaluation metrics [66] are flawed.

Given these issues, how should we evaluate? We agree with the claim in [45] that one of the best ways to understand a TSAD's strengths and limitations is to directly plot the scoring function next to the time series, as we did in Fig 15 above. In this next section, we will do exactly that, on an ambitious scale. In addition, in subsequent sections, we will use datasets and metrics that are free of the flaws discussed above.

2.5.2 A Visual Intuition for leftC^{22}MP

With the caveats in the last section in mind, in Fig 16, we show the performance of *left*C^{22}MP on datasets collected from twenty different papers. Clearly, we do not have the space to discuss, or even cite, all twenty papers. However, in [12] we have a document that gives detailed provenances for all datasets and shows larger figures.

These results strongly hint at the generality and effectiveness of *left*C^{22}MP, the top discordia peak at the ground truth locations. Nevertheless, they are informal and anecdotal evidence. Moreover, they only show that *left*C^{22}MP is expressive enough to *represent* the anomalies, not that we can learn an appropriate set of feature weights to discover them. In the following sections, we will provide more rigorous evaluation.

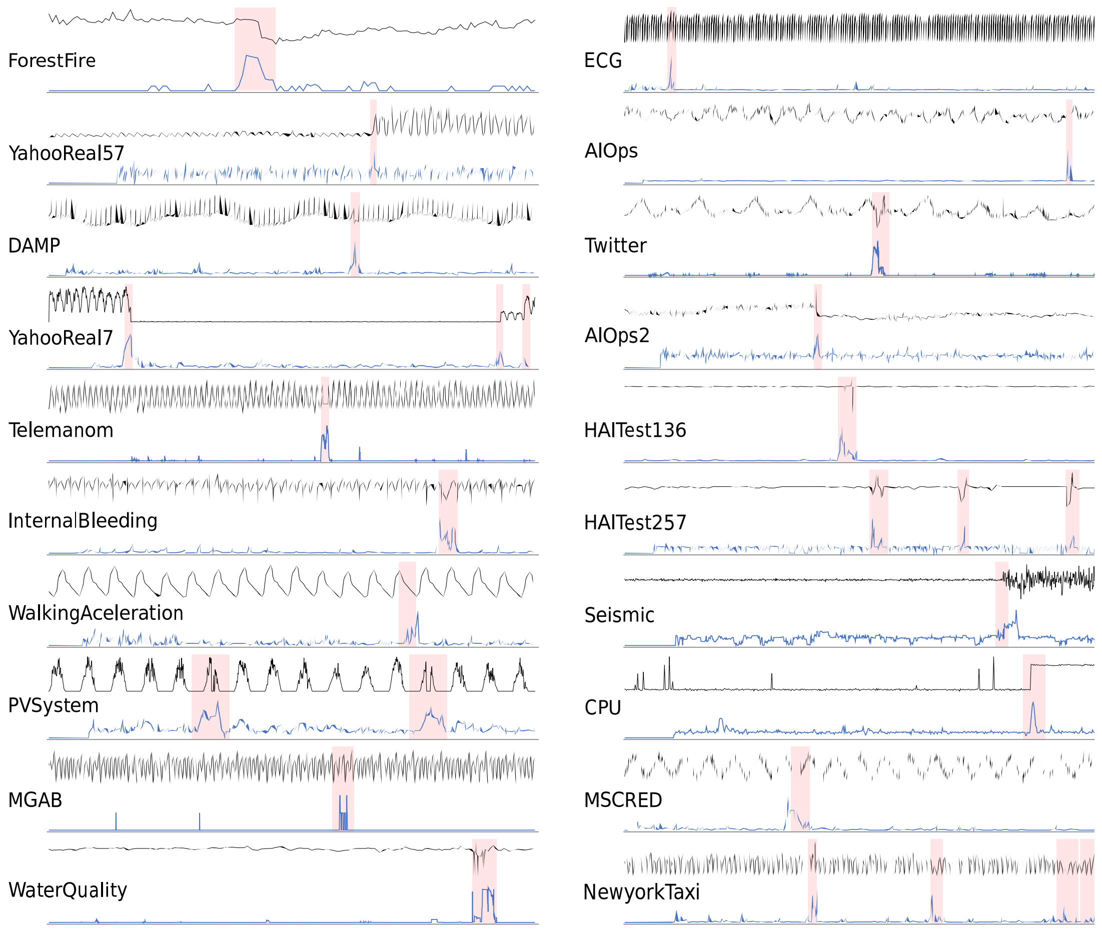


Fig 16 The performance of *left*C^{22}MP on datasets that appeared in twenty different papers in the last two decades.

2.5.3 The Hexagon ML/UCR dataset

While there are many benchmark datasets proposed in the literature, most of them do not allow comparisons among published results. The problem is that the metric for computing a success can vary greatly from paper to paper, making the published results incommensurate [17][18][35][47][72]. The **HEX**agon ML/**UCR** dataset is different in that the 250 datasets come with a concrete and well-reasoned metric of success. For example, different TSAD algorithms may tend to peak at the beginning, or the middle, or the end of an anomaly. These differences are generally inconsequential but combined with a brittle scoring function could make a good algorithm score poorly.

The HEX/UCR scoring function allows a little slop (±100) in the scoring boundaries to make this issue moot.

In addition, the HEX/UCR datasets are unusual in having detailed provenance, largely freeing them from the flaws noted in [66]. Many of the datasets are completely natural, the remainder are anomaly-free datasets that have had exactly one anomaly seamlessly added in a domain-specific way. The datasets come from diverse domains, including medicine, industry, biology and meteorology. Table 6 shows a comparison of our proposed method with rival approaches. The results highlighted in blue are results that we copied from other works. All other results we computed ourselves.

Table 6 A comparison of 14 algorithms on the HEX/UCR dataset

Method	Score	Method	Score
*left*C^{22}MP	0.568	{{{ DAMP + *left*C^{22}MP }}}	{0.692}
DAMP (*left*MP)[39]	0.556	USAD [5]	0.276
AE[5]	0.236	Telemanom [27]	0.468[4]
LSTM-VAE [46]	0.198	SCRIMP (*full* MP) [74]	0.416
RRCF [48]	0.030	MERLIN (generalized MP) [48]	0.440
MDI [48]	0.470	NORMA [9]	0.474
TranAD [48]	0.190	GANF [48]	0.240

Our proposed algorithm outperforms all rival approaches. It is slightly better than DAMP, which is regarded as SOTA [39]. The approach labeled {DAMP + *left*C^{22}MP} is *not* a true algorithm. It is a post-hoc ensemble where we pick the better of the two algorithms *after* seeing the labels. The ensemble's exceptionally high score tells us that the two algorithms in question have largely uncorrelated errors and suggests a future research direction in combining TSAD algorithms.

[4] Telemanom runs out of memory on the larger datasets. This score is extrapolated from the shorter/easier datasets, thus optimistic **Error! Reference source not found.**.

While the format of these datasets and the scoring function allows comparison of *accuracy* across papers, it does not allow comparison of *timing results*. However, even if we assume that all the datasets are sampled at 250Hz, C^{22}MP can comfortably process all datasets at least an order of magnitude faster than real time.

2.5.4 Sensitivity to Subsequent Length

In the previous section we showed that *left*C22MP can score very well on the HEX/UCR datasets, if we use the simple heuristic suggested by DAMP to set the value of m. Nevertheless, it is natural to ask how sensitive the algorithm is to that parameter.

In Fig 17, we consider the InternalBleeding dataset (chosen because it is one of the shortest datasets from UCR/HEX, and well can plot it in its entirety), and we test every parameter from ¼ to 4 of the DAMP suggested value. This plot suggests that the *left*C22MP algorithm is not particularly sensitive to the setting of *m*.

much touted in recent years, yet none seem to do particularly well. We believe dataset shown in Fig 18.A can cast light on why this was the case.

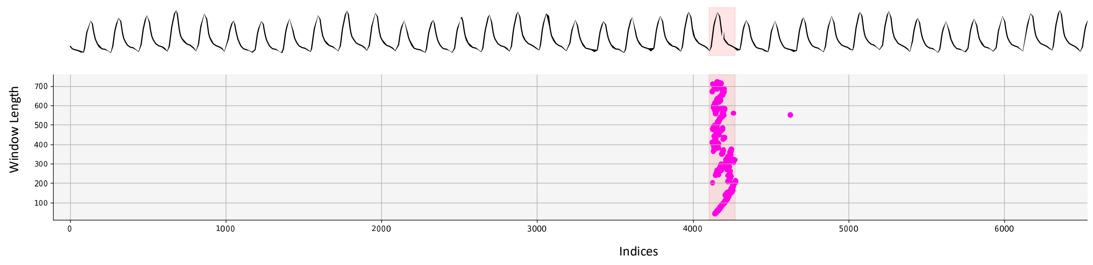


Fig 17 The *m* value suggested by DAMP for InternalBleeding dataset is 181. Therefore, the range of the *m* would be from 45 to 723. C^{22}MP is able to detect the correct location of the anomaly for almost all window length in this range.

2.5.5 Why is C^{22}MP so Robust?

The results in the previous section may appear a little surprising; deep learning approaches have been much touted in recent years, yet none seem to do particularly well. We believe dataset shown in Fig 18.A can cast light on why this was the case.

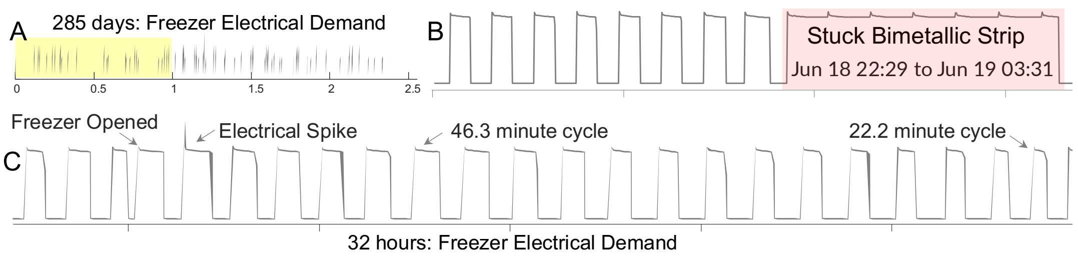


Fig 18 A) An electrical demand dataset for a single device, a freezer. B) The (rather obvious) anomaly caused by a stuck temperature sensor. C) A random section of the training data.

In Fig 18.B we zoom-in to the sole anomaly in this dataset. It does not appear to be particularly challenging, yet all deep learning approaches we tried fail to discover it. An examination of the training, a snippet is shown in Fig 18.C tells us why this is the case.

- The period of the data (the compressor on/off cycles) can vary based on the room temperature, which varies both daily and seasonally. Many published algorithms seem to work well with *strict* periodicity (like the Melbourne dataset shown in Fig 13) but cannot handle *variable* periodicity.
- The dataset (both the train and test) is replete with variations that are inconsequential. For example, very short off-cycles caused by the freezer being opened, or electrical spikes. These two simple events can appear in arbitrary arrangements that all look essentially the same in the *feature* space, but very different in a *shape* space.
- The training data is 100,000, not a large dataset by modern standards, but this size strains many deep learning approaches.

In Fig 19, we compare three approaches on this problem. As the figure shows, the *left*C^{22}MP easily finds this anomaly but Telemanom dramatically fails. As Telemanom is nondeterministic, we ran it five times; the *best* run had 32 false positives. Moreover, the total time required for a single run is 3.04 hours, which contrasts poorly with the 7.25 minutes required for *left*C^{22}MP. Here DAMP, using the subsequence length selection tool suggest in the original paper [39], did *just* find the anomaly, but it would have failed at a slightly different length.

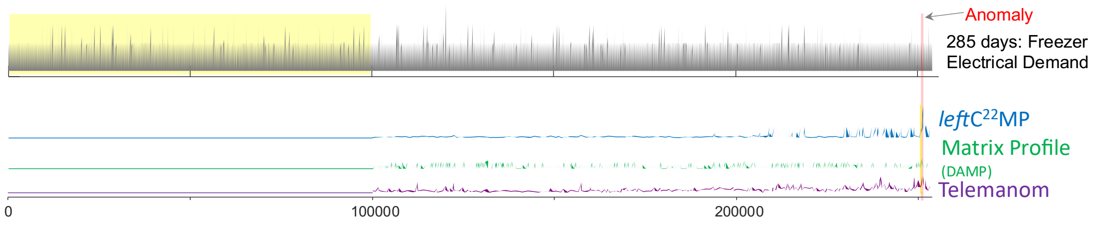


Fig 19 The *left*C^{22}MP can easily find the Stuck-Bimetallic-Strip anomaly. DAMP *just* finds it, and Telemanom fails.

To test the correctness of our explanations of Telemanom's (and more generally, most deep learning approaches) poor performance that we posited above, we built a proxy for the dataset with perfect square-waves, and tested algorithms on it as we introduced synthetic spikes and variable periodicity etc. The results (expounded in [12] for brevity) support our observations and suggest research directions for advocates of deep learning TSAD algorithms.

2.5.6 Case Study on Medical Data

The catch22 features are very expressive, but not completely so. Several papers have needed to augment the features with some domain-specific feature(s)[1] [37]. Here we give an example of where we had to do this, to create C23, with C23 = [C22 ∪ Max(T)].

We consider the task of anomaly detection in telemetry of CCT, Contraction in Cardiac Tissue (related to, but distinct from the more familiar ECGs). A recent paper [43] introduced a sophisticated data generator that takes real data and introduces into it highly plausible anomalies of amplitude, duration, morphology or timing. These anomalies are very interesting: Some are visually apparent; but some, for example, 'Fast Pulse Decay', are difficult to see even if you know where the anomaly is. Moreover, some anomalies are caused not by the *addition* of a new pattern or disturbance, but by the *omission* of an expected pattern (Pause and Missing Pulse).

In Table 7 we show the results of comparing our approach with two bespoke deep learning algorithms. To be fair to deep learning algorithms (which have many sensitive parameters and can be difficult to reproduce), we copied the results from [43] and used identical settings for our experiments (also averaging over 100 runs).

