UNIVERSITY OF CALIFORNIA
RIVERSIDE

Ultra-fast, Accurate, and Practical Anomaly Detection Algorithms for Time Series Data

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

Doctor of Philosophy

in

Computer Science

by

Yue Lu

December 2023

Dissertation Committee:
- Dr. Eamonn Keogh, Chairperson
- Dr. Tao Jiang
- Dr. Tamar Shinar
- Dr. Evangelos Papalexakis

The Dissertation of Yue Lu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgement

First and foremost, I would like to express my deep and sincere gratitude to Dr. Eamonn Keogh for his guidance and mentorship over the past four years. Working with him on research has been an enriching experience, allowing me to learn a great deal. I am particularly grateful for the numerous opportunities and assistance he has provided me throughout my academic journey. I feel incredibly fortunate and proud to be his student.

I also wish to extend my heartfelt thanks to all my lab mates: Ryan Mercer, Audrey Der, Renjie Wu, Seyedehmaryam Shahcheraghi, Sadaf Tafazoli, Thirumalai Vinjamoor Akhil Srinivas, and Dr. Abdullah Mueen. As an international student, their camaraderie and assistance have been a source of comfort and a sense of belonging in a foreign land. Their support has been an invaluable part of my journey.

Lastly, but certainly not least, I am grateful to all the co-authors of my papers for their invaluable contributions and assistance. Without their help and collective efforts, my current achievements would not have been possible. I deeply appreciate their willingness to invest their time and expertise in assisting me with my research work.

The text of this dissertation, in part (Chapter 2), is a reprint of the material as it appears in "DAMP: accurate time series anomaly detection on trillions of datapoints and ultra-fast arriving data streams" from Data Mining and Knowledge Discovery, pp.1-43, 2023. The co-author Dr. Eamonn Keogh, listed in that publication, directed and supervised the research which forms the basis for this dissertation.

I am profoundly appreciative of all the help and support I have received during my research journey, and I owe my deepest thanks to everyone who has been part of it.

ABSTRACT OF THE DISSERTATION

Ultra-fast, Accurate, and Practical Anomaly Detection Algorithms for Time Series Data

by

Yue Lu

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

Time Series Anomaly Detection (TSAD) has emerged as an intensely active field in data mining due to its profound implications in numerous real-world scenarios. While various solutions are proposed each year, empirical evidence indicates that *time series discord*, a simple distance-based technique, remains among the state-of-art methods. However, conventional algorithms to compute time series discords face limitations: they are restricted to batch cases and struggle with scalability beyond tens of thousands of datapoints. To address these challenges, we introduce DAMP, a novel algorithm that computes exact left-discords on fast-arriving streams, at up to 300,000 Hz using a commodity desktop, making it possible to find time series discords in datasets with trillions of datapoints for the first time.

However, time series discords have one other notable issue; the anomalies discovered depend on the algorithm's only input parameter, the subsequence length. To circumvent

this limitation, we introduce MADRID, a Hyper-Anytime Algorithm engineered to efficiently solve the all-discords problem. By using a novel computation ordering strategy, MADRID can reduce the absolute time to compute all-discords, and allow users to interact with their data in real-time. We demonstrate the utility of MADRID in various domains and show that it allows us to find anomalies that would otherwise escape our attention.

In our ongoing endeavor to make TSAD tasks practical and user-centric, we realize that we cannot overlook the user's perspective in defining an anomaly. We posit that without accurately capturing the user's knowledge and requirements, TSAD algorithms are likely to suffer from an influx of false positives, thus hindering their adoption. Hence, we present FIRE, a versatile framework that encapsulates the user's requirements and communicates them to the algorithms. FIRE's flexibility allows for implementation across different domains and algorithms. As we will show, it can make anomaly detection faster, more accurate, and more useful.

In summary, our proposed algorithms, DAMP, MADRID, and FIRE, demonstrate robust performance across diverse domains and provide a robust, versatile, and efficient solution to time series anomaly detection.

Table of Contents

List of Figures

List of Tables

1. Introduction

Time Series Anomaly Detection (TSAD) is one of the most important and widely used tools investigated by the data mining community [2][14][21]. There has been an explosion of interest in Anomaly Detection Algorithms (ADA) in both the commercial and academic settings in recent years. Although various ADAs are proposed each year, the majority of existing ADAs are primarily confined to solving simple problems. If noise, trends, concept drifts, and other factors exist within the time series data, these algorithms can easily be disturbed, consequently leading to false positives. Moreover, due to the complexity of their design, most ADAs require significant time and/or memory for computing, making it difficult for them to be extended to datasets exceeding one hundred thousand data points. We believe that some simple and straightforward approaches could enhance the speed, scalability, and accuracy of TSAD tasks. In this regard, in Chapters 2 to 4 of this thesis, we propose several TSAD methods aimed at ameliorating the deficiencies in speed, scalability, and accuracy identified in existing methodologies. The motivations underlying each of these methods are detailed in the subsequent three sections. Finally, Chapter 5 concludes the thesis and outlines potential directions for future work.

1.1 Scaling Time Series Anomaly Detection to Trillions of Datapoints and Ultra-Fast Arriving Data Streams

TSAD methods can be applied offline to investigate archival data, or online, to monitor critical situations where real-time human intervention is possible. For example, by summoning a doctor or shutting down a machine that may be about to damage itself.

Given its importance, it is unsurprising that this area attracts a lot of attention from the community, with dozens of algorithms proposed each year. However, in spite of the plethora of algorithms in the literature, there is increasing evidence that a twenty-year-old distance-based method called *time series discords* is still competitive [21]. Discords are competitive with deep learning methods in spite (or perhaps *because*) of their great simplicity. Time series discords (sometimes referred to in the literature by the name of the algorithm that is used to discover them, i.e., HOT SAX [49], Matrix Profile [65], SCRIMP [67], etc.), are the subsequences that have the greatest distance from their nearest neighbor in the training data. In most contexts, "training data" refers to all data encountered thus far. This makes discords particularly robust to concept drift, as newly ingested data instantly becomes part of the training data.

At least one hundred papers have reported using discords to solve problems in diverse domains, and discords seem to be the only time series anomaly detection technique to produce "superhuman" results (see discussion in Section 2.1). However, discords have three important limitations that have limited their broader adoption:

- If an anomalous pattern appears at least twice in the time series, then each occurrence will be the other nearest neighbor, and thus fail to optimize the discord definition. This is informally called the *twin-freak* problem.
- Discords are only defined for the *batch* case, but anomaly detection is most actionable in *online* settings.

- In spite of extensive progress in speeding up discord discovery, datasets with millions of datapoints remain intractable.

To address all the above issues, we will introduce the DAMP algorithm in Chapter 2 to rapidly and accurately identify anomalies in data streams exceeding the level of millions.

1.2 Discovering Time Series Anomalies of All Lengths

Although as outlined in Chapter 2, by only considering nearest neighbors to the *left* of the subsequence being examined, DAMP makes time series discords invariant to the so-called *twin-freak* problem and allows them to achieve state-of-the-art (SOTA) performance on several benchmark datasets including the Hexagon ML/UCR Anomaly Detection datasets [48][65]. However, time series discords have noted one weakness, the anomalies discovered depend on the algorithm's only input parameter, the subsequence length. Numerous researchers have pointed this out, for example [42] notes: "*setting the right window size is crucial…failure to provide an optimal window size may incur monetary damage*."

To illustrate the issue, consider Fig. 1 in which we searched for time series discords in an eight-month long time series of pedestrian traffic on Bourke Street in Melbourne Australia. Rather than rely on a single subsequence length *m*, we simply placed the DAMP algorithm [65] in a loop and tested all discord lengths from eight hours to twenty-four hours.

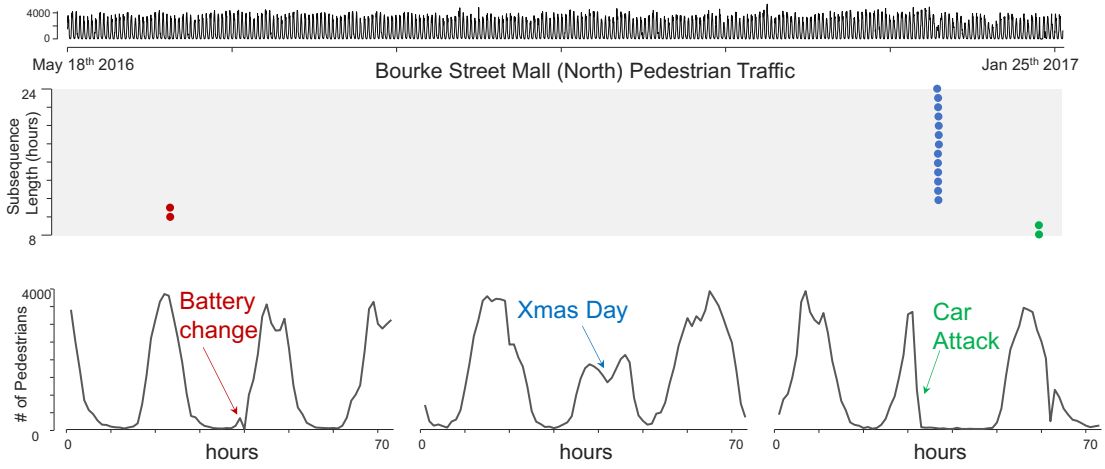

Fig. 1 *top*) An eight-month long time series representing pedestrian traffic in a street in Melbourne1. *center*) The location of the top-1 discord, for subsequence lengths (m) from eight to twenty-four hours. *bottom*) A zoom-in of the three discovered anomalies.

