C^{22}MP: Fusing catch22 and the Matrix Profile to Produce an Efficient and Interpretable Anomaly Detector

Sadaf Tafazoli, Yue Lu, Renjie Wu, Thirumalai Vinjamoor Akhil Srinivas, Hannah Dela Cruz, Ryan Mercer, Eamonn Keogh
University of California, Riverside

{stafa002, ylu175, rwu034, hdela004, tsrin002, rmerc002} @ucr.edu, eamonn@cs.ucr.edu

***Abstract*—**The Matrix Profile is a data structure that annotates a time series by recording each subsequence's Euclidean distance to its nearest neighbor. In recent years the community has shown that using the Matrix Profile it is possible to discover many useful properties of a time series, including repeated behaviors, anomalies, evolving patterns, regimes, etc. 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 algorithms to solve both problems and demonstrate that for most domains, the proposed C^{22}MP is a state-of-the-art anomaly detector.

Keywords—Anomaly Detection; Feature Extraction

I. Introduction

In recent years the Matrix Profile (MP) has emerged as one of the most promising general tools for time series data mining [30]. 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 *shapes*, but in the subsequence's *features*. In such domains the MP is of little use.

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 [13][17] 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 [13].

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 [8].

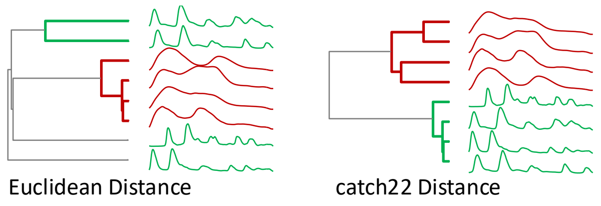

Fig. 1. Eight instances from the 50words dataset clustered using Euclidean distance (*left*) and catch22 (*right*). At least for these two classes, catch22 is better able to represent the conserved class-specific structure.

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 [30], 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.

In this work we introduce a novel representation and algorithm, C^{22}MP and ORR, to solve both problems. 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 [30]), 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.

The rest of the paper is organized as follows: in Section II we motivate the need for a feature-based anomaly detector before discussing related work in Section III. Section IV introduces the necessary definitions and notations. In Section V 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 VI, we conduct an extensive empirical evaluation before offering conclusions in Section VII.

II. Motivation

There are hundreds of proposed algorithms for Time Series Anomaly Detection (TSAD). It is natural to ask why we need another one. The vast majority of TSAD algorithms in the literature are only considering *shapes* of the subsequences. This is true of the Matrix Profile and related approaches [28][30], and seems to be true of deep learning approaches [13]. Reasoning about changes in shape clearly works very well for some data types: gait, ECG, daily traffic patterns, etc. But many datasets may have richer and more complex behaviors, such as the insect behavior shown in Fig. 2.

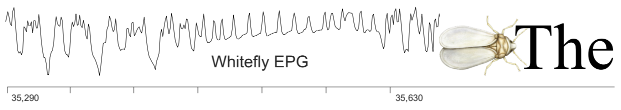

Fig. 2. *left*) A snippet of the data used in the experiments in Fig. 3. *right*) A whitefly (*Bemisia tabaci*) shown to scale with the font used in this paper.

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. 3 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.

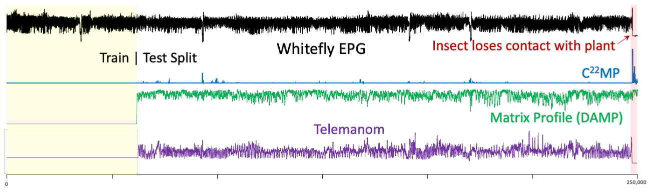

Fig. 3. *top*) Forty minutes of normal insect behavior that ends with a visually obvious anomaly. *bottom*) Three TSAD algorithms, including the one proposed in this work, evaluated on this dataset. Only C^{22}MP correctly peaks at the right place (extreme *right*).