Table 7 A comparison of four TSAD algorithms on CCT data. The best performing algorithms are highlighted in bold

Anomaly Type	**Autoencoder**	**LSTM**	***left*C^{22}MP**	***left*C^{23}MP**
Lower Amplitude	0.55	0.83	0.41	0.90
Higher Amplitude	0.71	0.99	0.18	0.95
Missing Pulse	0.63	0.91	0.97	0.97
Slow Pulse Decay	0.67	0.95	1.00	1.00
Fast Pulse Decay	0.61	0.88	0.98	0.98
Early/Anticipated Pulse	0.51	0.77	0.99	0.99
Pause/Block	0.61	0.90	1.00	1.00
Average Accuracy	0.61	0.89	0.79	0.97

In order to "stress test" our approach, we did not do any feature selection here, we simply used all 22, and all 23 features. These results are very promising. Note that we are beating two domain-informed deep learning approaches that are heavily customized for the task-at-hand, with our simple generic approach.

2.5.7 Learning a Threshold

Recall that in Section 2.2.1 we explained that TSAD algorithms can be seen as having two parts:

1. Identify the location that is most likely to contain a subsequence that will be considered anomalous.

2. Make a decision as to whether to actually sound an alarm.

We further explained why we focus our evaluation on the first part. Nevertheless, for completeness we will show how we can address the second task and make a binary decision flag the subsequence as anomalous or not. For this purpose, we simply need to learn a *threshold* on the training data.

To test the effectiveness of ORR on this second task, we conducted an experiment using the CCT data generator that was introduced in Section 2.5.6. We generated one hundred different datasets using different seeds, each of which consisted of eight columns. The first column contained only normal CCT data, while the remaining columns contained one specific type of anomaly: 'Slow Pulse Decay', 'Fast Pulse Decay', 'Early/Anticipate Pulse', 'Missing Pulse', 'Pause/Block', 'Higher Amplitude', and 'Lower Amplitude', respectively. We used the data in the first column (the normal time series) as training data. To learn the threshold, we calculated the *left*C^{23}MP of the training data across one hundred trials. Since the training data only contains normal CCT, the maximum value of the *left*C^{23}MPs can be used as decision threshold. Any sample with an anomaly score higher than the threshold is considered an anomaly. In this experiment we aimed to stress test the effectiveness of C^{23}MP and its ability in detecting different type of anomalies. To achieve this, we used the smallest and largest decision thresholds (with an additional epsilon value) obtained from one hundred trials for all the datasets as the threshold. The results of this experiment are presented in Table 8.

Table 8 The result of *Left*C^{23}MP on CCT data for the task two.

Anomaly Type	***left*C^{23}MP Threshold = 1.12**	***left*C^{23}MP Threshold = 1.44**
Lower Amplitude	0.69	0.02
Higher Amplitude	0.83	0.01
Missing Pulse	0.99	1.0
Slow Pulse Decay	1.00	1.0
Fast Pulse Decay	0.92	1.0
Early/Anticipated Pulse	0.76	0.38
Pause/Block	1.00	1.0
Average Accuracy	0.88	0.62

Table 8 shows that ORR has an average accuracy of 0.88 when threshold 1.12 is selected. However, the accuracy varies depending on the type of anomaly. Specifically, we can see that ORR performs poorly in detecting 'Lower Amplitude', 'Higher Amplitude' and 'Early/Anticipated Pulse' anomalies, but well in detecting other anomalies. To investigate this further, we analyzed the highest values of *left*C^{23}MPs for each type of anomaly across the one hundred trials, as shown in Fig 20.

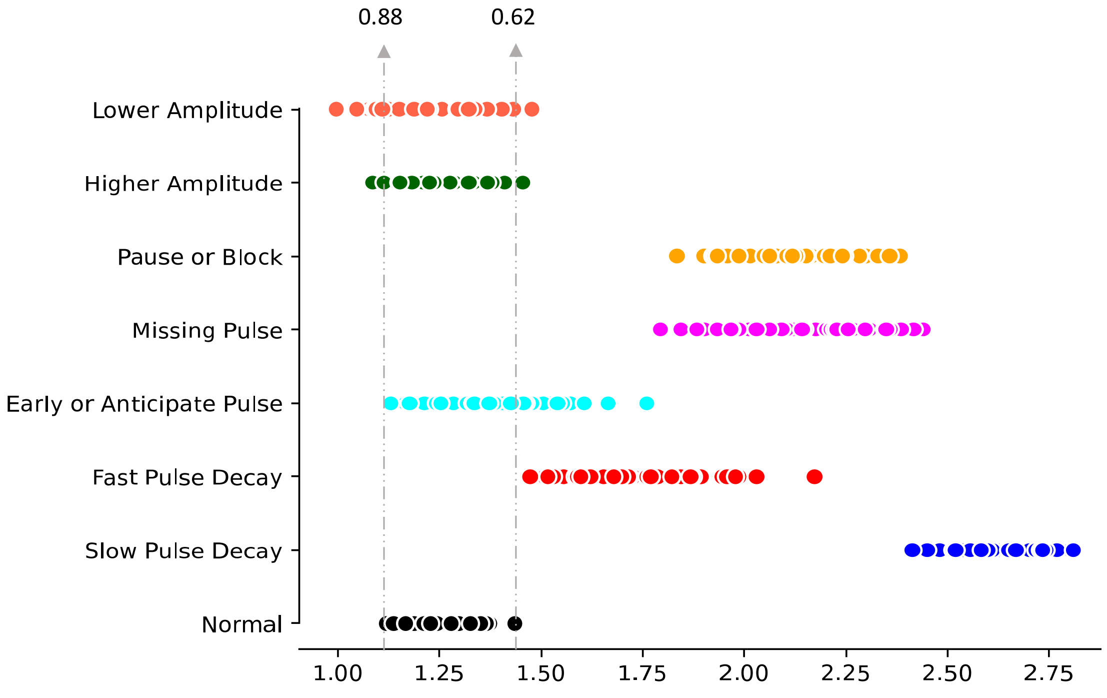


Fig 20 Comparing the distribution of the highest value across different type of anomalies.

We find that different types of anomalies have distinct clusters of high values, and the chosen threshold effectively separates normal from anomalous samples for the cases that we have high accuracy. Interestingly, we observed that for the cases in which ORR performs poorly, the distribution of high values is very similar to that of the normal case. This suggests that the anomalies in these cases may be subtle Fig 21. Shows examples of the CCT data with different type of anomalies. As we can see in the 'Lower Amplitude' and 'Higher Amplitude' only one single sample is distorted and for the 'Early/Anticipated Pulse' there is no distortion in the pattern only we have and early pulse.

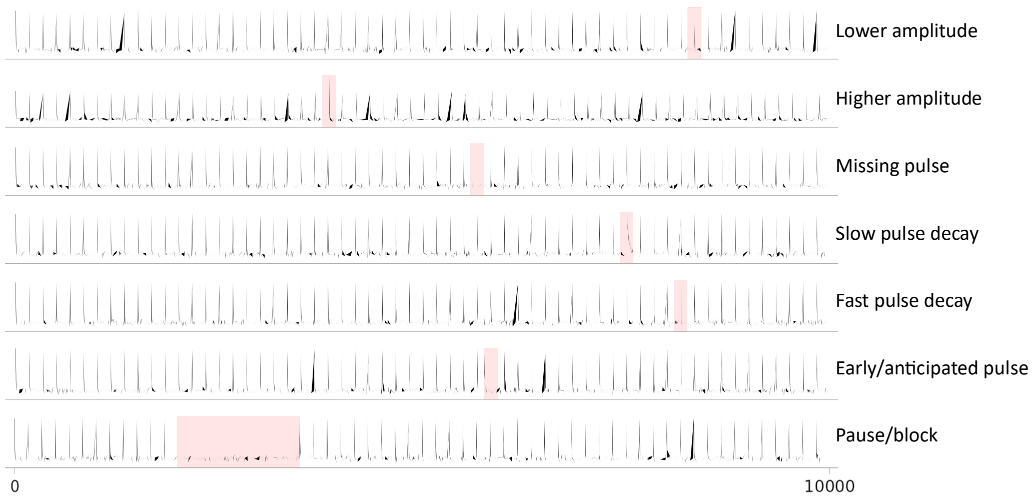


Fig 21 Examples of the CCT data with different type of anomalies. The cases observed to be difficult for *left*C22MP in Fig 20 are also very difficult to detect by visual inspection.

2.5.8 Case Study on Mouse Motion Capture

Researchers at Biology department of UCR are building a mouse model to understand factors that influence the onset of Parkinson's disease. As Fig 22 shows, one tool used to study the effect of cognitive decline is a treadmill *roller*, and the researchers have many hours of such data.

The biologists are interested in finding two types of anomalies here. First, there are simple feature extraction issues. For example, sometimes the motion capture tool (DeepLabCut [41]) is unable to find the mouse's paw as it is occluded by its tail (type **B** in Fig 22). Such anomalies can then be either manually corrected or excluded from analysis. More interestingly, however, there may be *behavioral* anomalies induced by gene knockouts or medications.

To test our ability to find both type of anomalies, the biologists used visual inspection to provide a short snippet of anomaly-free data, and a much longer region that terminated with an area dense with both quotidian and behavioral anomalies, which they carefully annotated.

After learning the feature weights using the snippet of positive-only data and our human-in-the-loop tool, we discovered the anomalies shown in Fig 22.*center*. While most anomalies were simple feature extraction issues, two anomalies surprised us. The first anomaly, type **D** (Fig 22.*top.right),* depicted a mouse grasping the treadmill and "riding" it around for a complete cycle. The second anomaly, type **C**, occurred when the mouse quickly "double-tapped" while walking on the roller, this locomotive "stutter" is likely due to a side effect of an induced pathology. This finding is particularly interesting because it is challenging to detect this anomaly with the naked eye. We invite the reader to examine [12] for more details and to view a video of the experiment.

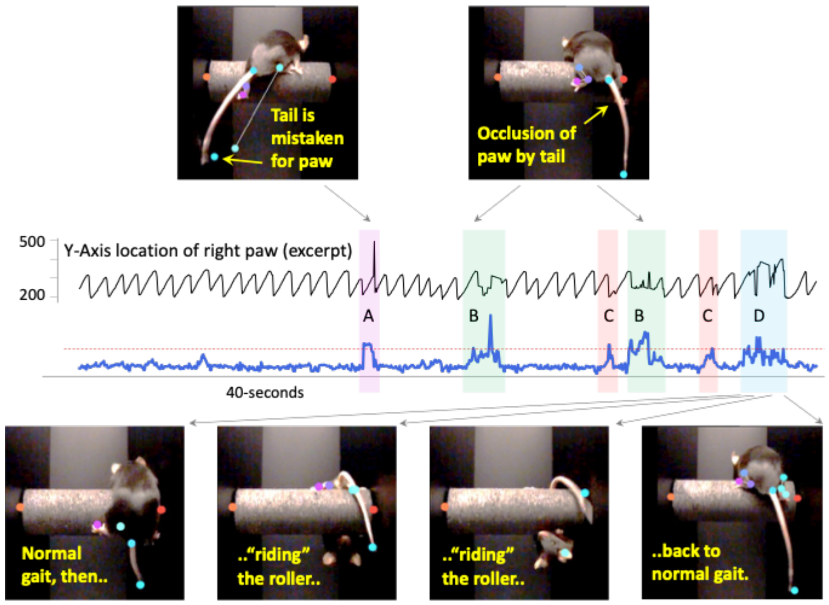


Fig 22 *center*) A trace from the right paw of a healthy mouse, with its companion C^{22}MP. *top and bottom*) Screenshots of the video with markers labeled and tracked by the DeepLabCut motion capture tool [41].

2.5.9 Case Study with Human in the Loop

In this section we evaluate the utility of using the “human annotation of synthetically distorted data” idea we introduced in Section 2.4.3.4 Consider the DISTORTEDTkeepForthMARS example from the UCR-Hexagon archive [66]. This dataset is of length 8,184, with the first 4,000 datapoints acting as a (positive only) training set. As shown in Fig 23, if we run top-1 anomaly detection on this with all twenty-two features, the best anomaly is a false positive.

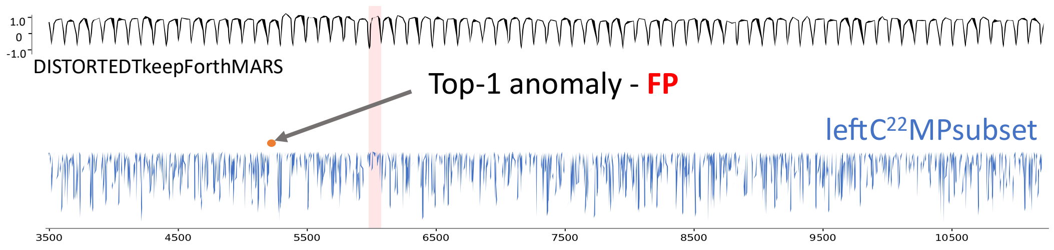
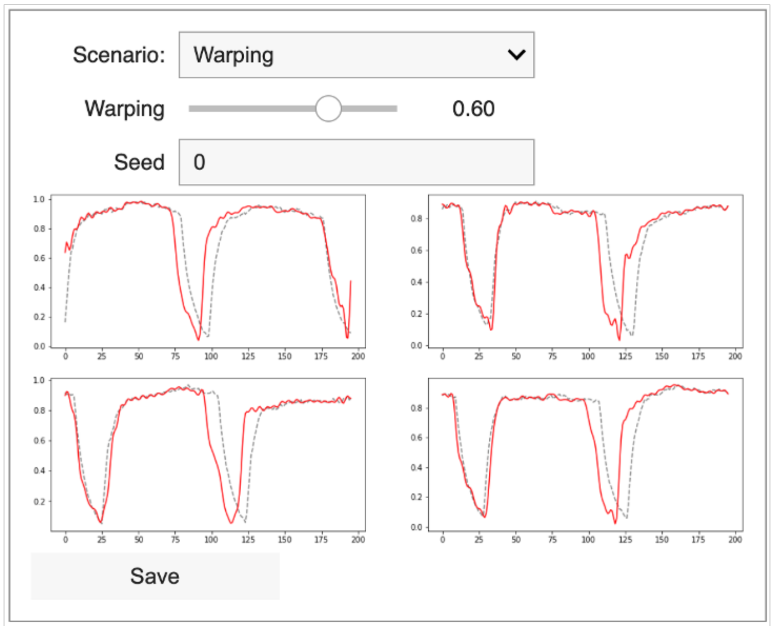


Fig 23 Using C^{22}full, the top-1 anomaly for DISTORTEDTkeepForthMARS is a false positive. The algorithm seems to have been confused by wandering baseline.