The results clearly depend on the subsequence length chosen. Had we chosen only eight hours, we would have only discovered the tragic car attack January 20th, had we chosen ten hours, we would have discovered the sensor battery change on June 12th, and had we chosen twelve hours (or longer), we would have only discovered Xmas day.

This example exposes an additional issue that is not well understood. We might have imagined that if we somehow had access to out-of-band data and *knew* that our anomalies should last only a certain time, we could set our subsequence length to that value. However, that is not the case; the manner in which an anomaly manifests itself within a time series is not only a function of the anomaly's intrinsic length, but also depends on the rest of the data. For example, the battery change anomaly is two hours long, yet it reveals itself best at a scale of ten hours; the small blip is only anomalous when seen in

[1] This figure, and several others in this work suffer from the small scale of reproduction allowed by this format. We encourage the interested reader to visit [51] for large scale images and additional context.

the context of the normal data that surrounds it. The obvious way to bypass this issue is to find anomalies at every possible length, however this seems to be an untenably slow idea. In Chapter 3, we will present MADRID, a novel anytime algorithm that can quickly identify anomalies of all lengths.

1.3 Encoding Information to Improve the Efficiency and Accuracy of Time Series Anomaly Detection

Perhaps surprisingly, while there are now over a thousand papers on anomaly detection, few of them attempt to define "anomaly" other than to echo the boilerplate definition of "Something that deviates from what is normal".

We argue that it is necessary to augment this definition to "Something that deviates from what is normal, given the users implicit and explicit knowledge of the domain". This is a subtle but critical distinction. We believe that the anomaly detection task is not to predict something about the data per se. The anomaly detection task should be to predict the user's discernment of the data. This change of focus offers two challenges: How can we model such information, and how can we communicate it to the algorithm? Note that the user's discernment of the data can be subjective, and depend on the domain and the user's role (engineer, manager, insurance underwriter etc.)

In Chapter 4, we will introduce FIRE, a novel framework for annotating and interpreting time series data, to enhance the efficiency and accuracy of time series anomaly detection tasks.

2. DAMP: Accurate Time Series Anomaly Detection on Trillions of Datapoints and Ultra-fast Arriving Data Streams

In this chapter, we introduce DAMP (Discord Aware Matrix Profile), a novel algorithm which solves all the problems listed in Section 1.1 and has the following useful properties.

- DAMP is not confused by repeated anomalies (twin-freaks), it simply flags the first occurrence. If desired, other occurrences can then be found by simple similarity searches. These other occurrences can be clustered, average, or otherwise summarized as appropriate.
- DAMP is defined for both online and offline cases. Moreover, DAMP has an extraordinary fast throughput, exceeding 300,000 Hz on standard hardware.
- As the previous bullet point suggests, DAMP is extraordinarily scalable. For the first time, this allows us to consider datasets with millions, billions and even trillions of datapoints.

The rest of this chapter is organized as follows. In Section 2.1 we motivate the use of *discords* as the time series anomaly definition most worthy of acceleration and generalization. We also concretely define a new term, *effectively online*, that allows DAMP to tackle ultra-fast real-time data sources found in industry and science. Section 2.2 contains the necessary definition and notation required, and Section 2.3 discusses related work, before we introduce our algorithm in Section 2.4. In Section 2.5 we conduct the most ambitious empirical evaluation of time series anomaly detection ever attempted.

2.1 Motivation

Before we continue, it is necessary to answer the following question. Why do we attempt to fix discord's scalability issues instead of inventing a new algorithm, or making one of the many dozens of more recently proposed methods more scalable?

The reason is that there is increasing evidence that discords remain competitive with the state-of-the-art[2][21]. 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 uses discords to find anomalies in climate data [17]. 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*" [9]. A group based in Halmstad University created a tool called IUSE for applying discord discovery to industrial datasets. One of their first applications was to a City Bus Fleet dataset, where they noted that the discords discovered did indeed have an objective meaning "*The discords in this case primarily consisted of significant drops of pressure … likely correspond to the drainage of the wet tank*." [24]. Finally, a team of researchers at the National Renewable Energy Laboratory, in Golden, Colorado, have used discords to find anomalies in a large building portfolio, showing that they could discover anomalies with diverse causes caused by both "*internal (occupant behavior) and external factors (weather conditions)*." [28]. There are several

[2] Note that some papers misattribute the success of their anomaly detection to the Matrix Profile or to HOTSAX, but these are simple different algorithms to compute time series discords.

other time series anomaly detection algorithms that are well cited [14][30], but most of the citations are from rival methods comparing these algorithms on a handful of benchmarks [35]. It is not clear that anyone actually uses these algorithms to solve real-world problems, as a detailed literature search does not produce any examples.

In addition, time series discords seem to be the only anomaly detection algorithm that has been demonstrated to perform at superhuman levels [21]. All other algorithms that we are aware of have shown to discover anomalies that are also readily apparent to the human eye. For example, a recent paper proposed a LSTMs network for anomaly detection and evaluated it on data retrieved from Mars [14]. However, the only anomaly shown in the paper shows a visually obvious anomaly where a repeated periodic pattern suddenly transitions to a literal flatline. Of course, this does not mean that such algorithms have no value, as human attention is very expensive. However, the literature also offers some examples where discords have found anomalies that are very subtle, defying the possibility of human discovery. For example, in [21], their Figure 8 and Figure 9 both seem to meet that criterion. For completeness, we will show some additional examples. Consider Fig. 2, which shows the vibration of an industrial motor [7][23].

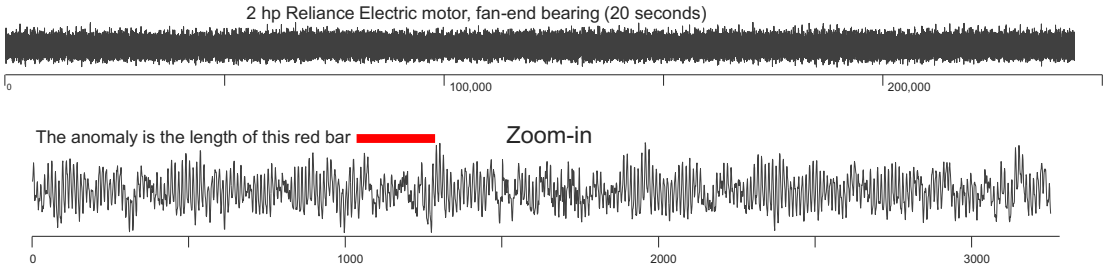

Fig. 2 *top*) A 20-second run of an industrial motor. *bottom*) a zoom-in of the region known to contain an anomaly, which is the length of (but not necessarily at the location of) the red bar. Here $m = 300$.

The data comes for a motor running under no load, however for a brief instant a load was applied and immediately removed, creating an anomaly. It is clearly fruitless to visually search for the anomaly in the *full* dataset, however, even if we zoom into a local region containing the anomaly, it is not clear where it is. In Fig. 3 we task time series discords with detecting the anomaly.

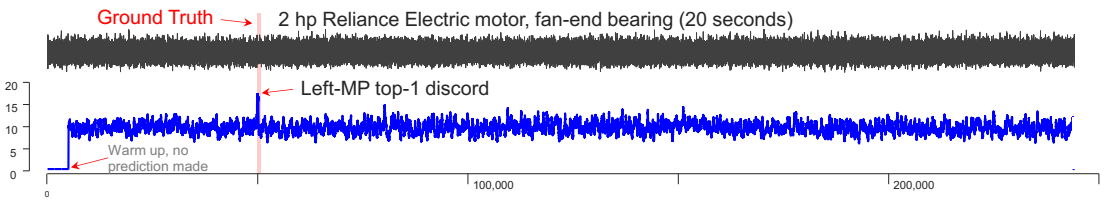

Fig. 3 *top*) A 20-second run of an industrial motor. *bottom*) The time series discord discovered by the Left-MP correctly locates the anomaly. Note that higher values are more anomalous. Here $m = 300$.

Beyond the accuracy of discords prediction here, note that this dataset contains 244,189 datapoints, representing about 20 seconds of wall clock time recorded at 12,000 Hz. We are not aware of any anomaly detection algorithm in the literature that could process this dataset in real-time, however, as we will show, DAMP *can*.

We also consider a dataset that is dramatically different to the bearing data. In Fig. 4 we show the Left-MP for an ECG which we know contains a single anomaly beat, a *ventricular contraction*.

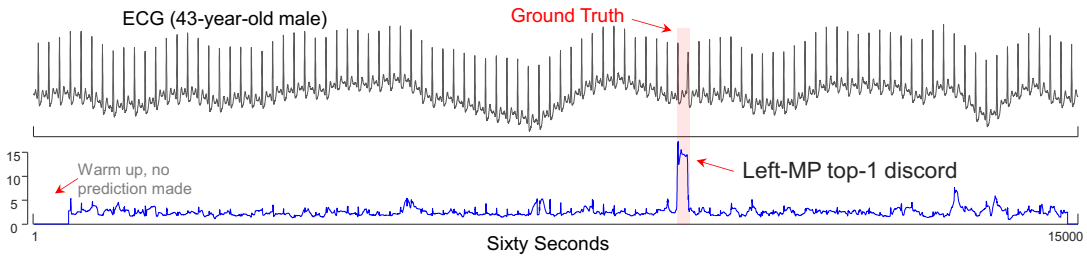

Fig. 4 *top*) A sixty-second snippet of an ECG. *bottom*) The top-1 time series discord correctly locates the anomaly. Here $m = 150$.