Both the Matrix Profile [28][30] and Telemanom (a SOTA deep learning approach) [13] fail to discover this anomaly, but our proposed feature-based method, C^{22}MP, strongly peaks the moment it encounters the anomaly. As we will show in Section VI, there are many other domains/situations for which a feature-based approach is more suitable. Note that we are not suggesting feature-based methods could completely replace shape-based approaches, rather they can complement them.

As an aside, it is worth previewing 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 working with tiny datasets[1][27].

III. Background and Related Work

Before discussing related work, we wish to clarify exactly *what* problem we are attempting to solve. A complete anomaly detection system can be said to be comprised of two parts:

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 some TSAD systems these two tasks are irrevocably intertwined. However, 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`, `alert`$_{\text{manager}}$, `alert`$_{\text{engineer}}$,…, `red-alert`). Finally, echoing the warning of Wu and Keogh [27] and other recent papers, if we are not careful evaluating the results of task '2' only can give a very misleading impression of how well we are doing.

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.

[1] The two most cited datasets for evaluating TSAD algorithms are tiny: NY-Taxi (length 10,320) and Yahoo! Webscope (mean length 1,415) [27].

The field of TSAD is very active[13][27][28][30] and increasingly diverse to review, see [27] and the references therein. Many methods define anomalies as the subsequences that are maximally far from their nearest neighbor(s), or the time series *discord* [28][30]. In contrast to such shape-based methods, deep learning approaches typically "*model complex nonlinear feature interactions*" [1][13]. However, the features in general are completely opaque to the user, and overfitting remains a significant problem.

Our work exploits the success of the catch22 feature set for time series [1][17]. 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.

IV. Definitions and Notation

Below we introduce all the necessary notations and definitions. 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. 4 demonstrates this notation on a running example dataset. This dataset has a local distortion (in this case, *noise*) which *may* be an anomaly, depending on the context.

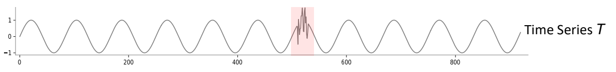

Fig. 4. Our running example has a local burst of noise, which may be an anomaly, depending on the context.

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. $T_{i,m} = [t_i, t_{i+1}, \dots, t_{i+m-1}]$, $1 \le i \le n - m + 1$.

The length of the subsequence is typically set by the user based on domain knowledge.

In our proposed algorithm we need to calculate *all-pairs-similarity* of *all* subsequences of a given time series. 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: $A = \{T_{1,m}, T_{2,m}, \dots, T_{n-m+1,m}\}$, 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 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. 5 illustrates this 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.

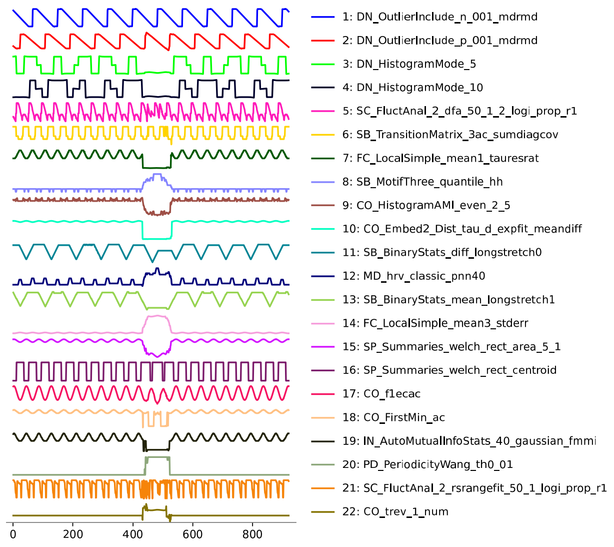

Fig. 5. C22 Feature profiles for the running example introduced in Fig. 4.

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 $d_{i,j}$ $(1 \le i, j \le n - m + 1)$ 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 [28][30]. 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 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 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*.