Moreover, note that the Matrix Profile (using the *left*MP, as recommended in [66]) also fails here.

We extracted the training data and used the ideas in Section 2.4.3.4 to create two classes of data: normal (sampled from original data), and (synthetic) anomalous (note during this process, we are learning the *constraints* that distinguish anomalous and normal behavior). We only considered the distortions of *warping*, *spike*, *noise*. The entire human-annotation session only took about one minute, and a complete video of the process is archived at [12]. Fig 24 shows a screenshot of this tool. Here the user can simply select the type of distortion (in this example *warping*), then adjust the amount of distortion they would like to be added to the data using the slider. In this example, we can see the original data in gray (dashed lines) and synthetically distorted samples in red. shows some sample data, both *before* and *after* the surgical intervention.



Scenario: Warping

Warping 0.60

Seed 0

Fig 24 A screen capture of a video showing adding warping distortion to DISTORTEDTkeepForthMARS data.

We then used the algorithm outlined in Section 2.4.3.3 to learn a feature set for this problem, the algorithm suggested using features {22, 10, 14, 12}. As shown in Fig 22, this feature set *does* find the sole true positive.

We repeated the same process with other UCR-Hexagon datasets, to see if learned feature sets differed from dataset to dataset. This *is* the case. For example, for DISTORTEDCIMIS44AirTemperature2 we learned the feature set {14,15,17,7,10}, and this small feature set succeeded where C^{22}full had failed.

We will not evaluate all the UCR-Hexagon datasets this way. It would clearly not be fair to rival methods to compare against an algorithm that is availing of human help. However, these examples speak to the *expressiveness* of the weighted C^{22}MP, and the relative ease of setting for good weight.

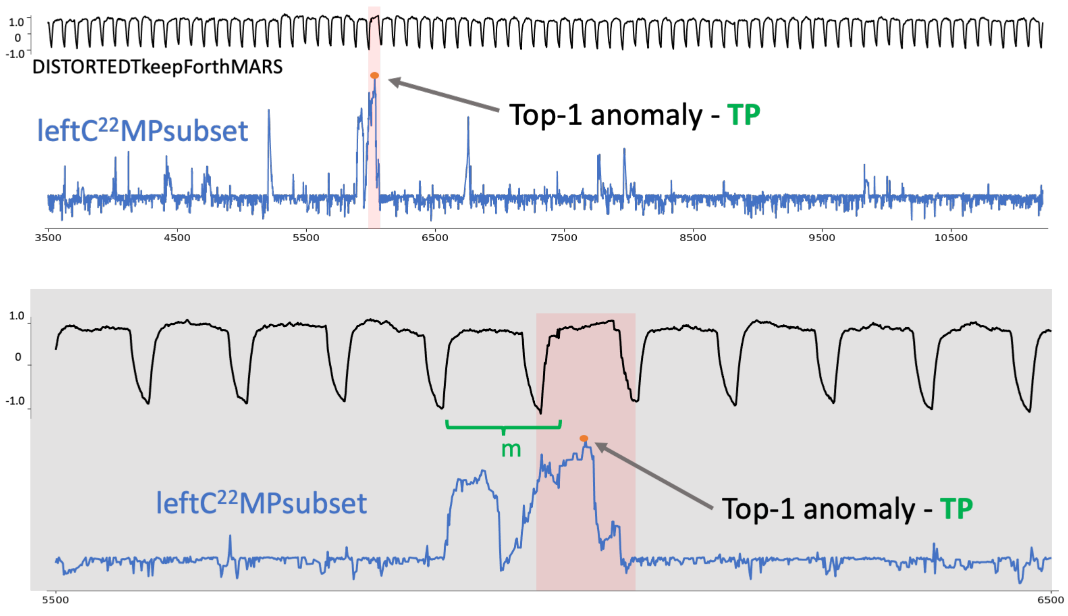


Fig 25 *top*) Top anomaly for DISTORTEDTkeepForthMARS using C^{22}subset is a true positive. *bottom*) A zoom-in of the top anomaly for DISTORTEDTkeepForthMARS using C^{22}subset.

2.5.10 How Fast is ORR?

The previous experiments suggested that we can compute the *left*C^{22}MP efficiently. Here we will more formally test this. We created a random walk time series of increasing lengths and compared the time for ORR to the time for the brute force approach. We choose random walk because it is the worst case for us, the lack of significant anomalies means that the early-abandoning technique is not as efficient as in a dataset that has anomalies.

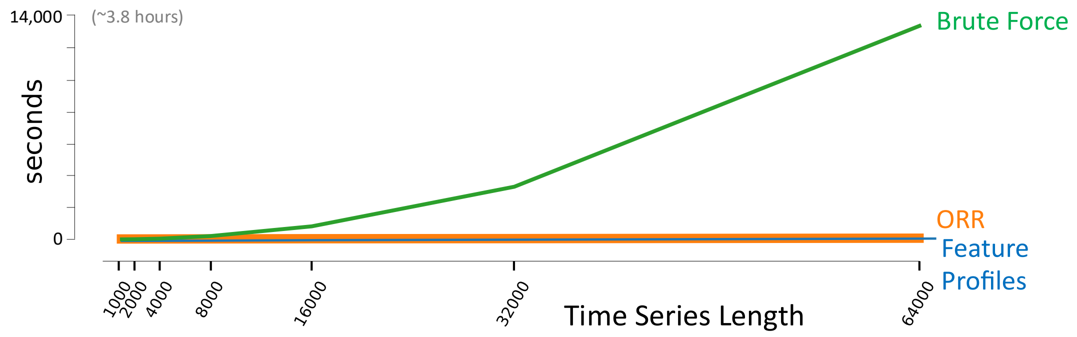


Fig 26 A comparison of ORR and the brute-force approach to compute the *left*C^{22}MP. The blue curve shows just the time to do the C22 feature extraction.

As the Fig 26 shows, ORR is extremely fast. By the time we consider 64,000 datapoints, it is ~185 times faster than brute force. Its throughput is about 880 Hz, which is faster than the arrival rate of almost all accelerometers/medical telemetry, etc. In the plot, it is difficult to tease apart the line for ORR, and the line for *just* the feature profile extraction step. In fact, about $^{2}/_{3}$rds of the time needed for ORR is in the feature extraction step. This suggests that it may be worth further optimizing that step. We first use the original authors MATLAB code [39], and found that with some simple optimizations, we could make it two orders of magnitude faster (our faster code is freely available [12]), we suspect further optimizations are possible.

2.6 Conclusion and Future Work

We have introduced a novel representation, $C^{22}MP$, and demonstrated that it provides state-of-the-art results for anomaly detection. It produces the highest published scores on the HEX/UCR benchmark, and it has been deployed in biomedical labs to aid research. Moreover, while $C^{22}MP$ is a generic TSAD technique, we have shown on the CCT dataset that it is able to beat domain specialized algorithms.

Here we confined our attention to just anomaly detection. In future work we plan to generalize the $C^{22}MP$ to some of the other MP primitives, including motifs, chains and snippets. This will not be trivial. Anomaly detection requires only the discovery of the *highest* values in the $C^{22}MP$, and that is amiable to early-abandoning search. The other primitives require finding the *lowest* values in the $C^{22}MP$, a much more difficult task to accelerate.

3. Discovering Multi-Dimensional Time Series Anomalies with *K* of *N* Anomaly Detection

In this chapter, we introduce an approach to tackle the problem introduced in section 1.2. An apparent solution for solving anomaly detection, when anomalies are only preserved in a subset of the problem, would be to search through all combinations of anomaly scores to find the subset that maximizes the anomaly score. However, it is clear that we must penalize subsets for their cardinality. In the example above, any anomaly score that we add to the correct two scores will slightly increase the maximum. In addition, it is important to realize that the nature of the problem precludes a greedy or dynamic programming search. In particular, under any reasonable anomaly score aggregation function, the highest scoring subset of size K is not necessarily a superset of the highest scoring subset of size K-1.

In this work we make the following contributions:

- We show that there are at least three different anomaly scoring aggregation functions that may be useful, depending on the circumstances.
- We demonstrate a novel search algorithm, TSADIS (Time Series Anomaly Detection through Incremental Search) that can discover the best K of N solutions, for all values of K from 1 to N, in just O($N \times LogN$) time. In contrast, a brute-force algorithm would take O(2^N).
- In many circumstances a user may wish to browse all K subsets, of size 1 to N. However, the user (which may be a person or an algorithm) may insist on selecting just one subset, the most *natural* subset that reflects the anomaly. Thus, we introduce a function that can rank and compare different sized subsets.

The rest of this chapter is organized as follows. In Section 3.1 we review our motivations and assumptions before considering related work. In Section 3.3 we introduce the necessary definitions and notations, then introduce our algorithms in Section 3.4. Section 3.5 contains an extensive empirical evaluation.

3.1 Motivation

Our work is predicated on the assumption that an anomaly will generally not exhibit itself on all the time series that monitor that system (at least not initially). Consider the following examples:

- A distillation column may be monitored by 1,000+ sensors. However, the overall system can be envisaged as being comprised of multiple subsystems. These subsystems may correspond to different physical regions in the column (i.e., trays at different levels), to different physical devices (i.e., pumps or valves), or to different logical processes (i.e., heat recovery or drainage). These subsystems may be weakly or strongly coupled [24].

In an ICU setting there may be as many as thirty sensors monitoring a patient's health. But most serious issues only manifest themselves on a subset of these time series, at least initially. For example, cardiac tamponade may present itself only on respiration and blood pressure. Hyperglycemia typically presents itself on respiration and glucometer.

3.2 Related Work

The topic of time series anomaly detection has seen a dramatic explosion of interest in recent years, as such it is a difficult area to survey in limited space. We refer the interested reader to [4][5][9][27][59][62] and the references therein.

There are two important points that we have gathered from our survey of the literature. The first is mostly due to a single paper [66], that forcefully suggests some of the apparent success of recently proposed algorithms may be questionable, due to severe problems with the commonly used benchmarks in this area. The second issue is noted in [15], which claims that a flaw in the most common evaluation metrics means that a "*random guess method can outperform state-of-the-art detectors*[5]". We do not weigh in on these issues, other than to state that we have not assumed the correctness of previous work, and have made an effort to avoid these issues in our work.

We have chosen to extend *time series discords* [59][70], to the *K* of *N* case, rather than one of the many other possibilities. The reason for this is that there is an increasing evidence that discords remain competitive with the state-of-the-art. Among the hundreds of time series anomaly detection algorithms proposed in the last two decades, only time series discords could claim to have been adopted by more than one hundred independent teams to actually solve a real-world problem. For example, a group of climatologists at France's UMR Espace-Dev laboratory use discords to find anomalies in climate data [55]. A team of researchers at NASA's JLP lab have applied discord discovery to planetary data, noting that "(*discords*) *detect Saturn bow shock transitions well*" [13]. There are several other time series anomaly detection algorithms that are well cited [10][22], but most of the citations are from rival methods comparing these algorithms on a handful of benchmarks [66].

3.3 Definitions and Notation

Here we introduce the necessary definitions and terminology, beginning with the definition of a *time series*:

Definition 1: A *time series* $T \in \mathbb{R}^C$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$: $T = [t_1, t_2, \dots, t_C]$, C is the length of T.

[5] At the time of writing this paper in on arxiv.org and is not peer reviewed. However, its claims seem irrefutable, and, in any case, we have independently confirmed them.

Typically we are not interested in global properties of a time series but rather shapes of small regions called *subsequences*:

Definition 2: A *subsequence* $T_{i,m}$ is a contiguous subset of values from T starting at index i with length m.

We can take any subsequence from a time series and compute its distance to all subsequences. We call an ordered vector of such distances a *distance profile*:

Definition 3: A *distance profile* D_i for time series T refers to an ordered array of distances between a given query subsequence $T_{i,m}$ and all subsequences in time series T.

As noted above, we are assuming that the distance is measured using the Euclidean distance between the z-normalized subsequences. This distance can be computed very efficiently using the MASS algorithm [58]. For a distance profile D_i of query $T_{i,m}$ the i^{th} position represents the distance between the query and itself, so the value must be 0. The values before and after position i are also close to 0, because the corresponding subsequences have overlap with query. We need our algorithm to ignore these trivial matches of the query and itself, and instead focus on *non-self matches:*

Definition 4: *Non-self match*: Given a time series T containing a subsequence $T_{p,m}$ of length m starting at position p and a matching subsequence $T_{q,m}$ starting at q, $T_{p,m}$ is a *non-self match* to $T_{q,m}$ with distance $d_{p,q}$ if $|p - q| \geq m$.

As we noted, our ideas and algorithms for combining individual anomaly scores into a multi-dimensional anomaly score are agnostic to the choice of individual anomaly scoring technique. However, for concreteness, and because there is increasing evidence that it is among the state-of-the-art, we will explicitly ground our ideas with time series discords. We can use the definition of non-self match to help define *time series discord*:

Definition 5: *Time series discord*: Given a time series T, the subsequence $T_{d,m}$ of length *m* beginning at position *d* is said to be a discord of T if $T_{d,m}$ has the largest distance to its nearest non-self match. That is, $\forall$ subsequences $T_{e,m}$ of T if M_D represents all non-self matching subsequences of $T_{d,m}$, and M_E represents all non-self matching subsequence of $T_{e,m}$, $min(d_{d,M_D}) > min(d_{e,M_E})$.

Although there are many ways to locate time series discords, the most effective methods exploit a proposed data structure called the *Matrix Profile* [70]:

Definition 6: A *Matrix Profile* (*MP*) of a time series T is a vector storing the z-normalized Euclidean distance between each subsequence and its nearest non-self match. Formally, *MP*$= [min(D_1), min(D_2), \dots, min(D_{C-m+1})]$, where D_i $(1 \le i \le C - m + 1)$ is the distance profile of query $T_{i,m}$ in time series *T*. The highest value of the *MP* is the time series discord.

We generalize the Matrix Profile to *multi-dimensional time series*, which we define as:

Definition 7: A *multi-dimensional time series* $T \in \mathbb{R}^{N \times C}$ is a set of co-evolving time series $T^{(i)} \in \mathbb{R}^{C}$: $T = [T^{(1)}, T^{(2)}, \dots, T^{(N)}]^T$ where *N* is the dimension of T and C is the length of T.