This dataset has a wandering baseline which is diagnostically meaningless, but which distracts the human eye (and many algorithms). However, once again time series discords have no problem detecting the anomaly, which noted cardiologist Dr. Gregory Mason says is on the cusp of his ability to detect by eye.

Finally, in Fig. 5 we consider a dataset that was explicitly created with the sole purpose of having anomalies that are "*difficult to spot for the human eye*" [31]. Here again discords are superhuman.

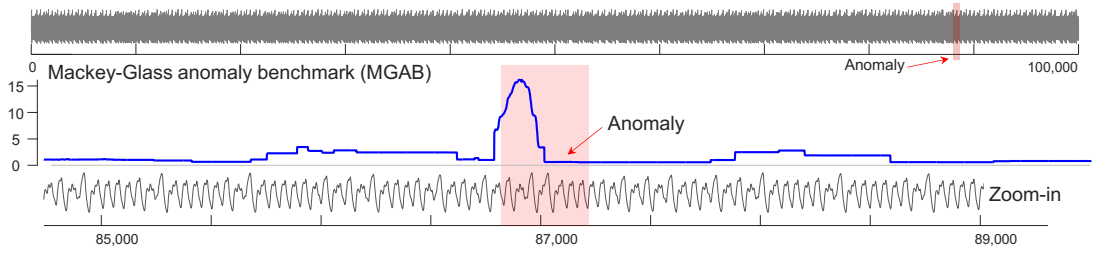

Fig. 5 *top*) The MGAB dataset was built to defy visual discovery of anomalies. *bottom*) The Top-1 time series discord correctly locates the anomaly. Here $m = 40$.

In summary, both the recent literature and our experiments suggest that time series discords are *at least* competitive with recently proposed algorithms, and thus worthy of accelerating to allow discords to be discovered in settings that are currently infeasible.

2.1.1 Effectively Online Anomaly Detection

Let us take a moment to make clear what the terms *batch* and *online* mean. If we are tasked with finding the top-*k* anomalies in a batch setting, we have random access to all data. For example, we could initially define April 1st as an anomaly, but when we later see data from say the summer months, we can change our mind, revisit April 1st, and reduce its anomaly score. For that matter, we could revisit April 1st, and *increase* its anomaly score. In contrast, in the *online* case we see the data incrementally arrive and must make an irrevocable decision as to the appropriate anomaly score. When recording this score, we do have access to all the data previously seen, but clearly we cannot see any future data. For some time series anomaly detection algorithms this distinction is important, and the algorithm can give different answers in the two settings. However, as we will show, the algorithm we propose in this chapter will produce the exact same answers in either setting.

Now that the terms *batch* and *online* are clear, it is helpful to introduce a new term, *effectively online*. A true online algorithm reports the instant it detects a monitored condition. However, let us imagine the following scenario: After a difficult cardiac surgery, a doctor decides she wants to monitor her patient for anomalous heartbeats, which may be an indication of postoperative Cardiac Tamponade (CT). If the patient does have an ECG suggestive of CT symptoms, the doctor has perhaps eight to ten minutes to confirm CT with an ultrasound and perform pericardiocentesis, a procedure done to remove fluid that has built up in the sac around the heart [18]. Clearly, in this situation an algorithm that reported an anomalous heartbeat ten minutes after its appearance would be

unacceptable. However, an algorithm that reported an anomalous heartbeat at most two seconds after it appears would be just as good as a true online algorithm. As such we propose the following definition:

Definition 1: An algorithm is said to be *effectively online*, if the lag in reporting a condition has little or no impact on the actionability of the reported information.

Note that the scale of the permissible lag is problem dependent. In the above scenario, two seconds made sense to the cardiologists we consulted. In an ultrafast arriving data stream, the permissible lag may be as little as 0.1 seconds, and for telemetry arriving from devices with a slow cycle rate, say the daily periodicity of pedestrian traffic, the permissible lag may be minutes to hours.

We suspect that many algorithms that are referred to as online in the literature, are really effectively online. The above discussion allows us to frame our contribution. Our proposed algorithm DAMP is parameterized by a single variable called *lookahead*.

- If *lookahead* is zero, DAMP is a fast *true* online algorithm.
- If *lookahead* is allowed to be arbitrarily large, DAMP is an ultrafast batch algorithm. We should not be surprised that a batch algorithm can be much faster, as it has access to all the information at once.

And now the *raison d'etre* for our digression:

- Even if *lookahead* is a small (but non-zero) number, DAMP is effectively online algorithm, yet it retains most or all the speedup of the arbitrarily large *lookahead* algorithm.

As we will show, DAMP allows for the discovery of time series discords in ultra-fast-moving streams for the first time.

2.2 Definitions and Background

We begin by defining the key terms used in this chapter. The data we work with is a *time series*.

Definition 2: A *time series* T is a sequence of real-valued numbers t_i : $T = [t_1, t_2, \ldots, t_n]$ where n is the length of T.

Typically, we consider only local *subsequences* of the times series.

Definition 3: 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 most human actions, ½ second may be appropriate, but for classifying transient stars, three days may be appropriate.

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

Definition 4: *Distance profile* D_i for time series T refers to an ordered array of Euclidean distances between the query subsequence $T_{i,m}$ and all subsequences in time series T. Formally, $D_i = d_{i,1}, d_{i,2}, \ldots, d_{i,n-m+1}$,where $d_{i,j}$ $(1 \leq i, j \leq n - m + 1)$ is the Euclidean distance between $T_{i,m}$ and $T_{j,m}$.

For 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. Our algorithm neglects these matches of the query and itself, and instead focuses on *non-self match*.

Definition 5: *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 $|p - q| \geq m$.

With the definition of non-self match, we can define *time series discords*.

Definition 6: *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 the distance between $T_{d,m}$ and its nearest non-self match is maximum. That is, $\forall$ subsequences $T_{c,m}$ of T, non-self matching set M_D of $T_{d,m}$, and non-self matching set M_C of $T_{c,m}$, $min(d_{d,M_D}) > min(d_{c,M_C})$.

Although there are many ways to locate time series discord, the most effective one recently is the *matrix profile* [39].

Definition 7: A *matrix profile P* of a time series T is a vector storing the z-normalized Euclidean distance between each subsequence and its nearest non-self match. Formally, $P = [min(D_1), min(D_2), \ldots, min(D_{n-m+1})]$, 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 matrix profile is the time series discord.

As we will explain below, we can compute a special matrix profile which only looks to the past. We call it the *left matrix profile*.

Definition 8: A *left matrix profile* P^L of a time series T is a vector that stores the z-normalized Euclidean 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 $P^L = [min(D_1^L), min(D_2^L), \ldots, min(D_{n-m+1}^L)]$.

Note that the term discord in this chapter refers to the highest value on the left matrix profile P^L, i.e., left-discord. For the sake of simplicity, we will refer to left-discord as discord where there is no ambiguity. It is clear that in the *online* case, we must use the Left-MP. However, here we argue that even in the *offline* case we should use it. To see why, consider the example shown in Fig. 6.

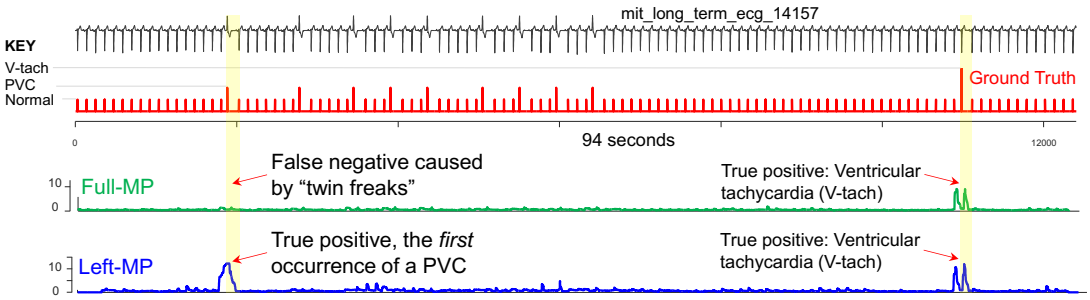

Fig. 6 *top to bottom*) A snippet of ECG with two types of anomalous heartbeats indicated by a ground truth vector. A full Matrix Profile can find the sole occurrence of V-tach, but is confused by the multiple occurrences of Premature Ventricular Contractions (PVCs), i.e., twin-freaks, and cannot find them. In contrast, the Left-MP flags the first occurrence of a PVC and the first (and only) V-tach. Here m = 150.

Here left-discords solve the twin-freak problem by reporting the *first* occurrence of the anomaly (later occurrences, if of interest, can be trivially found with subsequence search/monitoring).

2.3 Related Work

In recent years, there has been a surge of research interest in the topic of time series anomaly detection. For a detailed review, we refer the interested reader to [1][2][4][14][21][31] and the references therein. In addition to the work listed in Section 2.1, we have also compiled a longer annotated biography at [10] that explicitly discusses discords.