By leveraging the close relationship between 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 [30]. 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 [3][20][28][30]. 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,

$C^{22}MP = [min(D_1), min(D_2), \ldots, min(D_{n-m+1})]$, 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 $D_i^L = d_{i,1}, d_{i,2}, \ldots, d_{i,i-m+1}$ 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 $C^{22}MP^L = [min(D_1^L), min(D_2^L), \ldots, min(D_{n-m+1}^L)]$.

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 V.C, we will show 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.

V. The C^{22}MP

Having introduced the necessary notations, 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.

A. LeftC^{22}MP

In TABLE I., we outline a brute force algorithm to compute the *left*C^{22}MP, assuming the feature weight vector w has already been determined. 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 II. 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 lines 6 to 10, we iterate through all the subsequences that are before the candidate subsequence to check which one of them has the least distance to the candidate subsequence.

TABLE I. 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 [30]. 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. 6 demonstrates the output of this algorithm on the running example introduced in Fig. 4.

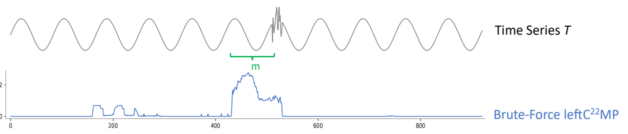

Fig. 6. Top-1 *left*C^{22}MP computed on our running example.

To calculate the feature profiles, we use the subroutine in TABLE II. 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. Finally, in line 7 we scale the feature profiles to make them unitless and commensurate.

TABLE II. 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
```

B. *Fast Computation of LeftC^{22}MP*

The brute force algorithm shown in TABLE I. 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 the more recent data will be more likely to be similar to current data.

In TABLE III. we formalize these observations with the ORR (Observed Repudiation of Regularity) algorithm. 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*.

TABLE III. EARLY-ABANDONING COMPUTATION OF *LEFT*C^{22}MP