Fig 27 illustrates this notation with a toy dataset. Suppose we have a three-dimensional time series.

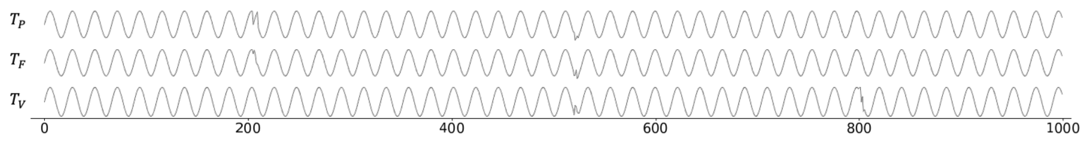


Fig 27 A toy three-dimensional time series that we will use as a running example. P = pressure, F = flowrate, V = viscosity.

We are interested in multi-dimensional time series that have anomalies that may be present on a subset of dimensions; thus, we call such anomalies a *K-dimensional anomaly*:

> **Definition 8:** A *K-dimensional-anomaly* (*KDA*) is an anomaly that is manifest on at least *K* time series.

The toy dataset in Fig 28 has three anomalies each occurring at a unique time. The anomaly marked in green is a 2-dimensional-anomaly (2*DA*) since it is present on at least two time series. The anomaly marked with blue is a 3-dimensional-anomaly (3*DA*) and finally the anomaly marked with red is a 1-dimensional-anomaly (1*DA*).

Notice that any iDA will also be a i-1DA. For example, a 3DA is also a 2DA and 1DA, and a 2DA is also 1DA.

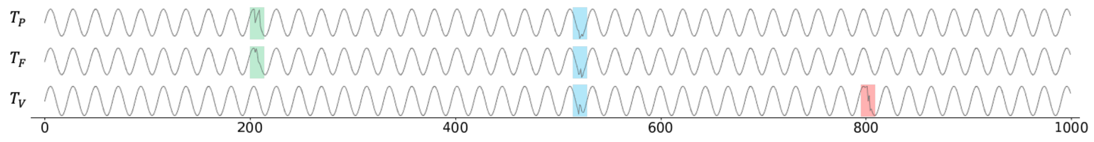


Fig 28 In this multi-dimensional time series we have three anomalies at index 205, 520 and 800 respectively. The anomaly marked by green is a 2DA, the second anomaly marked in blue is a 3DA and finally the anomaly marked by red is a 1DA anomaly.

We are ultimately interested in detecting the *natural anomalies* in a dataset:

> **Definition 9:** *Natural anomaly*: for a given timestamp t and the list of the *KDA*s at t, iDA is a natural anomaly if i is equal to anomalies' natural dimension.

The natural dimension of an anomaly is the maximum number of the time series that the anomaly is presented on. In our running example the natural dimension of the anomaly marked in green is two. The natural dimension of the anomalies marked in blue/red is three/one respectively.

Why are we interested in defining and finding the *natural anomaly*? It might be imagined that it is sufficient to simply declare that there was an anomaly at time t. However, anomaly detection is more actionable if we know *which* sensors are involved. For example, in petrochemical processing, a distillation column may be monitored by 1,000 sensors, and these may be spread over a 10,000 m^2 plant. If an anomaly is detected, the plant manager may need to dispatch a response team. Knowing which subsystem is experiencing failure could help her to quickly direct her team to the right location [19][61].

It is important to note that our natural anomaly definition makes no claim about causality or redundancy. For example, an over-pressurized boiler may produce an anomaly in a pressure^{Pa} time series, and that, in turn, may cause anomalies in both the temperature^{Fahrenheit} and temperature^{Celsius} time series[6]. We would expect the natural anomaly to discover an anomaly featuring these three traces. The task of discovering the direction of causality and the redundancy the two temperature measurements, is something for a downstream algorithm.

We will show in Section 3.4 how we can score *KDA*s. In Section 3.5 we show how we can distinguish the *natural anomalies* from other *KDA*s by comparing their scores.

[6] The example may seem frivolous. However, we have seen petrochemical datasets that record temperatures in both Kelvin and Fahrenheit. Moreover, because of rounding policies the correlation between these two time series was not 1.0

We have now defined the task-at-hand, to discover *K-dimensional-anomalies* and *natural anomalies*. We propose to do this by computing a Matrix Profile for each time series, and then "reasoning" about combinations of these *MP*s to see which combination is most likely to contain the anomaly.

The calculated Matrix Profiles for each time series in the *T* can be saved in an array called *all-matrix-profiles* (*MPs*):

Definition 10: *All-matrix-profiles* (MPs)*:* Given a multi-dimensional time series $T \in \mathbb{R}^{N \times C}$, all-matrix-profile *MPs* $\in \mathbb{R}^{N \times C}$ is an array containing the Matrix Profiles of all *N* time series in *T*. Formally, $MPs^{(i)} \in \mathbb{R}^{C}: MPs = [MP^{(1)}, MP^{(2)}, \ldots, MP^{(N)}]^{T}$.

Fig 29 illustrates the *MPs* for the toy dataset we are using as our running example.

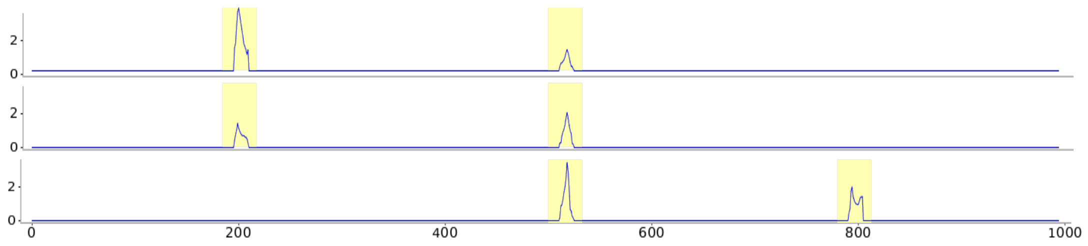


Fig 29 Matrix Profiles for the three time series in our running examples. Notice how the "bumps" reflect the anomalies in Fig 28.

Similar remarks apply to many other domains. In many cases, the majority of the data In an *N*-dimensional time series, an anomaly may present on 1 or 2 … or *K* time series (*K*≤*N*). Since we do not know in advance on which set of time series an anomaly is preserved, we need to (in principle) extract all the possible *anomaly-score sets*:

Definition 11: *Anomaly-score set* (S)*:* given an all-matrix-profiles $MPs \in \mathbb{R}^{N \times c}$ there are 2^N-1 possible combinations of the anomaly scores in MPs. Any possible combination of anomaly scores in MPs is an *anomaly-score set*.

We group all the anomaly-score sets of the same size in a list called *K-dimensional-anomaly-score*:

Definition 12: *K-dimensional-anomaly-score* (*KS*)*:* given an all-matrix-profile $MPs \in \mathbb{R}^{N \times c}$there are $\frac{N!}{(N-K)!K!}$ possible combination of anomaly-scores set of size *K*. The *KS* is a list of sets, the sets that contain all anomaly-scores with size *K*, where *K* > 0. Formally,

$$1S = [\{MP^{(1)}\}, \dots, \{MP^{(N)}\}]$$

$$2S = [\{MP^{(1)}, MP^{(2)}\}, \{MP^{(1)}, MP^{(3)}\}, \{MP^{(2)}, MP^{(3)}\}, \dots \{MP^{(N-1)}, MP^{(N)}\}]$$

$$3S = [\{MP^{(1)}, MP^{(2)}, MP^{(3)}\}, \{MP^{(1)}, MP^{(2)}, MP^{(4)}\}, \dots \{MP^{(1)}, MP^{(N-1)}, MP^{(N)}\}]$$

$$\dots$$

$$NS = [\{MP^{(1)}, MP^{(2)}, MP^{(3)}, MP^{(4)}, \dots, MP^{(N-1)}, MP^{(N)}\}]$$

Using the toy dataset as an example, the *K*-dimensional-anomaly-scores are as follows:

$$1S = [\{MP^{(P)}\}, \{MP^{(F)}\}, \{MP^{(V)}\}]$$

$$2S = [\{MP^{(P)}, MP^{(F)}\}, \{MP^{(P)}, MP^{(V)}\}, \{MP^{(F)}, MP^{(V)}\}]$$

$$3S = [\{MP^{(P)}, MP^{(F)}, MP^{(V)}\}]$$

To be clear, these are all the possible subsets of anomaly-scores, upon which an anomaly could manifest itself. Thus, if we wish to discover an anomaly that is present on, say, two dimensions, it must be one of the sets listed in 2*S* above.

The reader will appreciate that the *K*-dimensional-anomaly-score contains all the data we need to locate anomalies that are preserved on at least *K* time series.

While one could imagine many ways to mine a *K*-dimensional-anomaly-score and locate the appropriate *KDA*, we take a direct approach. We aggregate the anomaly-score sets (S) by taking the *min* value at each timestamp to generate the *Min Matrix Profiles* (MMP):

Definition 13: *Min Matrix Profile* (MMP): given an anomaly-score set $S \in \mathbb{R}^{K \times C}$, MMP is a vector storing the aggregated anomaly score of Matrix Profiles in S obtained by taking the min value at each timestamp. Formally, $MMP = \min_{j \in K} S_{i,j}$, $j = 1, \ldots, K$, where $1 \leq i \leq C$. The min method acts as an AND operator, so we would get a high value if and only if *all* Matrix Profiles have a high value at the given timestamp.

The high values (i.e., peaks) in the MMPs indicate the location of *KDA*s. Note that for a given *KS*, we have $\frac{N!}{(N-K)!K!}$ MMPs. Fig 30 illustrates this concept.

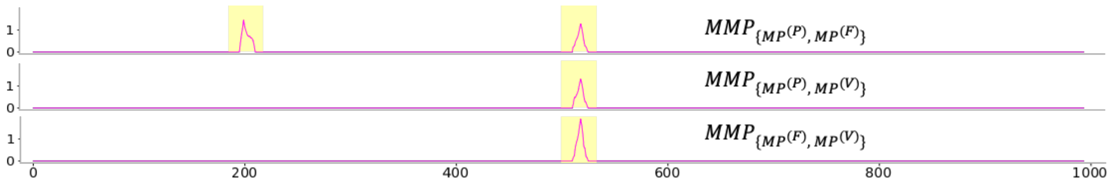


Fig 30 S The toy data's *2S* has three $\boldsymbol{MMP}$s ($\frac{\mathbf{3!}}{\mathbf{(3-2)!2!}} = \mathbf{3}$). Note that we only have peaks in the locations where anomalies are present on at least two dimensions. Moreover, $\boldsymbol{MMP}_{\{\boldsymbol{MP}^{(P)}, \boldsymbol{MP}^{(V)}\}}$and $\boldsymbol{MMP}_{\{\boldsymbol{MP}^{(F)}, \boldsymbol{MP}^{(V)}\}}$ give us a partial view of *2DA*'s location.

Each MMP gives us a partial view of the *KDA*s' location. In order to find *all possible KDA*s we take max function over all MMPs at each timestamp and generate a single vector to show the location of anomalies for that *KS*. We call this vector *KD-profile*:

Definition 14: *KD-profile* (*KDP*): given the list of the MMPs of a *KS*, *KDP* is a vector storing the aggregated anomaly score of the MMPs obtained by taking the max value at each timestamp. Formally,

$KDPs = \max_{j \in K} MMPs_{i,j}$, $j = 1, \ldots, K$, where $1 \leq i \leq C$.

The max method acts somewhat like an OR operator, so we will get a high value if *any* of the MMPs has a high value in the given timestamp. Fig 31 illustrates this notation.

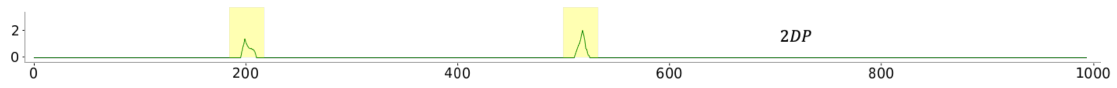


Fig 31 The two-dimensional-profile (2*DP*) of the 2*S* of our toy example. Notice that 2DP has peaks only where data has 2*DA* and there is no peak at index 800 where data has 1*DA*.

The calculated *KD*-profiles for all *K*-dimensional-anomaly-scores are saved in an array called *all-KD-profiles* (*KDPs*):

Definition 15: *All-KD-profiles KDPs* $\in \mathbb{R}^{N\times C}$ is an array, containing *KD*-profiles of the *K*-dimensional-anomaly-scores for $K = (1, 2, \dots, N)$.

Given the above definitions, we are now in a position to formalize our two problem statements.

Problem Statement 1: Given an *N*-dimensional time series *T*, a user selected subsequence length *m,* and selected scoring method, find the most significant *KDA*s for *K*= 1, *K* = 2, … , *K* = *N*. The output is a list *L* of length *N*, where each element of the list consists of a triple containing: the *location* (timestamp/index) of the *KDA*, the *set* indicating which time series are contributing to the *KDA*, and the last value is the *significance* score of the *KDA*.

We can refer to the three elements of the ith list item with $L_{i.location}$, $L_{i.set}$, and $L_{i.significance}$.

It is important to recognize that, in general, the elements of the list do not have to be *nested*. For example, we may have $L_{1.set}$={pressure}, and $L_{2.set}$={flow-rate | viscosity}. The fact that we do not require the nesting property is important to allow full expressiveness of representation, but it unfortunately does preclude certain search mechanisms for efficiently computing *L*, including branch and bound or dynamic programming.

Depending on the downstream application, a user can explore all *N* elements in the list *L*, or they can ask for only the natural anomalies in *L*:

Problem Statement 2: Given the list *L* of *KDA*s, return the most natural anomaly. Recall the example used in Fig 3. The true anomaly is $L_{2\text{-}set}$ ={1|2}. If we measure maximum 2*DA'* score on any *two* dimensions only in this dataset, the score of 8.33 reflects the embedded anomaly on {1|2}. Suppose we insisted on finding any anomaly on only one dimension. In this case maximum 1*DA'* score reflects time series {2} with a lower score of just 6.9. In the other direction, suppose we insisted on finding any anomaly on only three dimensions. In this case the set returned is {74|18|20}, and we know from our creation of the data that these time series are spurious. Critically, however, note that the *3DA*'s score has decreased to 0.