There are two important points that we have gathered from our survey of the literature. The first is due mostly to a single paper [35], 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.

Beyond four issues that [35] notes with benchmarks datasets, we wish to add another issue. Most of these benchmarks are minuscule. We suspect that the small datasets that the community has focused on are at least partly due to the poor scalability of current approaches. For example, a recent paper examines time series of length 140,256 and notes "*Given the length of the dataset, we sub-sample it by a factor 10*." [1]. This paper is by a research group at Amazon, who presumably does not lack for computational resources. For reference, it takes our proposed algorithm 0.9 seconds on the *full*-sized version of this dataset [10] on a commodity desktop.

In addition to the problems caused by using poor quality benchmarks, a recent paper suggests yet another compelling reason why much of the recent apparent success of recent research efforts should be viewed with caution. Paper [12] notes that "*most recent approaches employ an inadequate evaluation criterion leading to an inflated F1 score.*

(however) a rudimentary Random Guess method can outperform state-of-the-art detectors in terms of this popular but faulty evaluation criterion.".

A recent SIGKDD workshop keynote makes a related point about evaluation [16]. Suppose you have a year of data monitoring an industrial boiler, and it happens that on Christmas, the boiler leaks all day, causing an anomaly. One might imagine the best way to evaluate an algorithm on the task of discovering this anomaly would be a binary score, success/failure. However, many papers essentially consider each datapoint as if it was an independent event. Suppose they predicted all of Xmas day, and the first minute of the next day was an anomaly. They would report an F1 score of 0.9997. The four significant decimal digits imply some extraordinarily careful and significant measurement was made. However, with a little introspection will allow the diligent reader to see that this precision is unwarranted and misleading. The TSAD literature is replete with impressively large tables of numbers with four (and sometimes, five or six!) digits, that simply give the illusion of progress and rigor.

It is somewhat surprising that so few papers in the literature discuss time complexity. This can possibly also be attributed to issues with the benchmark datasets. For example, by far the two most discussed datasets in the literature are Yahoo and NY-Taxi (NAB), with lengths of 1,200 and 10,321 respectively. Even the most sluggish of algorithms are unlikely to be taxed by such tiny datasets. If building a particular highly-quality anomaly detection algorithm had a high *one-time* cost, then we might be willing to throw whatever computational resources are needed at the task, and then deploy the model in perpetuity. However, the situation is worse than that. In virtually any domain, the model will become

stale due to concept drift, and need to be periodically retrained, either on a regular schedule (say once a week), or when the model detects that it has drifted from the newly arriving data.

Recently a handful of papers have recognized that the slow training times for deep learning anomaly detectors can be an issue. For example, [32] notes that "*fast training times* (are needed) *to cope with the requirement of frequently re-updating the learning model*.". These authors then went on to introduce a "light-weight" anomaly detection system that can complete training in as little as twenty minutes (using GPUs) in a dataset of size 274,627. This kind of time frame may work for some domains, for example the three-year-long energy grid/weather data we consider in Section 2.5.1 Energy Grid Dataset. We surely could afford a few hours to build the model, and perhaps a few hours at the end of each month to retrain it. However, consider the machining dataset we examine in Section 2.5.2 Machining Dataset. Here we see the first thirty seconds of data, and then must *instantly* have a working model. While DAMP *can* do this, it is not clear that any other anomaly detector in the literature can. One might imagine that other methods could potentially look only at say, the first twenty seconds of data, and use the remaining ten seconds to build their model. However, this would require most of the algorithms in the literature to be accelerated by several orders of magnitude.

Finally, the reader may wonder why we do not test on the large collection of datasets during [26] in our empirical section. There are two reasons. First, the data collection includes datasets that [35] notes are deeply flawed, including mislabeled ground truth. If a significant fraction of the datasets have mislabeled ground truth, as Wu and Keogh

point out [35], and which the authors of [26] have acknowledged [25], it is hard to have any faith in evaluation on the overall data collection. For at least some of the datasets in this collection, including NAB-NYTaxi, NASA-MSL (trace G-1), YAHOO (A1-real46), it is known that at least 50% of the ground truth labels are incorrect [35]. With that amount of mislabeling, it would be fruitless to claim that one algorithm is superior to another because it was say 6.3% better than another. In any case, testing on small synthetic or unrealistic datasets seems pointless when we can test on large real datasets, as we do in this chapter.

In Table 9 we will compare to several rival methods. We refer the interested reader to the original papers for more detailed descriptions, but below we present a terse description of these rival methods.

An Auto-Encoder (AE) is a neural network architecture consisting of a combined encoder and decoder [2]. The encoder maps the input windows into a set of latent variables, while the decoder maps the latent variables back into the input space as a reconstruction. The difference between the input window and its reconstruction is the reconstruction error. The AE learns to minimize this error. The anomaly score of a window is the corresponding reconstruction error. A window with a high score is considered abnormal.

The Unsupervised Anomaly Detection (USAD) [2] extends the AE concept and constructs two AEs sharing the same encoder. The architecture is driven in two phases. In the first phase, the two AEs learn to reproduce the normal windows. In the second phase, an adversarial training teaches the first AE to fool the second one, while the second one

learns to recognize the data coming from the input or the reconstructed by the first AE. The anomaly score is the difference between the input data and the data reconstructed by the concatenated AEs.

Long Short-Term Memory Variational Auto-Encoders (LSTM-VAE) uses an LSTM to model temporal dependency [27], whereas the VAE projects the input data and its temporal dependencies into a latent space. During decoding, the latent space representation allows to estimate the output distribution. An anomaly is detected when the log-likelihood of the current data is below a threshold. LSTM-VAE's have the capacity to identify anomalies that span over multiple time scales [27].

Telemanom is a Long Short-Term Memory (LSTMs) networks, a type of Recurrent Neural Network (RNN) [14]. Once model predictions are generated, we offer a nonparametric, dynamic, and unsupervised thresholding approach for evaluating residuals.

NORMA can be thought of as a variant of a Golden Batch Matrix Profile, which uses a clustering preprocessing step to compact the training data into a small, therefore quickly searched, reference dataset [4].

SCRIMP is a fast method to compute the classic Matrix Profile.

2.4 DAMP

2.4.1 Intuitive Overview of DAMP

Before giving a formal explanation of our algorithm, we will first provide an intuitive description of how it works. We will start with discussing the batch case and then further

generalize to the (minor) steps required for the online case. As shown in Fig. 7, it will be helpful to explain the algorithm mid-execution, as it is processing the subsequence T_i.

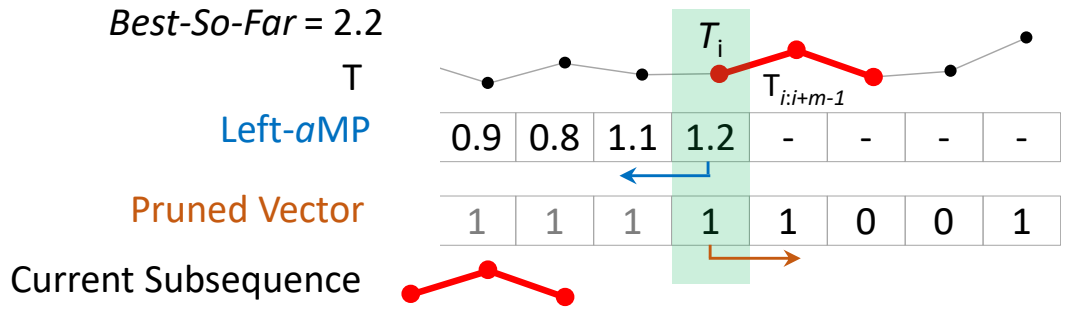

Fig. 7 A sketch of the DAMP algorithm in progress, processing the current subsequence. *top*) The time series T. *center*) The Left-*a*MP, its values between 1 and *i* are computed, its values after *i* have yet to be computed. *bottom*) the Pruned Vector indicates subsequences that can be ignored without affecting the final result.

Fig. 7.*top* shows the time series T being processed, the green bar indicating the current subsequence being processed at location i. Note that we have created two parallel vectors to accompany T. The Left-*a*MP is the vector we are computing. It is an approximation to the true Left-MP, with the following properties:

- If location j is the true left-discord for the time series $T_{1:j}$, then the discord value at aMP_j is not an approximation, but the true left-discord value.
- Otherwise, the approximation at aMP_j is strictly bounded: $MP_j \leq aMPj \leq \max(MP_{1:j})$

These properties tell us that we can take any prefix of T (inducing the special case of the entire length of T), and the left-discord reported by the Left-*a*MP will be the same as that reported by the Left-MP.

In Fig. 7.*bottom* we show the other parallel vector that accompanies T and the Left-aMP_j. The Pruned Vector tells us which subsequences could not be the left-discord, and hence

do not need to be processed. At initialization time, this vector is set to all '1's, indicating that all subsequences must be processed. However, as we process the data, we may be able to "peek into the future" and cheaply determine locations that could not be a discord, and flip their corresponding bits to '0'.

At the i^{th} location, the processing can be divided into two independent steps, *backward* processing and *forward* processing.

2.4.2 Backward Processing

The main task of backward processing is to discover whether the current subsequence $T_{i:i+m-1}$ is the left-discord, for which the naïve way would be to compute its nearest neighbor distance to any subsequences in $T_{1:i}$.