```
FFunction: 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 leftC22MP
1   FP = get_feature_profie(T, m, w)//get feature profiles (def. 4)
2   a_left_c22mp = zeros((len(T)-m+1,))
3   bsf = 0   // The current best discordia score
4   //Scan all subsequences in test data and calculate approximate left_c22mp
5   for i in range(s_idx, len(T) - m + 1):
6       lbsf = inf
7       for j in range(i-1-exclude_zone, -1, -1):
8           d = ditance(FP[i], FP[j])
9           if d < lbsf:
10              lbsf = d
11          if d < bsf and j > 0:
12             break
13          elif d >= bsf and j > 0:
14             continue
15          elif d >= bsf and j == 0:
16             bsf = lbsf
17      a_left_c22mp[i] = lbsf
18  return a_left_c22mp
```

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. 7 offers visual evidence of the correctness of ORR.

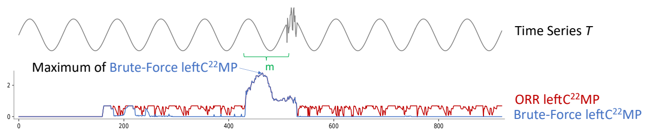

Fig. 7. A comparison of the output of the top-1 brute-force and the ORR algorithm on a small toy dataset.

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 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 VI.H, the early-abandoning algorithm is several orders of magnitude faster than brute-force search and allows us to address datasets with billions of datapoints.

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

The ORR algorithm introduced in TABLE III. 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. 8.*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. 8.*left*. Note that this clustering is still better than using Euclidean distance, but it is evident that a judicious choice of features can greatly benefit catch22.

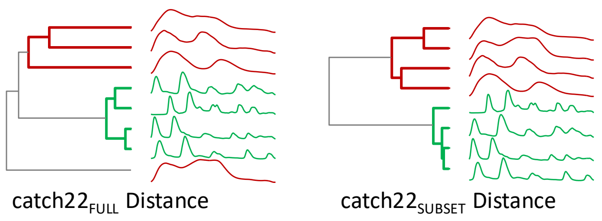

Fig. 8. Instances from the 50words dataset clustered using the full set of catch22 features (*left*) and a subset of catch22 features (*right*). The full set produces a reasonable clustering, but incorrectly considers a red object to be an outlier (cf.Fig. 1).

While this suggests that a good choice of feature 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.

1) Hand Coded Subsets of Features

In some cases, it may be possible for practitioners to use their domain knowledge to handcraft an appropriate 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 [7] [2].

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. For brevity, we relegate the pseudocode for this search to the website [7].

3) Feature Search by Feature Profiles

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. 4, 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. Again, for brevity, we relegate the pseudocode for this simple idea to the website [7].

4) Human in the Loop Feature Search

In Section 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. 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) and 3).

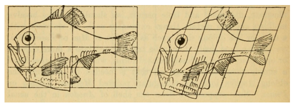

Fig. 9. A figure from [24] shows that we can take a known fish, here

Here we are inspired by an observation by Sir D'Arcy Thompson. Fig. 9 shows an image from his 1917 book *On Growth and Form* [24],

[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.

the *Olfer*'s Hatchetfish (*left*), and apply a simple geometric transformation to obtain another fish that really exists, here the *Diaphanous* Hatchetfish (*right*).

which shows it is possible to reproduce almost all the morphological diversity of fishes, with just a few samples and a set of geometric transformations.

We believe that we can similarly create plausible synthetic anomalies [15][25]. Consider the datasets shown in Fig. 10.

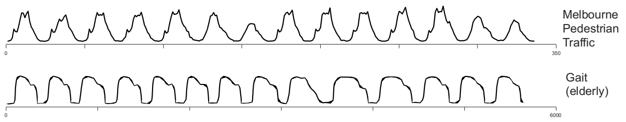

Fig. 10. Two datasets with about 14 cycles shown.

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 [7]). 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. 11 where we added the *same* amount of warping to the middle section of each time series.

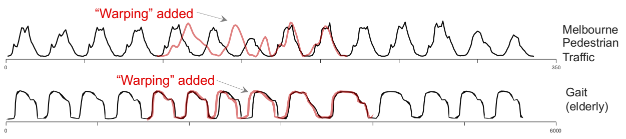

Fig. 11. The two datasets shown in Fig. 10 with the same amount of warping added to each's middle section.