Thus, our definition of *KDA*'s score allows a simple and obvious direction of detecting the "*natural anomalies*".

3.4 Algorithms

In this section, we introduce the algorithm to address the problem statements described above.

3.4.1 KDA Detection Algorithm

In the previous section we defined what we wish to compute. Here we discuss algorithms to actually compute these definitions. For concreteness, we begin with the brute-force algorithm to compute the *KDP*s as shown in Table 9.

Table 9 Brute-Force Algorithm to compute *KDP*s

```
Function:  Brute_Force_KDP (T, m)
Input:
   T: {Array-like} of shape (n_samples, n_time_series)
     Multi-dimensional Time series
   m: int
     Subsequence length
Output:
   KDPs: {Array-like} of shape {n_samples, n_time_series}
     All-kd-profiles
1   N = T.shape[1] // get the number of time series in T
2   // calculating all-matrix-profile (def. 10)
3   MPs=empty_like(T)
4   For i, ts in enumerate(T):
5      MPs[:,i] = matrix_profile(ts, m)
6   // get all possible set of combination of time series
7   C = get_combinatiosn(N)
8   // get all the K-dimensional-anomaly-scores (def. 12)
9   A = defaultdict(list)
10  For this_set in C:
11     k= len(this_set) // get the size of the set
12     S = MPs[:,this_set] // get the anomaly-score set (def. 11)
13     A[k].append(S)
14  // get all-KD-profiles (def. 15)
15  KDPs = []
16  For k, v in A.items():
17     MMPs = [] // calculate the MMPs (def. 10)
18     For S in v:
19        MMP = min(S, axis = 1)
20        MMPs.append(MMP)
21     KDPs.append(max(MMPs, axis = 1))
22  Return KDPs
```

In line 1 we obtain *N*, the number of time series in multi-dimensional time series *T*. Note that number of time series is equivalent to the maximum possible dimensions that an anomaly can manifest itself on. We iterate through all the time series in *T* and calculate their Matrix Profile to make the all-matrix-profile array in 4 to 5. A list containing all the possible combinations for a list with size *N* is calculated in line 7. We use this list to query all the possible anomaly-score sets from the all-matrix-profile array. We save the anomaly-score sets with respect to their sizes in a dictionary in line 10 to 13. Then we iterate through each *K*-dimensional-anomaly-score to generate its

KD-profile in lines 15 to 22. When calculating *KDP*s we also save the indices of time series contributing to each *KD*-profile.

Once we calculate the *KDP*s, the detection of *KDA*s is trivial. The peaks on each *KDP* indicate the location of a *KDA*. Fig 32 shows an example of the output of this algorithm on the toy example. Note that they are sorted from top to bottom. The top figure shows the 1*DP*, notice that it is the most "busy reflecting the fact that it shows all anomalies that appear on even a single dimension.

The second from the top figure shows the 2*DP*, note that it is less "busy", as there are (generally) fewer anomalies that appear on (at least) two dimensions, and so on.

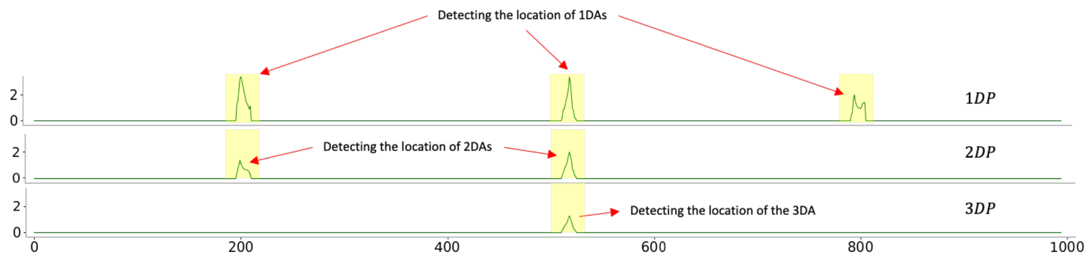


Fig 32 The all-*KD*-profile for the toy dataset. The peaks in each *KDP* represent the location of *KDA*s.

The information in the *KDP*s array can be seen as an index that can be used by downstream algorithms to interpret results and rank *KDA*s.

For example, it can be used to answer questions like "What is the most significant anomaly that appears on exactly three dimensions?". In Table 10 we will present a simple explicit algorithm to allow the user to answer such questions.

Table 10 Algorithm to compute Query *KDP*s

```
Function: Query_KDP (KDPs, KDP_idx, method, th)
Input:
  KDPs: {Array-like} of shape {n_samples, n_time_series}
    All-kd-profiles
  KDP_idx: {Array-like} of shape {n_samples, n_time_series}
    indexes of the time series contributing to KDPs
  method: {'min', 'sum, 'mean'}, default= 'sum'
    method to score anomalies
  k: int        Number of the dimensions of the anomaly
  th:           Threshold to detect anomaly, default =0
Output:
  KDA: tuple
    a triple (X,Y,S)containing:
    X: int              The timestamp/index of the KDA
    Y: array of int     The name/index of time series
    S: float            Significant of KDA
1   N = KDPs.shape[1] // get the number of the time series
2   //get the location of the KDA that are on exactly K dimensions
3   if k < N-1:
4     x = set(where(KDPs[:,k]>th)-set(where(KDPs[:,k+1]>th))
5     x = list(x)
6   else:
7     x = where(KDPs[:,k])
8   y = KDP_idx[x,:k+1] //get time series' contributing to KDA
9   s = KDP[x,:k+1] // get anomaly score from each dimension
10  // get the scores for each candidate KDA
11  if method == 'min'
12    score = s[:,k]
13  if method == 'sum'
14    score = sum(s[:,:k+1],axis=1)
15  if method == 'mean'
16    score = mean(s[:,:k+1], axis=1)
17  top = argmax(score)//get the idx of the KDA with highest score
18  most_significant_kda = (x[top], y[top], score[top])
19  return most_significant_kda
```

The number of the time series is extracted in line 1. Lines 3 to 7 extract the indices/timestamps of the *KDA*s that are preserved only on the *K* time series. The indices of the time series that contributed to the *KDA*s are extracted in line 8 using *KDP_idx* array. *KDP_idx* contains the indices of each time series that contributed to *KDP*s. The anomaly scores from each *KDP* are saved in array s. This information is used in lines 11 to 16 to calculate the *KDA*'s score using a user selected scoring method. Line 17 shows how the index of the most significant *KDA* is obtained. Finally, the *location* (timestamp/index) of the most significant *KDA*, the *set* contributing to it, and its significance *score* are returned.

For example, if we want to query the *KDP*s array shown in Fig 32 with "*What is the most significant anomaly that appears on just two dimensions?*", the output would be (200, {0,1}, 4.53). Note that in these examples we envision the queries coming directly from an end-user, however, these queries may also come from a downstream algorithm.

Depending on the downstream application different scoring methods can be used. In this work we introduce three different scoring methods: min, sum, mean. In sum the scores from each *KDP* contributing to the *KDA* are summed together. We can normalize the sum score for a given anomaly by dividing it by the amount of dimensions that given anomaly has. This is the exact approach of the mean method. With the mean method we penalize a subset for their cardinality. Finally, in the min method, the score for each *KDA* is equivalent to the anomaly score on the corresponding *KDP*. For example, the min score for a 2*DA* is the value of the peak on 2*DP* where that 2*DA* was located. We call this the min method since it reads the scores from the Min Matrix Profiles building up the *KDP*.

The reader will appreciate that the algorithm in Table 3 is only $\mathrm{O}(n)$ where n is the length of the timeseries, however Table 1 requires $\mathrm{O}(n \times 2^N)$ memory and $\mathrm{O}(n \times 2^N)$ time. This memory requirement is inconsequent, but the time requirement is intractable for all but the smallest time series. In the next section we will show how we can exploit sorting to dramatically improve the scalability of this algorithm.

3.4.2 Fast KDA Algorithm

In the previous section, we used the brute force algorithm to detect *KDAs*. As we will later show, it is extremely effective at detecting *K* of *N* anomalies, but there is a problem: it is simply too slow to compute. For example, if we have twenty time series with a length of 1,000, it takes about fifteen minutes. This may be tenable, but with just fifty time series of that same length it takes several millennia [63].

However, we have a simple way to make it faster. It is based upon the following observations. We do not actually need to enumerate and score every combination of *MP*s. That approach *does* solve the task-at-hand but produces spurious information and requires many redundant calculations. Instead, we can exploit the independence of values at each time point. Consider a single time point i on the time series. We can look at the values of all the Matrix Profiles at location i and sort them into a list. Then the value of the 1*DP* at the i^{th} location is just the largest value in our sorted list, the value of the 2*DP* at the i^{th} location is the second largest value in our sorted list, and so on. Table 11 formalizes this idea.

Table 11 TSADIS: Fast KDP Algorithm

```
Function:  Fast_KDP (T, m)
Input:
   T: {Array-like} of shape (n_samples, n_time_series)
     Multi-dimensional Time series
   m: int                Subsequence length
Output:
   KDPs: {Array-like} of shape {n_samples, n_time_series}
     kd-profiles
   KDP_idx: {Array-like} of shape {n_samples, n_time_series}
     The indexes of the time series contributing to KDPs
1  // calculating all-matrix-profile (def. 9)
2  MPs=empty_like(T)
3  For i, ts in enumerate(T):
4    mp = matrix_profile(ts, m)
5    MPs[:,i]=mp
6  // get all-kd-profiles (def. 15)
7  KDPs, KDP_idx = sort(MPs, axis= 1, order = descending)
8  return KDPs, KDP_idx
```

In lines 2 to 5 we iterate through every time series and calculate their Matrix Profile. We sort the values in the *MPs* across the y-axis in descending order. We also save the indices that sort the *KDP*s in an array of the same size as *KDP*s.

3.4.3 Robustness to Noise

In real-world applications, multi-dimensional time series data often contains noise (see Fig 34). When noise becomes significant, it may effect anomaly detection models. We have based our algorithm on the Matrix Profile, which is a vector recording the smallest z-normalized Euclidean distance of each subsequence of a time series to all other subsequences. Therefore, if data is noisy, all the distances will generally become higher. We can remove this added base value to all distances in Matrix Profile before calculating the *KDP*s to suppress the high anomaly score caused by the noise. We estimate this base value by calculating the 75th percentile of the distances in Matrix

Profile. This is a very intuitive parameter. We could also achieve essentially the same result just by *smoothing* the time series, before computing the Matrix Profile.

3.4.4 Online KDAs Detection

We have demonstrated the ability to detect *KDA*s in the previous section, however, we assumed that the entire time series was available. Here, we show how to detect *KDA*s online without that assumption. When working with streaming data, we need to take the new incoming single data point and compare its subsequence with the rest of the time series, compute the distance profile for this subsequence and update the existing Matrix Profile. This can be accomplished using the algorithm introduced by [70],which maintains the Matrix Profile in an incremental fashion by taking the existing data *T* and calculating Matrix Profile *MP*. When a new data point, *t*, arrives it appends *t* to *T* and compares the new subsequence with all extant subsequences and updates the historical values. Further, it determines which of the existing subsequences is the nearest neighbor to the new subsequence and appends this information to the Matrix Profile, which continues as additional data stream in. Note that incremental Matrix Profile is different from batch Matrix Profile since it does not waste time re-computing any past pairwise distances, it only computes new distances and updates the appropriate arrays where necessary, making the algorithm very fast. This algorithm can be used to keep track of the extreme values of an incrementally-growing Matrix Profile and report a new discord when there is a new maximum value. This implies the ability to extract *KDA*s in streaming data by using incremental Matrix Profile for generating *KDP*s.

3.5 Empirical Evaluation

We have designed all experiments such that they are easily reproducible. To this end, we have built a webpage [63] that contains all datasets, code and random number seeds used in this work. This philosophy extends to all the expository examples in the previous sections. In our comparisons to other systems, unless otherwise stated, we take numbers directly from the original papers, rather than reimplement the approaches. This is not laziness on our part. Many of these approaches are very complex, and it would be difficult to ensure that we have reimplemented them to the authors' satisfaction. This ensures that we are comparing to the best possible implementation, on data that the authors think suits their approach.

3.5.1 Preamble: Metric of Success

Several recent papers have suggested that many techniques to evaluate anomaly detection algorithms are flawed [66] .For example, Kim et. al. show that a commonly used scoring metric would highly rank an algorithm that simply randomly guesses [54]. Keogh argues that many papers report results with unwarranted and misleading precision [56].For example, suppose in a year of data, all of Xmas day is an anomaly because the sensor was turned off for maintenance. The correct way to report successes here would be binary, either 0/1 or 1/1. However, many papers report successes based on the sampling rate, something like 1440/1440. Or the algorithm may miss one datapoint, allowing the author to report 0.9993. There is a huge difference between the intellectually honest 0/1 and 0.9993. The latter implies an incredible precision that is just not warranted. The problem is compounded by the fact that in most cases it is impossible to say *exactly* where an anomaly begins or ends.

With this background in mind, we have taken great care to design a scoring function that produces sensible output, with only appropriate precision reported, and for which a random guess would score very poorly.

- If the ground truth for the anomaly is given as a region $T_i : T_j$, we score a prediction *correct* if it is anywhere within the range $T_{i-m} : T_{j+m}$. The reason for the "bracketing" of the region is that for some algorithm's prediction *P*, the location of the most anomalous region, *P* may be reporting the *beginning*, or the *end*, or the *middle* of a subsequence. Our scoring function will not penalize for this trivial detail.
- If an algorithm predicts that a set of *K* time series has an anomaly at location *P*, we count as a success each time that a prediction was true. For example, if we predict there was an anomaly at time *P* on time series {2,5,9}, and we examine the ground truth to discover that only {2,9} really had anomalies at *P*, we report our success as 2/3.

We believe that this metric of success is fair and intuitive. However, we note that the excellent results of TSADIS below is also apparent with other common metrics of success.

3.5.2 Comparison to MSCRED

Perhaps the most cited recent paper on multi-dimensional time series anomaly detection in recent years is [73]. Here the authors introduce a Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) and demonstrate its effectiveness on the dataset shown in Fig 33.