However, note that in general we may not need to find the nearest neighbor, any neighbor whose distance is less than the *Best-So-Far* will disqualify the current subsequence from being the discord. This suggests an early abandoning scheme that we can optimize with the two following observations:

- Instead of incrementally searching from the beginning, we should expect to be able to abandon earlier if we search *backwards* from the i^{th} location. The reason this is true is that the patterns can drift over time. In other words, the pattern most likely to be similar to the current subsequences is generally the subsequence *just* before the current subsequence[3].

[3] This observation is true for heartbeats, gaits, machine cycles etc. One exception is for events tied to a cultural calendar. For example, for taxi demand or electrical power demand, the most similar day to any given day, is not the previous day, but the same day one week earlier.

- The MASS algorithm is optimized for queries with powers of two length. For example, using the machine that performed all the experiments in this chapter, we find that a MASS search with a query of length 512, takes 0.025 seconds for a time series of length 524,288 (i.e., 2^{19}). But if we delete a single point to get a 524,287, it takes 0.177 seconds. This suggests we should attempt to construct a backward search algorithm that is comprised mostly or solely of such $p^{integer}$ length queries.

These two observations suggest an algorithm. We should look backwards at the prefix that is the next power-of-two longer than m. If that yields a neighbor that is less than the *Best-So-Far (BSF)* we are done, we simply place that value in $a\text{MP}_i$ as our approximation. If that was not the case, we double the length of the prefix to *two* times the next power-of-two longer than m, and try again. We continue to iteratively double until we find a nearest neighbor distance that is less than the *Best-So-Far*, or until our prefix includes the full span back to the beginning of T. In that latter case, we use the nearest neighbor distance to update both the *Best-So-Far* and $a\text{MP}_i$.

2.4.3 Forward Processing

In the forward processing step, we attempt to discover and prune subsequences that cannot be left-discord. If we take the current subsequence and compare it to the suffix of T, that is, to $T_{i+m:n}$ (the search must start at $i+m$ to avoid self-match), any subsequence that is less than the *Best-So-Far* distance to the current subsequence can be pruned (have its corresponding bit in the Pruned Vector set to '0').

In principle, we could do this search from $i+m$ to the end. However, the two observations in the previous section still apply. While the next few cycles may be similar and yield a good pruning rate, over time the patterns tend to drift and the pruning rate falls. The combination of a long expensive similarity search and the lower pruning rate means that the forward step may not "pay" for itself. So instead, we can look forward a limited amount, say *four* times the next power-of-two longer than m.

After completing both the backward and forward processing, the algorithm increments the current pointer from i to the next index which has a '1' in the Pruned Vector, and repeats the two processing steps.

2.4.4 Formal Pseudocode for DAMP

Here we give the pseudocode shown in Table 1 to formalize the intuition of the previous sections. For ease of explanation, we first consider only the batch case.

Table 1: The Main DAMP Algorithm

```
Function:  DAMP(T, m, spIndex)
Input:  T:   Time series
        m:   Subsequence length
        spIndex: Location of split point between training and test data
Output: aMP: Left approximate Matrix Profile
1  PV = ones(1,length(T)-m+1)
2  aMP = zeros(1,length(T)-m+1)
3  BSF = 0              // The current best discord score
4  // Scan all subsequences in the test data
5  For i = spIndex to length(T) - m + 1
6     If NOT PV_i       // Skip the pruned subsequence
7        aMP_i = aMP_i-1
8     Else
9        [aMP_i, BSF] = BackwardProcessing(T, m, i, BSF)
10       PV = ForwardProcessing(T, m, i, BSF, PV)
11 return aMP
```

In lines 1 and 2 we initialize two vectors that are essentially the same length as the time series *T*, but are actually of length *n*-*m*+1. These are *PV* (Pruned Vector), a Boolean vector that indicates which indices can be dismissed without evaluation, and *a*MP, which is the approximate Matrix Profile we wish to compute. The current highest discord score encountered during execution is stored in the *BSF*, initialized to zero in line 3.

In lines 5 to 10, we iterate through all subsequences of length *m* in the test data. In each iteration, we first determine whether the current subsequence was pruned, i.e., whether it is marked as 0 in the *PV* (line 6). If yes, we assign the discord score of the previous subsequence to the current subsequence and then skip to the next subsequence (line 7). If the current subsequence was not pruned, we must process it. In line 9 we call BackwardProcessing to calculate the discord score of the current subsequence. In particular, if the backward search finds a value higher than the current highest discord score (*BSF*), BackwardProcessing returns the *exact* score of the current subsequence and updates the *BSF*; otherwise, BackwardProcessing returns an approximate score of the current subsequence and does not update the *BSF*. Note that while this score is approximate, it is bounded between the true score and the current *BSF*.

At this point we have completely processed the current location. However, before we increment our loop index to process the next location, we take a brief digression. We will use the current subsequence to look "forward", finding any subsequences ahead of it that have a distance to it that is less than the current *BSF*. It is easy to see that any such subsequences could not be a better discord than the current *BSF*, as when they do BackwardProcessing, they would find the current subsequences to be close enough to

disqualify them. This observation allows us to prune these "near-enough" neighbors of the current subsequence. Concretely, line 10 invokes ForwardProcessing to find out the subsequences that can be pruned within a specific range in the future (if any), and their corresponding vectors are marked as 0 and recorded in the Pruned Vector *PV*. Finally in line 11 we return the left approximate Matrix Profile computed by the DAMP algorithm.

Table 1 provides a high-level overview of how the DAMP algorithm works. Let us now "zoom in" and look at the two core subroutines of DAMP, BackwardProcessing and ForwardProcessing. We begin with Table 2 to explain backward processing, whose intuition we laid out in Section 2.4.2 Backward Processing.

Table 2: DAMP Backward Processing Algorithm

```
Function:  [aMP_i, BSF] = BackwardProcessing(T, m, i, BSF)
Input:  T:   Time series
        m:   Subsequence length
        i:   Index of current query
        BSF:       Highest discord score so far
Output: aMP_i:Discord value at position i
        BSF:       Updated highest discord score so far
```

```
1   aMP_i = inf
2   prefix = 2^nextpow2(m)  // Initial length of prefix
3   While aMP_i ≥ BSF
4      If the search reaches the beginning of the time series
5         aMP_i = min(MASS(T_1:i,T_i:i+m-1))
6         If aMP_i > BSF  // Update the current best discord score
7             BSF = aMP_i
8         break
9      Else
10        aMP_i = min(MASS(T_i-prefix+1:i,T_i:i+m-1))
11        If aMP_i < BSF
12            break      // Stop searching
13        Else           // Double the length of prefix
14            prefix = 2*prefix
15  return aMP_i, BSF
```

In line 1 we begin by initializing the discord score of the current query at position *i* to positive infinity. Then in line 2 we specify the initial length of the backward processing

and store it in the variable *prefix*. We employ 2^nextpow2(*m*) to define this initial length. Specifically, when we feed the subsequence length *m* into 2^nextpow2(*m*), it will return the smallest power of 2 greater than *m*. Recall that we are doing this because MASS is significantly faster when the length of the time series is a power of two. Since we are going to do a "piecewise" search of the time series that precedes the subsequence being processed, it makes sense to make these pieces be a power of two in length.

The loop in lines 3-14 evaluates the exact or approximate discord score of the current query. Here we adopt the idea of "iterative doubling". At the beginning, we find the nearest neighbor of the current query in the initial length *prefix* and save the distance between the current query and the nearest neighbor into $a\text{MP}_i$ (line 10). If this distance is lower than the current highest discord score, this means that we find a nearest neighbor for the current query within *prefix* that is more similar than the current discord and its nearest neighbor, so it cannot be a discord, and the iteration terminates (lines 11-12). However, if the distance between the query and its nearest neighbor $a\text{MP}_i$ is higher than the current highest discord score *BSF*, we double the length of the backward processing and continue the search in the next iteration (lines 13-14). This idea is shown in Fig. 8.

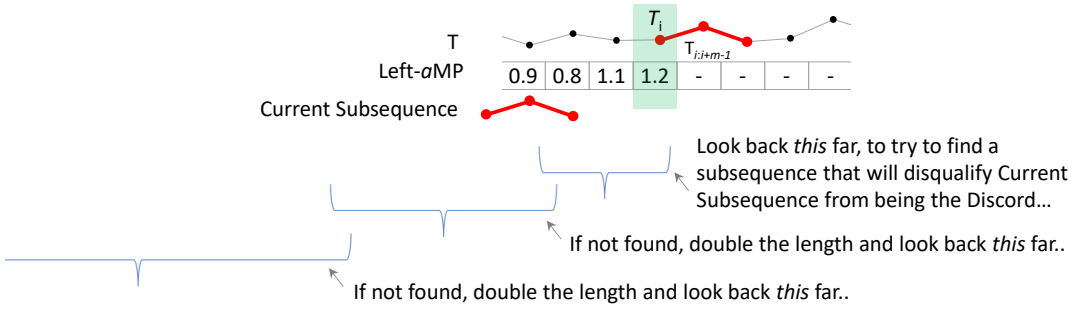

Fig. 8 A visualization of the *iterative doubling* search policy used in lines 10-14 of Table 2. See also Fig. 7.