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. [6][7][26][29]), 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. However, we found that for each distortion we could make high-quality examples with just a few lines of code (see [7]). In [7], we show a video of a typical interaction with a user.

VI. Empirical Evaluation

To make certain that our experiments are reproducible, we have built a website [7] 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.

A. 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 [27]. 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. 12, which shows an excerpt of one dimension of the 51-dimensional SWAT benchmark [10]. This is one of the most cited and used benchmarks in the literature, appearing in at least 200 papers. We ran the fixed weigh *left*C^{22}MP algorithm on this dataset. This example highlights the problems the community has noted:

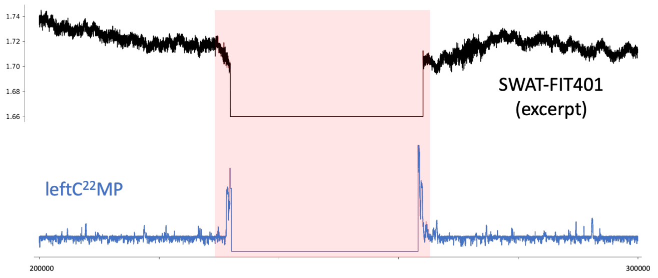

Fig. 12. An excerpt from the SWAT-FIT401 dataset. The ground truth anomaly is highlighted in red. Unsurprisingly *left*C^{22}MP finds this anomaly.

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 [27]: `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".
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.

In summary, we have sympathy for the claims that both the common benchmarks/evaluation metrics [27] are flawed.

Given these issues, how should we evaluate? We agree with [20] 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. 12 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.

B. A Visual Intuition for *left*C^{22}MP

With the caveats in the last section in mind, in Fig. 13, 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 [7] 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.

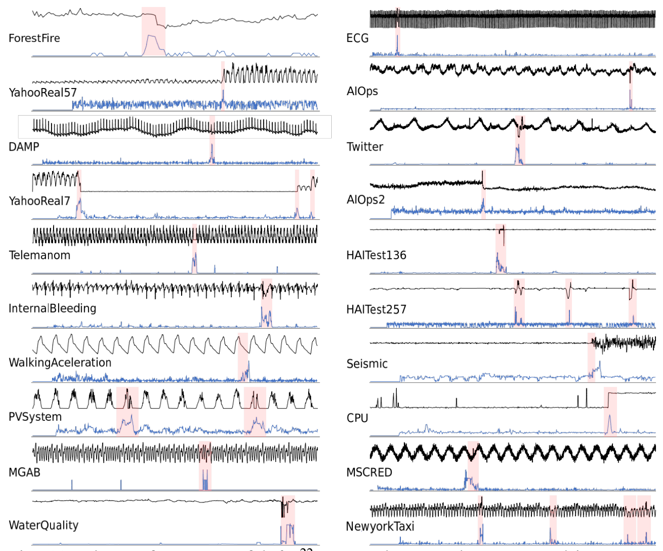

Fig. 13. The performance of *left*C^{22}MP on datasets that appeared in twenty different papers in the last two decades.

C. 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. The **HEX**agon ML/**UCR** dataset is different in that the 250 datasets come with a concrete and well-reasoned metric of success. In addition, the HEX/UCR datasets are unusual in having detailed provenance, largely freeing them from the flaws noted in [27]. 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 and biology. TABLE IV. shows a comparison of our proposed method with rival approaches.

TABLE IV. A COMPARISON OF 14 ALGORITHMS ON THE HEX/UCR DATASET

Method	Score	Method	Score
leftC^{22}MP	0.568	{{{ DAMP + *left*C^{22}MP }}}	{0.692}
DAMP (*left*MP) [28]	0.556	USAD [2]	0.276
AE [2]	0.236	Telemanom [13]	0.468[3]
LSTM-VAE [21]	0.198	SCRIMP (*full* MP) [30]	0.416
RRCF [23]	0.030	MERLIN (generalized MP) [23]	0.440
MDI [23]	0.470	NORMA [4]	0.474
TranAD [23]	0.190	GANF[23]	0.240

The results highlighted in blue are results that we copied from other works. All other results we computed ourselves.

Our proposed algorithm outperforms all rival approaches. It is slightly better than DAMP, which is regarded as SOTA [28]. 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.

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.

D. 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 the dataset shown in Fig. 14.A can help explain.

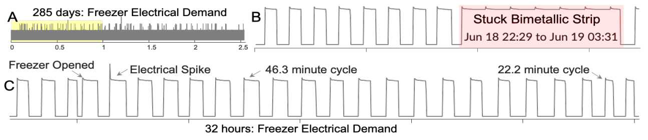

Fig. 14. A) An electrical demand dataset for a single device, a freezer. B) The (rather obvious) anomaly caused by a stuck temperature sensor. C) A random section of the training data.

In Fig. 14.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 data, a snippet illustrated in Fig. 14.C, provides insight why this is the case.

- The period of the data (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. 10) but cannot handle *variable* periodicity.
- The dataset is replete with inconsequential variations. For example, short off-cycles caused by the freezer being opened, or electrical spikes. These 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 deep learning approaches.