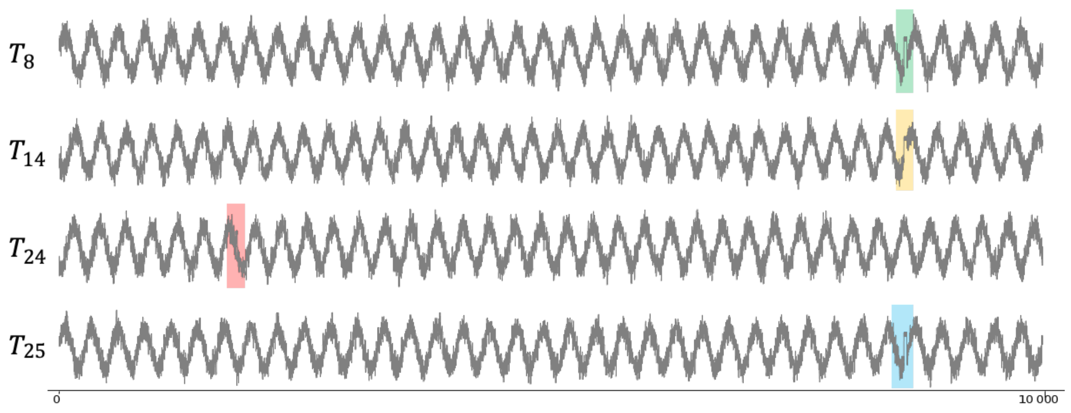


Fig 33 Four (out of thirty) sample time series from MSCRED dataset.

This dataset is designed to be challenging, with sinusoidal waves that differ in phase and frequency, and with "shock wave like" anomalies at different scales modeling three different types of root causes randomly inserted. Some of the anomalies are obvious, but some (see T_{24} in the above) are very subtle.

To evaluate TSADIS on this dataset we ask the following question of the data: *Find the Top-5 anomalies in Three Dimensions*. Table 12 summarizes the results.

Table 12 The results of TSADIS on the MSCRED dataset.

ID	Ground Truth	TSADIS Prediction	TSADIS Score	Default Rate
A	11810, {24,15,28}	11744, {24,28,15}	3/3	0.012
B	12760, {21,26,5}	12715, {26,21,5}	3/3	0.012
C	14540, {3,16,2}	14418, {16,2,3}	3/3	0.012
D	17790, {**9**,5,20}	17682, {**12**,20,5}	2/3	0.012
E	18620, {25,14,8}	18546, {8,25,14}	3/3	0.012

These results are almost perfect, with a single error made on the case "D". The original authors use a different metric of success, average recall over five runs, obtaining 0.8. If we use *that* metric of success, we score 0.93.

Beyond the significant improvement in effectiveness, there are several other reasons to prefer TSADIS over MSCRED. The MSCRED approach required significant training data to achieve its results. In contrast, we can use TSADIS with *zero* training data. If we do so, our performance drops a little to 0.86, but we still beat MSCRED.

MSCRED is a very complicated approach, featuring a fully convolutional encoder, an attention based ConvLSTM, a temporal attention mechanism, a mini-batch stochastic gradient descent and several other elements. It is difficult to be sure how many parameters must be tuned here, but it seems to be at least twelve. Moreover, because the output of the algorithm is stochastic, it takes significant effort to understand the effects of the parameters on the performance. In contrast, TSADIS only requires the setting of a single parameter, *m*. Here we set *m* to 250, which was, by visual inspection, a "whole" number approximately equal to the average period length. However, for examples B and C, we could have set *m* to any value between 150 and 650, a huge range, and still have obtained perfect results. The other examples have a slightly smaller range, but are still robust to *m*. In fairness, this robustness to the choice of *m* is a property of the Matrix Profile [59][70]that we inherit.

3.5.3 Comparison to Isolation Forest|AE-LSTM|Prophet

Another recent paper considers a real-world multi-dimensional time series dataset of photovoltaic (PV) systems [29], and compares three state-of-the-art approaches: AutoEncoder Long Short-Term Memory (AE-LSTM), Facebook-Prophet, and Isolation Forest. Examples of the data are shown in Fig 34.

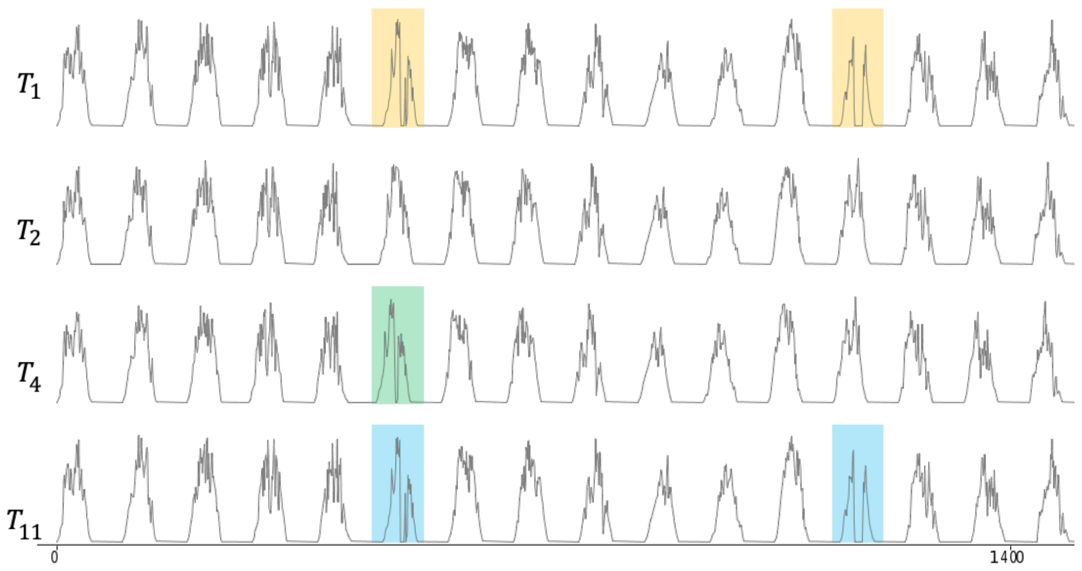


Fig 34 Four sample time series (out of twenty-two) selected from the photovoltaic (PV) systems dataset. Sixteen days are used.

The authors of this study actually solve an easier problem than the more general task we consider. They essentially ask: For a given *single*-dimensional time series, assuming we know it has an anomaly, can we detect where it is? Of the three approaches compared, only AE-LSTM obtained a perfect score. In contrast, we ask the more difficult multi-dimensional questions. *Find the Top-1 anomalies in K dimensions*. There is an objective ground truth provided only for the cases *K* is 1, 2, and 6, Table 13summarizes the results.

Table 13 The results of TSADIS on the photovoltaic dataset.

ID	Ground Truth	TSADIS Prediction	TSADIS Score	Default Rate
A	465, {0,4,11,16,18,19}	470, {0,4,11,16,18,19}	6/6	0.054
B	1140, {0,11}	1171, {0,11}	2/2	0.054
C	1320,{20}	1333,{20}	1/1	0.054

Here we achieved perfect results, again completely ignoring the training data that the three other methods relied upon. In this dataset dawn to dusk is about 60 datapoints, so we set *m* = 60. However, if we had set *m* to any value in the range 50 to 130, we would still have gotten perfect results.

3.5.4 Comparison to MGAB

In a recent work [62],the authors note some frustration with the community confining their interest to datasets containing anomalies that are readily apparent to the naked eye. This observation is an independent confirmation of the "triviality" argument of claims of Wu and Keogh [66]. However, the authors of [62], explicitly take action to redress the problem, by creating a dataset where the "*anomalies are for the human eye very hard to distinguish from the normal (chaotic) behavior*". Fig 35 suggests that they were successful.

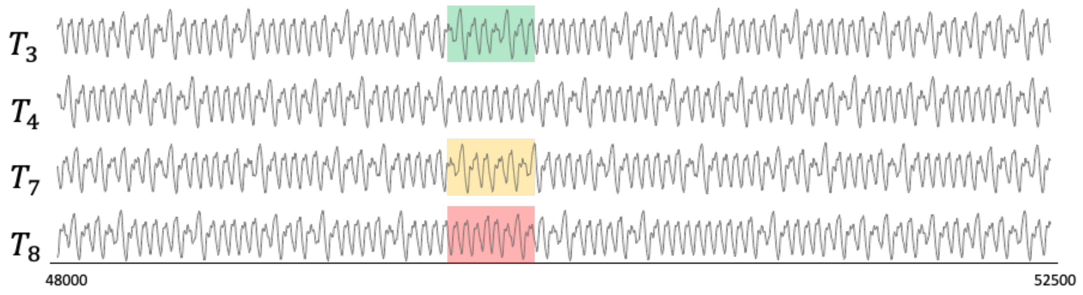


Fig 35 Four sample time series (out of ten) selected from the MGAB dataset. Only 4,500 datapoints (out of 97,600) are shown.

As with the previous example, the authors consider an easier problem than the more general task we consider, asking, for a given *single*-dimensional time series, can we detect where it is? We again will ask the harder multi-dimensional questions. *Find the Top-1 anomalies in K dimensions*. We do this for $K = 4$ and $K = 5$. Table 14 summarizes the results.

Table 14 The results of TSADIS on the MGAB dataset.

ID	Ground Truth	TSADIS Prediction	TSADIS Score	Default Rate
A	84614, {0,1,2,5,6}	84697, {2,5,1,0,6}	5/5	0.082
B	49765, {2,3,7,8}	49802, {8,7,3,2}	4/4	0.082

Here we achieved perfect results. In the original paper, the authors compare to five anomaly detection methods (DNN-AE, LSTM-ED, NuPIC, LSTM-AD, TCN-AE

[62]). As explained above, results are not exactly commensurate, however it is interesting to note that all five methods used training data, yet none was able to obtain perfect results. In our experiments we do include the training data (or rather, we make no effort to exclude it), but we in no way delineated it or labeled it as training data. Yet we were able to obtain perfect results.

3.5.5 NeurIPS Benchmark

One of the most frequently cited recent papers on time series anomaly detection in recent years is [35]. Here the authors revisit time series anomaly definitions and benchmark the synthetic criterion and the existing algorithms with a behavior-driven taxonomy. They propose five types of anomalies: global (point), Contextual, Shapelet, Seasonal and Trend. Following the new taxonomy, they generate 35 synthetic datasets. Specifically, they adopt sinusoidal wave to generate 20 univariate sequential data with different ratio of anomalies, where each dataset only includes one kind of anomaly. Then, they also generate fifteen multivariate sequential data which combine different kinds of anomalies into single dataset. Fig 36 shows one example of the multivariant datasets in this benchmark.

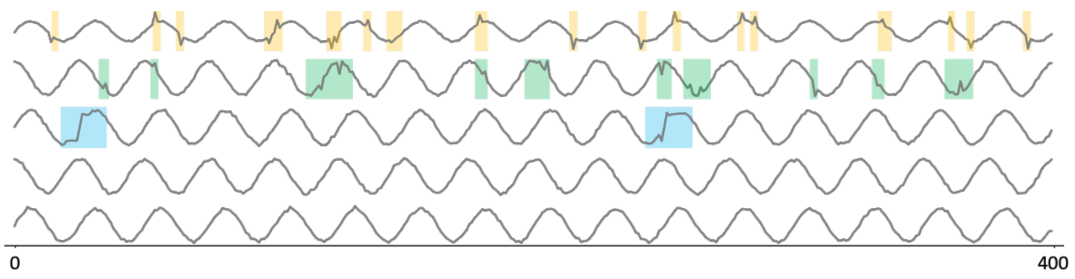


Fig 36 One example of the multi-dimensional time series in NeurIPs benchmark. Three type of anomalies is injected in this data: Global anomaly in first dimension, Contextual anomaly in second and Shapelet anomaly in the third dimension.

They benchmark nine state-of-the-art anomaly detection algorithms on these fifteen multivariant datasets using the F1 score. Fig 37 (*left*) is a screengrab from their paper that shows this result.

We repeated the same experiment for TSADIS and brush our result (pink line) onto the author's original plot, taking every care to make sure the experiments are commensurate. As you can see in Fig 37 (*right*) TSADIS outperforms all other rival methods.

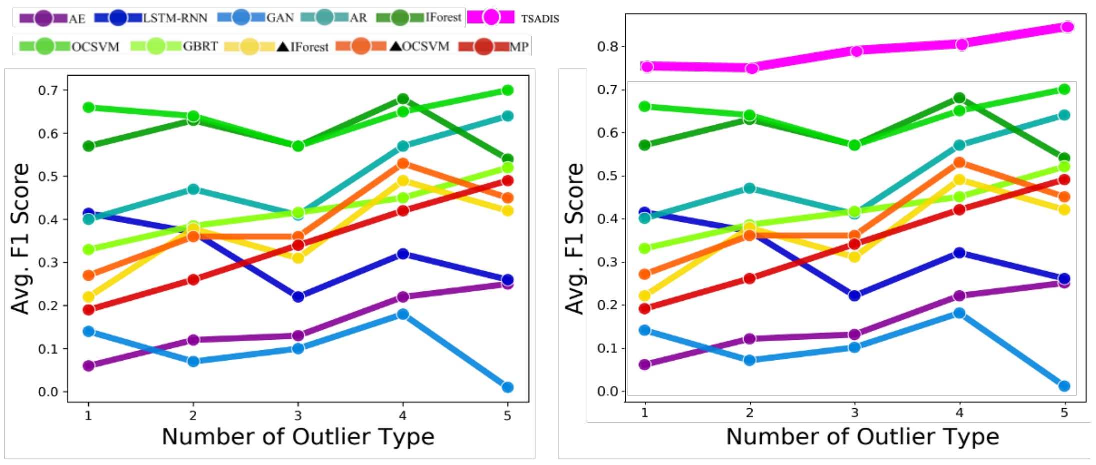


Fig 37 A screen captured from [35].*Right*) we repeated the benchmark experiment for TSADIS and brush our result (pink line) onto the author's original plot. As we can see TSADIS outperform all the rival algorithms (Autoencoder(AE), recurrent neural networks with long short-term memory units (LSTM-RNN), Generative adversarial network(GAN), Autoregression (AR), Isolation forest (IForest), One-class SVM (OCSVM), Gradient boosting regression (GBRT), Matrix profile (MP), ΔIForest, ΔOCSVM are subsequence clustering).