We keep iteratively doubling until we compute a score smaller than the *BSF* within the range *prefix*, or search to the beginning of the time series *T*. If the search gets to the beginning of the time series, we first find the nearest neighbor of the query from position 1 to *i* and store the distance to the nearest neighbor in $a\text{MP}_i$ (lines 4-5). After that, we will check whether $a\text{MP}_i$ is still larger than *BSF* (line 6). If yes, this means that we cannot find a nearest neighbor that is similar enough to the current query, and clearly, the current query is the new discord. In this case, we will update the highest discord score and break out of the loop (lines 7-8). Finally, line 15 returns the result of backward processing, the score of the current query $a\text{MP}_i$, and the current highest discord value *BSF*.

Note that if the search reaches the very beginning of the time series, our computation is performed in the global region (from 1 to *i*), not in the local region *prefix*, in which case the discord score of the current query $a\text{MP}_i$ is an *exact* value; whereas if our score is computed in the local region *prefix*, $a\text{MP}_i$ is an approximate value, but bounded between the true score and the current *BSF*.

If we *just* use the backward processing step (line 9 of Table 1), then we have a fast online algorithm to compute the *a*MP. However, the use of forward processing as outlined in Table 3 can speed up the processing by at least a further order of magnitude. This is the algorithm whose intuition was laid out in Section 2.4.3 Forward Processing. The purpose of forward processing is admissible pruning. That is, if there is evidence that some future subsequences cannot be a discord, we will ignore these subsequences and no longer perform expensive processing on them. To achieve this in line 1 we need to define *lookahead*, the range of how many subsequences to peek ahead.

Table 3: DAMP Forward Processing Algorithm

```
Function:  PV = ForwardProcessing(T, m, i, BSF, PV)
Input:  T:   Time series
        m:   Subsequence length
        i:   Index of current query
        BSF:       Highest discord score so far
        PV:  Pruned Vector
Output: PV:  Updated Pruned Vector
1   lookahead = 2^nextpow2(m)       // Length to "peek" ahead
2   If the search does not reach the end of the time series
3     start = i + m
4     end = min(start + lookahead - 1,length(T))
5     D'_i = MASS(T_start:end, T_i:i+m-1)     // Definition 4
6     indices = all indices in D'_i with values less than BSF
7     indices = indices + start - 1     // Convert indices on distance
8                                       //profile to indices on time series
9   PV_indices = 0                    // Update the Pruned Vector
10  return PV
```

Here we also use 2^nextpow2(m), i.e., the smallest power of 2 larger than the subsequence length m. After that, we need to determine whether the forward search exceeds the range of T to ensure that our processing is safe and there is no out-of-bounds problem (line 2). Line 3 defines the start position of the forward search, namely *start*. To avoid self-matching, we set the *start* to the position after the end of the query, that is, $i+m$. Line 4 explicitly defines the end position of the forward search, and since the length of our forward search is *lookahead*, or n. We can easily conclude that *end* is *start* + *lookahead* - 1. In line 5, we calculate the distance profile D'_i by calling the MASS function.

The distance profile D'_i here is slightly different from the one described in Definition 4 because it is computed under a specific range. That is, D'_i stores the distance between the current query and all subsequences in the range of *lookahead* (from *start* to *end*) instead of the distance between the current query and all subsequences of T. Once the distance

profile D_i' is constructed, we can use it for pruning. Suppose there exist subsequences in the future that are more similar to the current query than the discord to its nearest neighbor. In that case, these subsequences cannot be a discord, so we can prune them. Therefore, we can use the current highest discord score *BSF* as a criterion to find all the indices in the distance profile with values lower than the *BSF* (line 6). Since the indices on the distance profile start at 1 and are not aligned with the true indices of the time series, we need an additional step in line 7 to convert the indices on the distance profile to the true indices of the subsequence. After line 7 we get a list of indices for the subsequences that can be pruned out. The Pruned Vector values at the corresponding positions specified in the list *indices* are set to 0 (line 9), indicating that when later iterations process the subsequences listed in *indices* we can simply skip them. At last, line 10 returns the updated Pruned Vector *PV*.

The forward processing algorithm has exactly one parameter, the *lookahead* length. How should we set this? In Fig. 9.*left* we sketch out the tradeoffs involved. A longer *lookahead* can prune more subsequences, but this comes at the cost of more expensive similarity searches. The good news is that the speedup is dramatic, that the sweet spot is early (given us effectively online detection), and that the exact value of the *lookahead* parameter is not too critical. All datasets we examined exhibit this "U-shaped" behavior, although the similarity searches. As Fig. 9.*right* shows, this intuition is borne out by experiment. The height of the base of the "U" can be lower (smooth and highly periodic data) or higher.

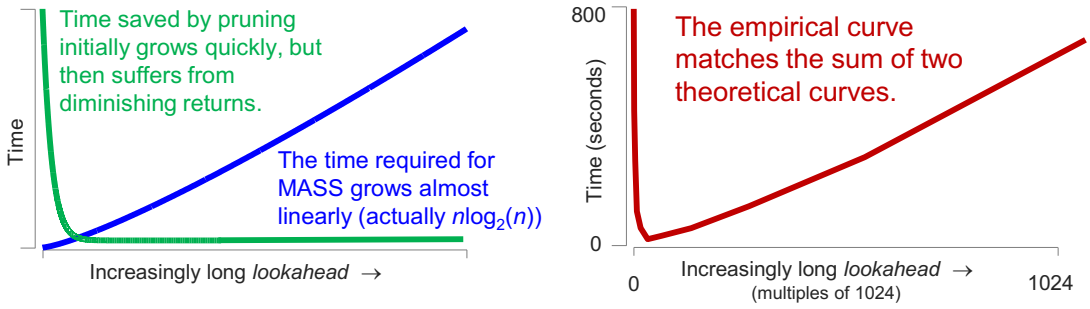

Fig. 9 *left*) The theoretical *lookahead* tradeoff is based on two factors. As the *lookahead* grows, the pruning rate becomes greater, but the cost of the similarity search increases. *right*) The empirically measured effectiveness of forward processing (on random walks of length 2^{20}) is indeed the sum of the two factors. Here m = 1,024.

Finally, this is a good place to mention an important caveat about interpreting a Left-*a*MP that is computed using forward processing. Failure to understand this caveat may lead a user to think the *a*MP is indicating an anomaly where there is none. Consider Fig. 10 which compares the results of Brute Force and DAMP with and without forward processing for a dataset from the KDD Cup 2021.

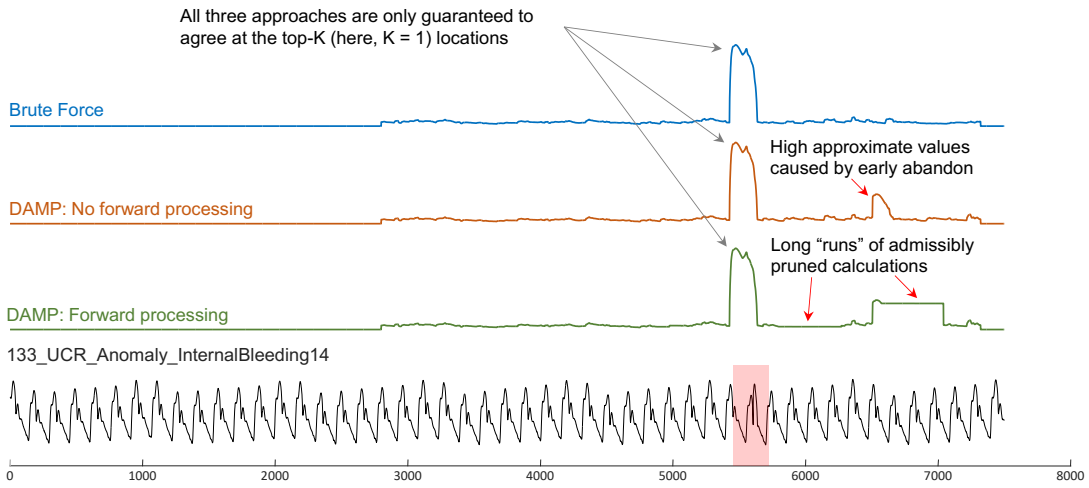

Fig. 10 *bottom-to-top*) UCR-133 dataset with an anomaly highlighted in red. The top-1 Left-*a*MP computed with forward processing. The top-1 Left-*a*MP computed without forward processing. The Left-MP computed without backward and forward processing. Both top-1 Left-*a*MPs have a secondary peak at around 6500, which seems to indicate an anomaly, whereas in fact they are caused by early abandon and are not meaningful. The top-1 Left-*a*MP computed with forward processing produces long constant runs that indicate that the algorithm admissibly skipped those regions. Here $m = 180$.

As can be seen from Fig. 10, the discord scores calculated by different approaches look different for the same data set. So how should we interpret these results?

First, the Brute Force illustrated by the blue curve is identical to the one shown in Fig. 23. It does not have forward and backward processing, and for each iteration, it searches from the current position to the beginning of the time series. Therefore, each value on Left-MP generated by Brute Force is an exact value. That is, for any of the peaks on Left-MP, there could be physical meaning that we can interpret.