[3] Telemanom runs out of memory on the larger datasets. This score is extrapolated from the shorter/easier datasets, thus optimistic [28].

In Fig. 15, we compare three approaches on this problem. The *left*C22MP easily finds this anomaly but Telemanom 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*C22MP.

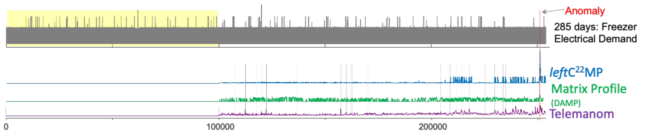

Fig. 15. The *left*C22MP can easily find the Stuck-Bimetallic-Strip anomaly. DAMP *just* finds it, and Telemanom fails.

Here DAMP, using the subsequence length selection tool suggested in the original paper [28], did *just* find the anomaly, but it would have failed at a slightly different length. In contrast, *left*C22MP is successful even if we double or halve this suggested setting for the subsequence length.

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 [7] for brevity) support our observations and suggest research directions for advocates of deep learning TSAD algorithms.

E. 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][16]. 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) data. A recent paper [19] introduced a sophisticated data generator that takes real data and introduces into it highly plausible anomalies of amplitude, duration, morphology or timing. Interestingly, some anomalies are caused not by the *addition* of a new pattern or disturbance, but by the *omission* of an expected pattern (`Pause / Missing Pulse`). In TABLE V. we show the results of comparing our approach with two bespoke deep learning algorithms. To be fair to deep learning algorithms (which have sensitive parameters and can be difficult to reproduce), we copied the results from [19] and used identical settings for our experiments (also averaging over 100 runs).

TABLE V. A COMPARISON OF FOUR TSAD ALGORITHMS ON CCT DATA

Anomaly Type	Autoencoder	LSTM	*left*C22MP	*left*C23MP
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**

In order to "stress test" our approach, we did not do any feature selection here, we simply used all 22, and all 23 features. 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.

F. Case Study on Mouse Motion Capture

Researchers at <blinded> are building a mouse model to understand factors that influence the onset of Parkinson's disease. As Fig. 16 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 [18]) is unable to find the mouse's paw as it is occluded by its tail (type **B** in Fig. 16). Such anomalies can then be excluded from analysis. More interestingly, however, there may be *behavioral* anomalies induced by gene knockouts or medications. To test our ability to find both types 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.

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. 16.*top*. While most anomalies were simple feature extraction issues, two anomalies surprised us. The first anomaly, type **D** (Fig. 16.*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 watch a video of this anomaly[7].

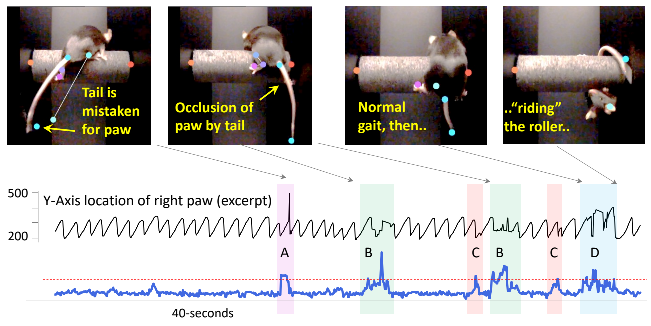

Fig. 16. *bottom*) A trace from the right paw of a healthy mouse, with its companion C22MP. *top*) Screenshots of the video with markers labeled and tracked by the DeepLabCut motion capture tool [18].

G. 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 V.C.4) Consider the dataset UCR-Hex-50 (Mars) [27]. This dataset is of length 8,184, with the first 4,000 datapoints acting as a (positive only) training set. As shown in Fig. 17, if we run top-1 anomaly detection on this with all twenty-two features, the best anomaly is a false positive.

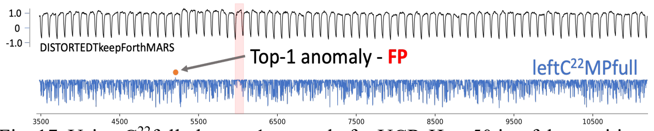

Fig. 17. Using C^{22}full, the top-1 anomaly for UCR-Hex-50 is a false positive. The algorithm seems to have been confused by wandering baseline.