The original paper was a little vague on some details of how they evaluated the algorithms. Thus, to produce this result, we used some reasonable and typical assumptions about how to calculate the F1 score. However, under any assumptions, we are still highly competitive. The interested reader can read [63] for more details.

Note that one of the algorithms in this benchmark is MP, referring to Matrix Profile. *Our* Matrix Profile-based result is much better. Normally, one might attribute this to a suboptimal choice of subsequence length, however, that's not the case here, since the authors explain that they search for the best subsequence size and only report the best result. The other reason, that seems to be more likely, is that the combination method that they used is naïve. We reviewed their code and realized they use the approach suggested in [69] to, first, generate the multi-dimensional matrix profiles and then combine them by addition. Therefore, it seems the excellent result we achieve is *solely* due to the novel approach we use to combine Matrix Profiles, providing more evidence for the utility of our ideas.

To summarize this section, in Fig 37 (*right*) we show that TSADIS outperforms ten state-of-the-art rival methods. Moreover, we did not choose (and possibly cherry-pick) the data that was used, and we did not implement (and possibly badly implement) the ten rival methods. As such, this experiment offers forceful evidence that TSADIS produces state-of-the-art results for multi-dimension anomaly detection.

3.5.6 Sensitivity to Additional Dimensions

In the experiments on benchmark datasets, we used *all* the available dimensions. It is natural to ask what would have happened if there were even more dimensions that did not have anomalies. At some point these additional dimensions will surely confuse our algorithm, but for *how many* spurious dimensions could we allow and still produce useful results?

To test this, we repeated the experiments on MSCRED, photovoltaic and MGAB datasets, with an increasing number of additional dimensions, until our results got worse. It is important that the extra dimensions come from the same domain, as our algorithm could survive the addition of thousands of dimensions that contain pure random noise. For MGAB, we used the authors' original data generator. For photovoltaic, we did not have additional data, so we used sine waves with a one-day period and the same noise level as the original data. For MSCRED, we used a technique similar to what the original authors used to generate the dataset. Table 15shows the results.

Table 15 The results of the increasing dimension test

Dataset	**#Dim added before failure**	**Failure case**	**Default Rate at failure**	**Dataset**
Photovoltaic	38	{20}	0.019	Photovoltaic
MGAB	11	{0,1,2,5,6}	0.039	MGAB
MSCRED	4	{24,15,28}	0.011	MSCRED

Note that the default rate decreases as we add dimensions, as there are more regions to make an incorrect prediction. In general, these results suggest that our algorithm is robust to spurious dimensions.

3.5.7 Timing Results

We have the luxury of being able to tersely summarize the time overhead for our algorithm. The time taken is dominated by the time needed to compute the Matrix Profiles, but this must be done by any approach that uses the Matrix Profiles. Thus, the only question is, how much *overhead* does our *K* of *N* approach incur? The answer, for all experiments in this work, is less than 5% (Concretely, for MSCRED it is 0.17%, for MGAB 0.03%, and for photovoltaic is 3.7%). Moreover, because we have based our algorithm on the Matrix Profile, we can take advantage of an active community that is constantly accelerating the Matrix Profile. For example, [39] allows the Matrix Profile to be computed on datasets with billions of datapoints on commodity desktop machines.

3.5.8 Selecting the Right K

In the experiments above, we evaluated our algorithm's accuracy, *given* that the user requested the correct value for *K*. Here we will evaluate our algorithms to predict the correct *K*.

As shown in Fig 38 we created a dataset that comprises of sine waves. We can add a simple anomaly to a time series, by taking the absolute value of a single period, and we can add noise to the time, which we measure in terms of a standard deviation of the original time series.

FIG1example, which is shown in Fig 38.

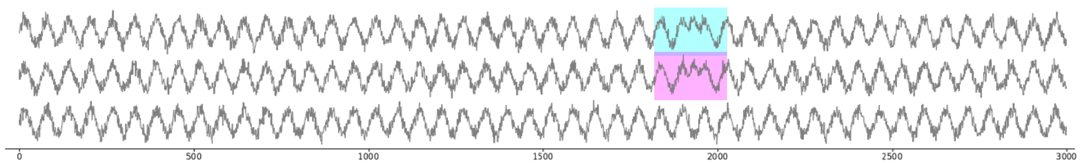


Fig 38 Three sample time series (out of ten) selected from the created dataset. Only 3,000 datapoints (out of 10,000) are shown. These examples are at noise level $\mathbf{1.5} \times \boldsymbol{std}$.

We performed the following experiment. We created ten different ten-dimensional datasets, varying the number of dimensions that contained an anomaly from one to ten. We then tested to see if we could recover the correct number of anomalies, by plotting the *KDA*'s score (using the sum function) for all possible values of *K* from 0 to 10.

In order to count our prediction as correct, we insist that our algorithm must correctly predict:

- How *many* anomalies there are (what is true *K*?)
- On *which* of the ten time series there is an anomaly.
- The *location* of the anomaly.

Fig 39 shows the predicted number of anomalies for each case, when we have a noise level of $0.1 \times std$.

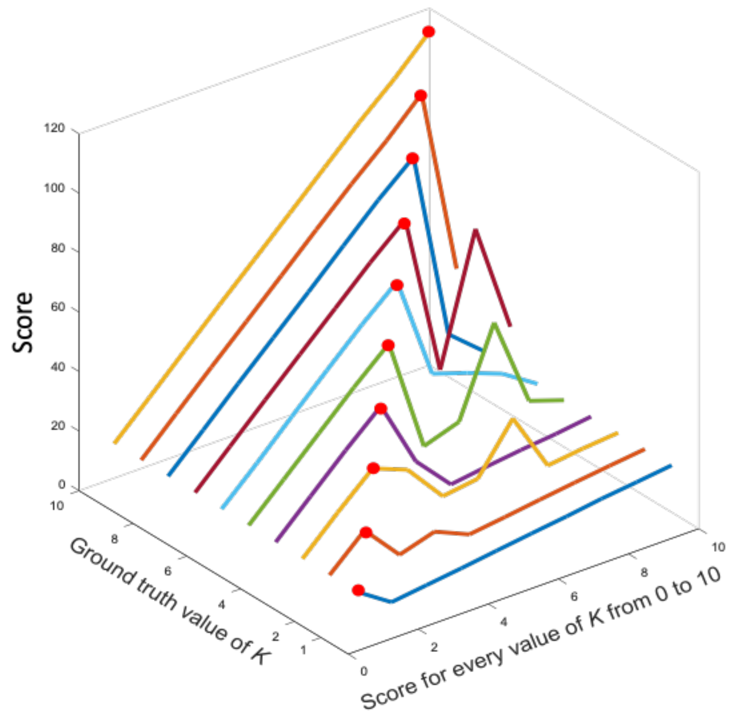


Fig 39 Our predicted number of anomalies (red dots) for datasets with a known ground truth number of anomalies that ranged from 1 to 10.

Given our success here, we can now ask how well we would do in the face of increasing noise levels. We repeated the experiment above, just for the case where the ground truth value of *K* was five. We did this ten times, and we successfully predicted *K* = 5 in each of the ten runs.

As shown in Fig 40, we repeated the entire process for increasing amounts of noise, until we had at least one failure. As it happens, for noise level $1.5 \times std$ we had two failures out of the ten runs.

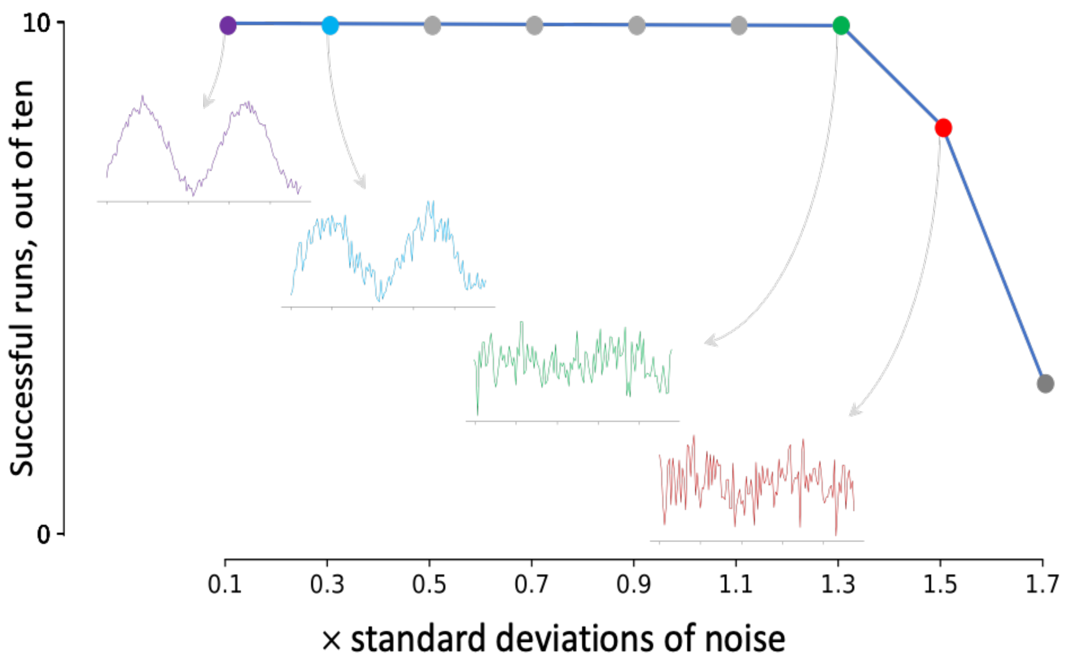


Fig 40 Testing our algorithms ability to predict the true number of time series that are involved in a five-dimensional anomaly in the ten-dimensional datasets, for increasing levels of noise.

In Fig 40.*inset* we show examples of the noise level at each setting. The reader will appreciate that the amount of noise our algorithm can tolerate before failure (shown in green), is considerable. To be fair, we are inheriting this robustness from the Matrix Profile, which is robust to noise [70].

3.6 Conclusions and Future Work

We have shown that at least one state-of-the-art anomaly detection algorithm, the Matrix Profile, will generally fail if it is forced to consider all *N* dimensions of a *N*-dimensional time series. By casting the problem as a *K* of *N* anomaly detection we both improve the accuracy of anomaly detection and attribute the anomaly to the correct set of responsible time series. We compare the results of our algorithm on three datasets/approaches, showing, in each case, that we could match or improve upon the original author's approach despite framing the problem in a way that makes it more difficult for ourselves, for example, by completely ignoring the training data. We have made all code/data available to allow the community to confirm, exploit and expand our findings. We also compared TSADIS with nine state-of-the-art anomaly detection algorithms benchmarked by NeurIPS Benchmark, demonstrating that our proposed solution outperforms all rival methods.

4. Conclusions

In this thesis, we introduced C^{22}MP, a novel data structure that presents numerous opportunities for innovative data mining applications. We introduced the concept of "*discordia*" and elucidated its distinctions from "*discord*" in time series analysis. We explained that a time series *discordia* is a subsequence uniquely characterized by its features, as opposed to a time series *discord*, which pertains to subsequences with distinctive shapes. We demonstrated that C^{22}MP enables us not only to detect time series discords but also to identify time series *discordia*.

Additionally, we introduced the ORR algorithm, a fast, efficient, and interpretable method for generating a C^{22}MP profile and detecting *discordia* within a time series. Notably, the ORR algorithm has found its initial application in biomedical research labs, aiding in the detection and comprehension of factors influencing the onset of Parkinson's disease.

We evaluated the accuracy and performance of C^{22}MP in detecting anomalies using twenty datasets from twenty different papers. Furthermore, we employed the UCR hexagon benchmark dataset, which includes 250 datasets, to compare the ORR algorithm with twelve state-of-the-art anomaly detection methods. Our results demonstrated that our proposed algorithm outperforms all rival approaches. We also discussed the potential for using C^{22}MP to detect other Matrix Profile primitives, including motifs, chains, and snippets.

In Chapter one, we addressed the primary challenges of detecting anomalies in multi-dimensional time series. These challenges arise from the fact that in an *N*-dimensional time series, anomalies typically manifest in only *K* of the time series, where *K* is less than *N*. We discussed if we had prior knowledge of which *K* time series exhibit anomalies, their locations could be easily determined. However, lacking this prior knowledge, the search space becomes of size 2^N, rendering a greedy search approach unfeasible. In Chapter three, we introduced TSADIS, a novel and straightforward algorithm that resolves this issue by enabling the rapid identification of the best K of N anomaly subset for any K value. Moreover, we presented a simple metric capable of ranking the top anomaly subsets for all K values ranging from 1 to N. While our methods are adaptable to various anomaly scoring models, we illustrated their effectiveness using the Matrix Profile as a concrete example.

We present a comprehensive evaluation of our algorithm's performance across three datasets/approaches. In each instance, we demonstrate our ability to either match or surpass the original author's approach, even though we intentionally made the task more challenging for ourselves. For example, we achieved better results by entirely disregarding the training data.

We conducted a comparison between TSADIS and nine state-of-the-art anomaly detection algorithms, as benchmarked by NeurIPS Benchmark. The results clearly indicate that our proposed solution outperforms all competing methods. We also acknowledged the limitations of TSADIS in detecting anomalies resulting from decorrelation between time series and proposed a potential solution to address this issue. In future work, our aim is to further develop this idea to generalize TSADIS for capturing such types of anomalies.

The possibilities for future research are extensive, and we anticipate that researchers from diverse communities will uncover additional uses and properties of ORR and TSADIS beyond our initial scope. To foster community engagement and the expansion of our ideas, we have made all code and supplemental materials readily accessible [12][63].

Bibliography

[1] Agrahari R, et. al. Assessing Feature Representations for Instance-Based Cross-Domain Anomaly Detection in Cloud Services Univariate Time Series Data. IoT. 2022 Jan 29;3(1):123-44.

[2] Alzantot M, Chakraborty S, Srivastava M. Sensegen: A deep learning architecture for synthetic sensor data generation. In2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops) 2017 Mar 13 (pp. 188-193). IEEE.

[3] Aminifar F et. al. A review of power system protection and asset management with machine learning techniques. Energy Systems. 2022 Nov;13(4):855-92.

[4] Aubet FX, Zügner D, Gasthaus J. Monte Carlo EM for deep time series anomaly detection. arXiv preprint arXiv:2112.14436. 2021 Dec 29.