By contrast, when using both DAMP algorithms to search for the top-k left-discords, the k highest peaks *do* correctly show the location and strength (the height of the peaks) of the top-k left-discords (in Fig. 10, $k = 1$). However, the remaining $k + 1$ peaks should not be assumed to indicate slightly smaller anomalies. This is because both DAMP methods perform the iterative doubling backward search, yielding either approximate or exact

discord scores. Whether the score is exact or approximate depends on two cases, which we detailed in Section 2.4.4 Formal Pseudocode for DAMP and will not repeat here. In brief, for most iterations of DAMP, the backward processing is terminated before it reaches the beginning of the time series, thus producing approximate scores on Left-*a*MP that are larger than the exact scores. Therefore, most peaks on Left-*a*MP except for the top-*k* ones are probably "false positives" due to this early abandoning scheme. For example, the Left-*a*MPs represented by the orange and green curves in the figure both have a secondary peak at around 6,500, which seems to indicate an anomaly at that location; however, by comparing it to the Left-MP results shown by the blue curve, it is clear that the scores at around 6,500 are actually below average. Thus, these secondary peaks cannot be interpreted as anomalies.

Further, we can observe a lot of piecewise constant regions on the Left-*a*MP generated by DAMP using forward processing, i.e., the green curve. They simply indicate regions that were pruned by encountering a matching subsequence that was below the current *Best-So-Far* and had no practical meaning. For example, at the end of the green curve, there are two long constant plateaus, one of which has a relatively high value. As we can see by comparing that region to the corresponding region in the topmost blue curve, we should not assume that there are any anomalies in that region.

Again, to summarize: The top-*k* peaks of the top-*k* Left-*a*MP should be interpreted as having the correct values of top-*k* discords of T, but the remaining values of the top-*k* Left-*a*MP have no meaningful interpretation.

2.4.5 The Time and Space Complexity of DAMP

Since all computation results are stored in a one-dimensional vector of size n, the space complexity of DAMP is just the size of the original data, O(n). The worst-case time complexity is O($n$log$n$) per datapoint ingested, the time required to do a full similarity search with MASS [19]. However, empirically, on diverse real-world datasets, more than 99.999% of the times we enter the loop in line 3 of Table 2 we will break out in the first iteration (line 12), making the algorithm effectively O($m$log$m$) per datapoint ingested, and linear in the time series length. Fig. 25 shows this linear assumption strongly holds up to at least $n = 2^{30}$.

2.4.6 DAMP Variants

There are more general cases that can be easily handled by modifying the basic DAMP, for example:

- The algorithm as explained in Table 1 is a *batch* algorithm. To make it an *effectively online* algorithm, we simply must reduce the size of the *lookahead* (Table 3, line 1) to the largest delay we are willing to accept (including possibly *zero* delay).
- The algorithm as explained in Table 1 computes the Left-*a*MP, however we can modify it to compute the classic Full-*a*MP. If the backward processing step reaches the beginning of the time series, instead of updating the *BSF*, we do the same type of iterative doubling search, but *forward* from the current index (not to be confused with forward pruning search in Table 3). We have made this code available at [10], but we do not consider it further here, due to page limits.

- It may be useful to limit how far back the backward processing can look, essentially redefining anomalies as "*the subsequence with the maximum distance to any of the X subsequences before it*". We call this variant the *X-Lag-Amnesic DAMP*.
- Instead of searching an ever-growing amount of previously seen data in the BackwardProcessing step, we can search a fixed pool of explicit training data. For example, an engineer could curate a dataset that contains all the allowable behaviors for a manufacturing process (i.e., the "*Golden Batch*").

There are several other useful variants that we have considered, and we suspect the community will quickly exploit the scalability of the basic DAMP algorithm to invent further variants.

Below we give more details about the two useful variants of DAMP, *X*-Lag-Amnesic DAMP and Golden DAMP mentioned above. To help the reader better understand how these two variants work, let us start with the most basic variant, namely, *Classic DAMP*.

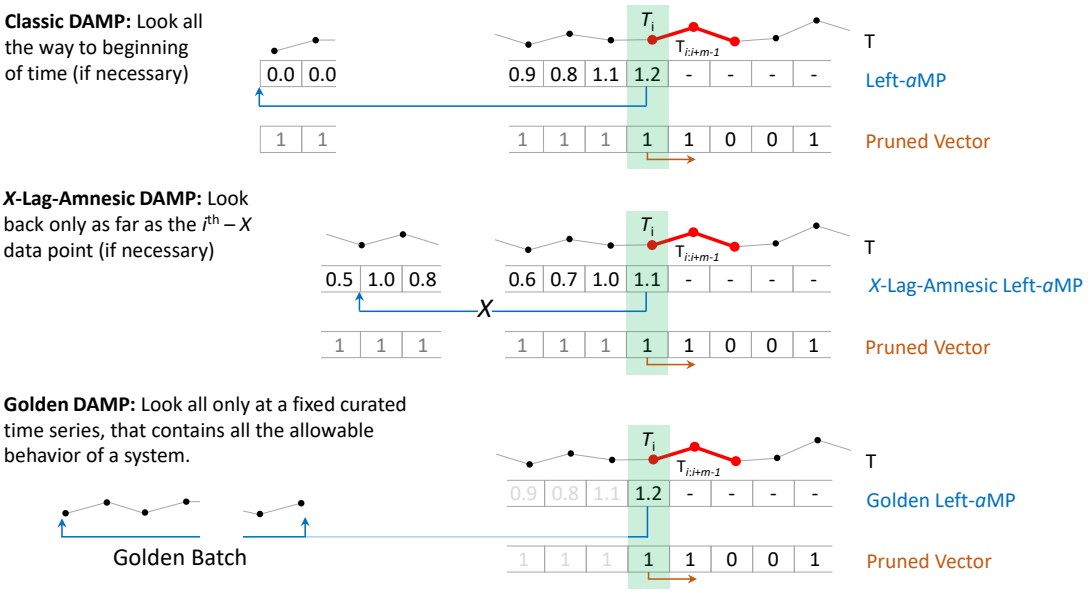

Fig. 11 Three variants of DAMP. *top*) Classic DAMP *middle*) *X*-Lag Amnesic DAMP *bottom*) Golden DAMP.

The Classic DAMP algorithm illustrated in Fig. 11.*top* was already discussed in Sections 2.4.1 Intuitive Overview of DAMP and 2.4.4 Formal Pseudocode for DAMP. It is worth noting here that for Classic DAMP, all data collected before the current time $T_{1:i-1}$ are our training data by default, and our backward search is executed on this progressively growing training data. This means that to calculate the discord score of the current subsequence $T_{i:i+m-1}$, Classic DAMP searches all the way forward from position i by the iterative doubling process, and, in the worst case, all the way to the beginning of the time series, i.e., $T_{1:i}$. Therefore, as we process more and more data points over time, our backward search may also require more and more time.

As we shall see in our experimental section, empirically this is not a problem on the dozens of datasets we consider. Nevertheless, *X*-Lag-Amnesic DAMP and Golden

DAMP allow us to provide a strict bound on the worst-case behavior, in addition to possessing other useful properties.

2.4.6.1 X-Lag-Amnesic DAMP

In some settings we may require an algorithm that can show us the most unusual behavior in *just* the last few minutes, days, months, or years. In that case, a DAMP variant that constrains how far back the backward search can look is required. Formally, we refer to such a DAMP variant as *X-Lag-Amnesic DAMP*.

Compared with Classic DAMP, the time overhead of *X*-Lag-Amnesic DAMP is bounded and controllable. This is because it only cares about what happened in a fixed unit of time before the present, and its calculation is based on fixed-size and real-time updated training data. For example, if we only need to find anomalies that occurred in the most recent month, *X*-Lag-Amnesic DAMP will perform an iterative doubling search in the most recent month's data rather than searching through all past data. Consequently, the time cost of *X*-Lag-Amnesic DAMP is bounded by the length of *X* as opposed to increasing gradually.

In addition, *X*-Lag-Amnesic DAMP can better deal with concept drift. For time series in some domains, their patterns change over time and the dependence between their data weakens as the distance increases, at which point it makes no sense to consider data that is too far from the present. For example, for many batch processes in the food and beverage industry the time series patterns are known to drift over each day, due to changes in ambient temperature and humidity. A pattern that happens during the nightshift may be anomalous because the process is "running hot". That is to say, it is

exhibiting behaviors that would be normal if it was in the middle of a hot day, but these behaviors are anomalous given that they are observed in the cool of the night. It might be obvious if we compare only to the patterns in the previous hour or so, but it will not be obvious if we allow comparisons back to the previous midday. Obviously, since *X*-Lag-Amnesic DAMP focuses only on what happened recently, it can avoid such issues caused by concept drift. By contrast, Classic DAMP is more vulnerable to this, as its backward search may cover all data that occurred before the present, and all these data have the same weight for the discord score calculation regardless of their proximity to the current subsequence.

Fig. 11.*middle* describes how the *X*-Lag-Amnesic DAMP works. Here we introduce a new parameter *X*, the maximum length that the backward processing algorithm can look back, specified by the user as needed. The framework of the *X*-Lag-Amnesic DAMP algorithm is the same as Classic DAMP; it retains the forward and backward processing steps, in which the forward processing is identical to Classic DAMP. The only difference between *X*-Lag-Amnesic DAMP and Classic DAMP is that for the current subsequence being processed $T_{i:i+m-1}$, we only perform a backward search on the *X* data points before it, not on all the previous data. However, the search is still iteratively doubled: it terminates either when it finds the nearest neighbor with a distance smaller than the *BSF* or when it reaches the beginning of *X*. Therefore, to make *X*-Lag-Amnesic DAMP work, we simply need to change lines 4-5 of Table 2 for Classic DAMP to the five lines shown in Table 4.