We extracted the training data and used the ideas in Sections V.C.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 video of the process is archived at [7]. We then used the algorithm outlined in Section V.C.3) to learn a feature set {22, 10, 14, 12} for this problem. As shown in Fig. 18, 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 UCR-Hex-6 (AirTemperature) 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.

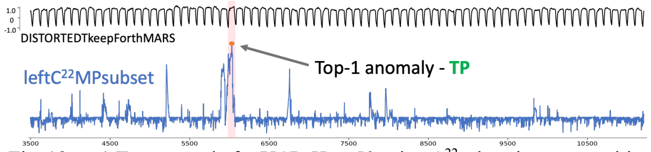

Fig. 18. *top*) Top anomaly for UCR-Hex-50 using C^{22}subset is a true positive.

H. 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. As the Fig. 19 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.

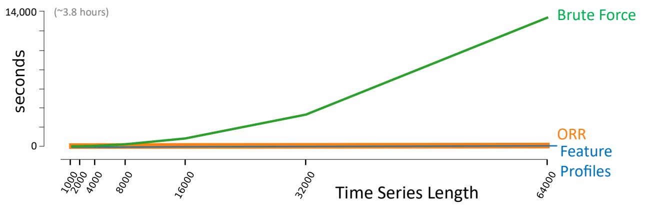

Fig. 19. A comparison of ORR and the brute-force approach to compute the *left*C^{22}MP. The blue curve shows the time to do the C22 feature extraction.

VII. Conclusions and Future Work

We have introduced a new representation C^{22}MP and shown 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 will generalize the C^{22}MP to other MP primitives, including motifs, chains, and snippets.

References

[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] Audibert J, Marti S, Guyard F, Zuluaga MA. From Univariate to Multivariate Time Series Anomaly Detection with Non-Local Information. In International AALTD 2021 (pp. 186-194).
[3] 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.
[4] 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.
[5] 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.
[6] Brophy E, Wang Z, She Q, Ward T. Generative adversarial networks in time series: A survey and taxonomy. arXiv:2107.11098. 2021 Jul 23.
[7] C^{22}MP(2023) Supporting webpage: sites.google.com/view/c22mp/home
[8] Dau HA et. al. The UCR time series archive. IEEE/CAA Journal of Automatica Sinica. 2019 Nov 8;6(6):1293-305.
[9] Geiger A, Liu D, Alnegheimish S, Cuesta-Infante A, Veeramachaneni K. Tadgan: Time series anomaly detection using generative adversarial networks. In 2020 IEEE Big Data, 2020 Dec 10 (pp. 33-43).
[10] 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.
[11] Guyon I, Elisseeff A. An introduction to variable and feature selection. Journal of machine learning research. 2003;3:1157-82.
[12] Huang H, Baddour N. Bearing vibration data collected under time-varying rotational speed conditions. Data in brief. 2018; 21:1745-9.
[13] 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).
[14] Jackson TD, et. al. The motion of trees in the wind: a data synthesis. Biogeosciences. 2021 Jul 6;18(13):4059-72.
[15] Lai KH, Zha D, Xu J, Zhao Y, Wang G, Hu X. Revisiting time series outlier detection: Definitions and benchmarks. In NeurIPS 2021.
[16] Liu H, Gao Z, Wang Z, Deng Y. Time Series Classification with Shapelet and Canonical Features. Applied Sciences. 2022 30;12(17):8685.
[17] 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.
[18] Lauer J, et al. Multi-animal pose estimation and tracking with DeepLabCut. BioRxiv. 2021.
[19] Marimon X, et al.. Detection of abnormal cardiac response patterns in cardiac tissue using deep learning. Mathematics. 2022 5;10(15):2786.
[20] Nakamura T, Imamura M, Mercer R, Keogh E. Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In 2020 IEEE ICDM 2020, pp. 1190-1195.
[21] 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.
[22] Ren H, et al. Time-series anomaly detection service at microsoft. In ACM SIGKDD 2019.
[23] 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:2212.11080. 2022.

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

[25] Turowski M, et. al. Modeling and generating synthetic anomalies for energy and power time series. In ACM e-Energy 2022.

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

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

[28] Yue Lu, Renjie Wu, Mueen, Maria Zuluaga, Eamonn. Keogh: Matrix Profile XXIV: Scaling Time Series Anomaly Detection to Trillions of Datapoints and Ultra-fast Arriving Data Streams. KDD 2022: 1173-82.

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

[30] Zhu Y, et al. Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. In 2018 IEEE ICDM (pp. 837-846).