[5] Audibert J, Marti S, Guyard F, Zuluaga MA. From Univariate to Multivariate Time Series Anomaly Detection with Non-Local Information. InInternational Workshop on Advanced Analytics and Learning on Temporal Data 2021 Sep 13 (pp. 186-194). Springer, Cham.

[6] Audibert J, Michiardi P, Guyard F, Marti S, Zuluaga MA. Do Deep Neural Networks Contribute to Multivariate Time Series Anomaly Detection?. arXiv preprint arXiv:2204.01637. 2022 Apr 4.

[7] Audibert J, Michiardi P, Guyard F, Marti S, Zuluaga MA. Usad: Unsupervised anomaly detection on multivariate time series. InProceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining 2020 Aug 23 (pp. 3395-3404).

[8] Bhatnagar A, Kassianik P, Liu C, Lan T, Yang W, Cassius R, Sahoo D, Arpit D,

Subramanian S, Woo G, Saha A. Merlion: A machine learning library for time series. arXiv preprint arXiv:2109.09265. 2021 Sep 20.

[9] Boniol P, Linardi M, Roncallo F, Palpanas T, Meftah M, Remy E. Unsupervised and scalable subsequence anomaly detection in large data series. The VLDB Journal. 2021 Nov;30(6):909-31.

[10] Bontemps L, Cao VL, McDermott J, Le-Khac NA. Collective anomaly detection based on long short-term memory recurrent neural networks. InFuture Data and Security Engineering: Third International Conference, FDSE 2016, Can Tho City, Vietnam, November 23-25, 2016, Proceedings 3 2016 (pp. 141-152). Springer International Publishing.

[11] Brophy E, Wang Z, She Q, Ward T. Generative adversarial networks in time series: A survey and taxonomy. arXiv preprint arXiv:2107.11098. 2021 Jul 23.

[12] C22MP(2022) Supporting webpage: sites.google.com/view/c22mp/home

[13] Daigavane A, Wagstaff KL, Doran G, Cochrane CJ, Jackman CM, Rymer A. Unsupervised detection of Saturn magnetic field boundary crossings from plasma spectrometer data. Computers & Geosciences. 2022 Apr 1;161:105040. [11]Dau HA et. al. The UCR time series archive. IEEE/CAA Journal of Automatica Sinica. 2019 Nov 8;6(6):1293-305. URL www.cs.ucr.edu/~eamonn/time_series_data_2018.

[14] Dau HA et. al. The UCR time series archive. IEEE/CAA Journal of Automatica Sinica. 2019 Nov 8;6(6):1293-305. URL www.cs.ucr.edu/~eamonn/time_series_data_2018.

[15] Doshi K, Abudalou S, Yilmaz Y. TiSAT: time series anomaly transformer. arXiv preprint arXiv:2203.05167. 2022 Mar 10.

[16] Fährmann D, Damer N, Kirchbuchner F, Kuijper A. Lightweight long short-term memory variational auto-encoder for multivariate time series anomaly detection in industrial control systems. Sensors. 2022 Apr 9;22(8):2886.

[17] Fengming Z, Shufang L, Zhimin G, Bo W, Shiming T, Mingming P. Anomaly detection in smart grid based on encoder-decoder framework with recurrent neural network. The journal of china universities of Posts and Telecommunications. 2017 Dec 1;24(6):67-73.

[18] Geiger A, Liu D, Alnegheimish S, Cuesta-Infante A, Veeramachaneni K. Tadgan: Time series anomaly detection using generative adversarial networks. In2020 IEEE International Conference on Big Data (Big Data) 2020 Dec 10 (pp. 33-43). IEEE.

[19] Goel P, Datta A, Mannan MS. Industrial alarm systems: Challenges and opportunities. Journal of Loss Prevention in the Process Industries. 2017 Nov 1;50:23-36.

[20] Goh J, Adepu S, Junejo KN, Mathur A. A dataset to support research in the design of secure water treatment systems. Intl. conference on critical information infrastructures security 2016 (pp. 88-99). Springer.

[21] Goswami M, Challu C, Callot L, Minorics L, Kan A. Unsupervised Model Selection for Time-series Anomaly Detection. arXiv preprint arXiv:2210.01078. 2022 Oct 3.

[22] Guo Y, Liao W, Wang Q, Yu L, Ji T, Li P. Multidimensional time series anomaly detection: A gru-based gaussian mixture variational autoencoder approach. InAsian Conference on Machine Learning 2018 Nov 4 (pp. 97-112). PMLR.

[23] Guyon I, Elisseeff A. An introduction to variable and feature selection. Journal of machine learning research. 2003;3(Mar):1157-82.

[24] Haimes YY. Modeling and managing interdependent complex systems of systems. John Wiley & Sons; 2018 Oct 2.

[25] Huang H, Baddour N. Bearing vibration data collected under time-varying rotational speed conditions. Data in brief. 2018; 21:1745-9.

[26] Huet A, Navarro JM, Rossi D. Local Evaluation of Time Series Anomaly Detection Algorithms. In Proceedings of the 28th ACM SIGKDD 2022 (pp. 635-645).

[27] Hundman K, Constantinou V, Laporte C, Colwell I, Soderstrom T. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In Proc of 24th ACM SIGKDD 2018 (pp. 387-95).

[28] Hwang WS, Yun JH, Kim J, Min BG. Do you know existing accuracy metrics overrate time-series anomaly detections? In Proceedings of the 37th ACM/SIGAPP SAC 2022 (pp. 403-412).

[29] Ibrahim M, Alsheikh A, Awaysheh FM, Alshehri MD. Machine learning schemes for anomaly detection in solar power plants. Energies. 2022 Feb 1;15(3):1082.

[30] Idé T. Why does subsequence time-series clustering produce sine waves?. InKnowledge Discovery in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings 10 2006 (pp. 211-222). Springer Berlin Heidelberg.

[31] Jackson TD, et. al. The motion of trees in the wind: a data synthesis.

Biogeosciences. 2021 Jul 6;18(13):4059-72.

[32] Keogh E. (2022) Irrational Exuberance Why we should not believe 95% of papers on Time Series Anomaly Detection. SIGKDD 2021 Keynote. www.youtube.com/watch?v=Vg1p3DouX8w&.

[33] Keogh E, Lin J. Clustering of time-series subsequences is meaningless: implications for previous and future research. Knowledge and information systems. 2005 Aug;8:154-77.

[34] Kim S, Choi K, Choi HS, Lee B, Yoon S. Towards a rigorous evaluation of time-series anomaly detection. In Proceedings of the AAAI 2022 Jun 28 (Vol. 36, No. 7, pp. 7194-7201).

[35] Lai KH, Zha D, Xu J, Zhao Y, Wang G, Hu X. Revisiting time series outlier detection: Definitions and benchmarks. In 35th Conference on NeurIPS Datasets and Benchmarks Track. 2021.

[36] Li D, Chen D, Jin B, Shi L, Goh J, Ng SK. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. InArtificial Neural Networks and Machine Learning–ICANN 2019: Text and Time Series: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings, Part IV 2019 Sep 9 (pp. 703-716). Cham: Springer International Publishing.

[37] Liu HY, Gao ZZ, Wang ZH, Deng YH. Time Series Classification with Shapelet and Canonical Features. Applied Sciences. 2022 Aug 30;12(17):8685.

[38] Loh WY. Classification and regression trees. Wiley interdisciplinary reviews: data mining and knowledge discovery. 2011 Jan;1(1):14-23.

[39] Lu Y, Wu R, Mueen A, Zuluaga MA, Keogh E. Matrix profile XXIV: scaling time series anomaly detection to trillions of datapoints and ultra-fast arriving data streams. InProceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2022 Aug 14 (pp. 1173-1182).

[40] Lubba CH, Sethi SS, Knaute P, Schultz SR, Fulcher BD, Jones NS (2019) catch22: CAnonical Time-series CHaracteristics. Data Min Knowl Disc 33(6):1821-1852.

[41] Lauer J, Zhou M, Ye S, Menegas W, Nath T, Rahman MM, Di Santo V, Soberanes D, Feng G, Murthy VN, Lauder G. Multi-animal pose estimation and tracking with DeepLabCut. BioRxiv. 2021 Jan 1.

[42] MacQueen J. Classification and analysis of multivariate observations. In5th Berkeley Symp. Math. Statist. Probability 1967 Jun 21 (pp. 281-297). Los Angeles LA USA: University of California.

[43] Marimon X, Traserra S, Jiménez M, Ospina A, Benítez R. Detection of abnormal cardiac response patterns in cardiac tissue using deep learning. Mathematics. 2022 Aug 5;10(15):2786.

[44] Munir M, Siddiqui SA, Dengel A, Ahmed S. DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. Ieee Access. 2018 Dec 19;7:1991-2005.

[45] Nakamura T, Imamura M, Mercer R, Keogh E. Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In2020 IEEE ICDM 2020 Nov 17 (pp. 1190-1195).

[46] Park D, Hoshi Y, Kemp CC. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. IEEE Robotics and

Automation Letters. 2018 Feb 2;3(3):1544-51.

[47] Ren H, Xu B, Wang Y, Yi C, Huang C, Kou X, Xing T, Yang M, Tong J, Zhang Q. Time-series anomaly detection service at microsoft. InProceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining 2019 Jul 25 (pp. 3009-3017).

[48] Rewicki F, Denzler J, Niebling J. Is it worth it? An experimental comparison of six deep-and classical machine learning methods for unsupervised anomaly detection in time series. arXiv preprint arXiv:2212.11080. 2022 Dec 21.

[49] Piatetsky-Shapiro G. Data types/sources analyzed.

[50] Saarela M, Jauhiainen S. Comparison of feature importance measures as explanations for classification models. SN Applied Sciences. 2021 Feb;3:1-2.

[51] Thompson, D. W., 1917. On Growth and Form. Cambridge University Press.

[52] Tuli S, Casale G, Jennings NR. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. arXiv preprint arXiv:2201.07284. 2022 Jan 18.

[53] Turowski M, et. al. Modeling and generating synthetic anomalies for energy and power time series. In Proceedings of the 13th ACM e-Energy 2022 (pp. 471-484).

[54] Keogh E. Irrational Exuberance Why we should not believe 95% of papers on Time Series Anomaly Detection. SIGKDD workshop talk (2022). https://www.youtube.com/watch?v=Vg1p3DouX8w

[55] Khansa HE, Gervet C, Brouillet A. Prominent discord discovery with matrix profile: application to climate data insight. In10th international conference of advanced computer science & information technology (ACSIT 2022) May 2022 (pp. 21-22).

[56] Kim S, Choi K, Choi HS, Lee B, Yoon S. Towards a rigorous evaluation of time-series anomaly detection. InProceedings of the AAAI Conference on Artificial Intelligence 2022 Jun 28 (Vol. 36, No. 7, pp. 7194-7201).

[57] Kravchik M, Shabtai A. Efficient cyber attack detection in industrial control systems using lightweight neural networks and pca. IEEE Transactions on Dependable and Secure Computing. 2021 Jan 8;19(4):2179-97.

[58] Mueen A, Viswanathan K, Gupta CK, Keogh E. The fastest similarity search algorithm for time series subsequences under Euclidean distance. url: www cs unm edu/~ mueen. FastestSimilaritySearch html (Accessed 24 May 2016). 2015.

[59] Nakamura T, Imamura M, Mercer R, Keogh E. Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In2020 IEEE international conference on data mining (ICDM) 2020 Nov 17 (pp. 1190-1195). IEEE.

[60] Park JY, Wilson E, Parker A, Nagy Z. The good, the bad, and the ugly: Data-driven load profile discord identification in a large building portfolio. Energy and Buildings. 2020 May 15;215:109892.

[61] Sanchez-Pi N, Leme LA, Garcia AC. Intelligent agents for alarm management in petroleum ambient. Journal of Intelligent & Fuzzy Systems. 2015 Jan 1;28(1):43-53.

[62] Thill M, Konen W, Bäck T. Time series encodings with temporal convolutional networks. InInternational Conference on Bioinspired Methods and Their Applications 2020 Nov 16 (pp. 161-173). Cham: Springer International Publishing.

[63] TSADIS webpage: https://sites.google.com/view/tsadis

[64] Wang R, Liu C, Mou X, Guo X, Gao K, Liu P, Wo T, Liu X. Deep Contrastive

One-Class Time Series Anomaly Detection. arXiv preprint arXiv:2207.01472. 2022 Jul 4.

[65] Wen Q, Sun L, Yang F, Song X, Gao J, Wang X, Xu H. Time series data augmentation for deep learning: A survey. arXiv preprint arXiv:2002.12478. 2020 Feb 27.

[66] Wu R, Keogh E. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. IEEE TKDE. 2021.

[67] Yairi T, Kato Y, Hori K. Fault detection by mining association rules from house-keeping data. Inproceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space 2001 Jun 18 (Vol. 18, p. 21). Citeseer.

[68] [50]Yankov D, Keogh E, Rebbapragada U. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowledge and Information Systems. 2008 Nov;17:241-62.

[69] Yeh CC, Kavantzas N, Keogh E. Matrix profile VI: Meaningful multidimensional motif discovery. In2017 IEEE international conference on data mining (ICDM) 2017 Nov 18 (pp. 565-574). IEEE.

[70] Yeh CC, Zhu Y, Ulanova L, Begum N, Ding Y, Dau HA, Silva DF, Mueen A, Keogh E. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In2016 IEEE 16th international conference on data mining (ICDM) 2016 Dec 12 (pp. 1317-1322). Ieee.

[71] Yoon J, Jarrett D, Van der Schaar M. Time-series generative adversarial networks. Advances in neural information processing systems. 2019;32.

[72] Zhang C, Kuppannagari SR, Kannan R, Prasanna VK. Generative adversarial

network for synthetic time series data generation in smart grids. In2018 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm) 2018 Oct 29 (pp. 1-6). IEEE.

[73] Zhang C, Song D, Chen Y, Feng X, Lumezanu C, Cheng W, Ni J, Zong B, Chen H, Chawla NV. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. InProceedings of the AAAI conference on artificial intelligence 2019 Jul 17 (Vol. 33, No. 01, pp. 1409-1416).

[74] Zhu Y, Yeh CC, Zimmerman Z, Kamgar K, Keogh E. Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. In 2018 IEEE ICDM 2018 (pp. 837-846).