.

Table 4: Pseudo code snippet for X-Lag-Amnesic DAMP

1	**If** Starting position of the search < max(i-X,1) **Or** X < *prefix*
2	**If** $i - X < 1$
3	$a\text{MP}_i$ = min(MASS($T_{1:i}$,$T_{i:i+m-1}$))
4	**Else**
5	$a\text{MP}_i$ = min(MASS($T_{i-X:i}$,$T_{i:i+m-1}$))

In line 1 we added two new criteria for search termination, i.e., reaching the beginning of the time series $T_{i-X:i}$, or the maximum length of looking back X is less than the initial length of the iterative doubling search *prefix*. In both cases, we do not iteratively double our search anymore. We have reached the limit of the history we think should inform our decision. Instead, we only search for the nearest neighbor of the current subsequence in the range i-X to i (lines 4-5). Moreover, there is a special case where the number of data points that arrived has not yet reached X (i<X+1). In this case, we can only conduct the backward search in all available data $T_{1:i}$ as shown in lines 2-3.

Others works have noted the utility of amnesic anomaly detection (although not using that phrase), including the SAND algorithm of [5]. However, SAND requires significant effort to build a reference dataset, and the setting of several unintuitive parameters.

2.4.6.2 Golden DAMP

Recall that Classic DAMP has a parameter called *spIndex*, which sets the location of the split point between the training and test data in the initial state. When Classic DAMP processes a time series, it assumes that the data before *spIndex*, $T_{1:spIndex-1}$ are normal, which may lead to three issues. First, this causes the algorithm to ignore the potential anomalous behavior present in $T_{1:spIndex-1}$, resulting in certain false-negative results.

Second, this approach may have the algorithm wasting time searching redundant data. It is possible that the patterns in $T_{1:spIndex-1}$ are highly redundant, such as 1,000 heartbeats that are essentially identical. If the heartbeats all have the same pattern, it would suffice for the algorithm to take just one of them to learn[4]; there is no need to consider the same pattern 1,000 times, which will waste a lot of time. Further, it may be difficult for $T_{1:spIndex-1}$ to contain every normal pattern, which can cause the algorithm to incorrectly identify normal behavior that does not appear in $T_{1:spIndex-1}$ as an anomaly. For example, if $T_{1:spIndex-1}$ only contains data on the solar zenith angle during the day, the algorithm may incorrectly identify normal solar zenith angles at night as anomalies. These potential problems can undermine the accuracy and efficiency of the algorithm.

Golden DAMP is our proposed solution to the above three problems. It processes each subsequence not by referring to information that occurred before the current time, but to user-defined, curated, out-of-band information, denoted as *Golden Batch*. The Golden Batch implicitly defines every possible legal behavior, such as every possible dance move, every normal heartbeat, etc. It includes all the things the user expects to happen in the system. With this correct and comprehensive priori knowledge, the algorithm will be able to make more accurate and efficient decisions.

This idea of creating a curated collection of data that spans the space of all possible acceptable behaviors is well known in the process industry [37]. For example, food/beverage engineers will often set aside one day to create a recipe under all

[4] Actually, using exactly one heartbeat (or *pattern* more generally), may make the downstream algorithms brittle to the choice of the starting point of the heartbeat. To bypass this issue, we always extract two consecutive beats.

combinations of conditions encountered: under cool conditions, under hot conditions, with carbonated infeed, with flat infeed etc. However, the use of these batch profiles is typically human comparison of the evolving process to the Golden Batch(es) [37]. Here we are interested in automatic anomaly detection. In addition, note that while the Golden Batch data can be hand curated, it can also be created automatically by various numerosity reduction algorithms [15][36].

Further note that the execution time of Golden DAMP is also bounded because its training data is the Golden Batch with a fixed size. Therefore, as we explained in Section 2.4.6.2 Golden DAMP, the cost of Golden DAMP's backward search is proportional to the size of Golden Batch.

Fig. 11.*bottom* illustrates the idea of Golden DAMP. When processing the current subsequence $T_{i:i-m+1,}$ Golden DAMP no longer looks backward in the time series T but toward the Golden Batch, a vector containing all acceptable patterns. We still use the iterative doubling search policy shown in Fig. 8 for Golden Batch. The search keeps iteratively doubling until it finds the nearest neighbor within the *prefix* whose distance from $T_{i:i-m+1}$ is less than the *BSF*, or it gets to the beginning of the Golden Batch. After computing the approximate or exact discord score for position i, we invoke the same forward processing procedure as in Classic DAMP to disqualify future subsequences that are unlikely to become a discord.

The implementation details of Golden DAMP are given in

Table 5 and Table 6. Since most of them are the same as Table 1 and Table 2, we will highlight the parts that we changed

Table 5: The Main Golden DAMP Algorithm

```
Function:  Golden_DAMP(T, m, GoldenBatch)
Input:  T:   Time series
        m:   Subsequence length
        GoldenBatch:     A long time series with all possible normal
        patterns
Output: aMP: Left approximate Matrix Profile
```

```
1   PV = ones(1,length(T)-m+1)
2   aMP = zeros(1,length(T)-m+1)
3   BSF = 0                 // The current best discord score
4                           // Scan all subsequences in the test data
5   For i = 1 to length(T) - m + 1
6      If NOT PV_i          // Skip the pruned subsequence
7         aMP_i = aMP_i-1
8      Else
9         [aMP_i, BSF] = BackwardProcessing(T, m, i, BSF, GoldenBatch)
10        PV = ForwardProcessing(T, m, i, BSF, PV)
11  return aMP
```

The main framework of Golden DAMP is shown in Table 5. Golden DAMP has a new input, *GoldenBatch*, a long vector that joins all normal patterns together. As with Table 1, the algorithm starts with initialization in lines 1-3. Since we already have the training data *GoldenBatch*, we no longer need to use the first *spIndex*-1 data of the time series *T*. As a result, in line 5 we adjust the processing range of Golden DAMP from $T_{spIndex:n-m+1}$ to $T_{1:n-m+1}$. After that, within the loop, lines 6-7 decide whether to process the current subsequence $T_{i:i-m+1}$ according to the value in the pruned vector *PV*. If the subsequence at position *i* needs to be processed, we first invoke *BackwardProcessing* in line 9 to calculate the discord score for position *i* and update the current highest discord value, and then call *ForwardProcessing* in line 10 to determine the subsequences to be pruned in the future. Finally, lines 5-10 iterate through each subsequence in $T_{1:n-m+1}$ and line 11 returns the Golden Left-*a*MP. In particular, the *ForwardProcessing* here is identical to that of

Classic DAMP, so we do not repeat it below. However, we partially changed *BackwardProcessing* from Table 2 of Classic DAMP, so we give Table 6 detailing the backward processing for Golden DAMP.

Table 6: Golden DAMP Backward Processing Algorithm

```
Function:  [aMP_i, BSF] = BackwardProcessing(T, m, i, BSF, GoldenBatch)
Input:  T:   Time series
        m:   Subsequence length
        i:   Index of current query
        BSF:            Highest discord score so far
        GoldenBatch:    A long time series with all possible normal
        patterns
Output: aMP_i:Discord value at position i
        BSF:       Updated highest discord score so far
```

```
1   aMP_i = inf
2   prefix = min(2^nextpow2(m),length(GoldenBatch))
3   While aMP_i ≥ BSF
4      If the search reaches the beginning of the Golden Batch
5         aMP_i = min(MASS(GoldenBatch_1:end,T_i:i+m-1))
6         If aMP_i > BSF  // Update the current best discord score
7            BSF = aMP_i
8         break
9      Else
10        aMP_i = min(MASS(GoldenBatch_end-prefix+1:end,T_i:i+m-1))
11        If aMP_i < BSF
12           break      // Stop searching
13        Else          // Double the length of prefix
14           prefix =2*prefix
15  return aMP_i, BSF
```

Table 6 illustrates the backward processing algorithm of Golden DAMP. As the backward search is performed on top of Golden Batch, we need to enter *GoldenBatch* into the algorithm. The first two lines of Table 6 are still the initialization phase. Line 1 is the same as in Table 2, initializing the discord score of the current subsequence to positive infinity. In line 2 we define the initial length of the iterative doubling search *prefix*. Here we set it as the lower bound of $2^{\text{nextpow2}(m)}$ and Golden Batch size to prevent

possible array out-of-bounds problem at line 10. Then in the loop in lines 3-14, we perform the iterative doubling search, which starts from the end of Golden Batch and goes backwards. We keep searching in $GoldenBatch_{end-prefix+1:end}$ until we find the nearest neighbor whose distance from the current subsequence is less than *BSF* (line 11) or reach the beginning of Golden Batch (line 4). Specifically, if we find the nearest neighbor within the range *prefix*, we assign the approximate discord score of the current subsequence to $a\text{MP}_i$ and stop the search (lines 10-12); if not, in lines 13 and 14 we double the length of *prefix* and continue the search in $GoldenBatch_{end-prefix+1:end}$. If the search finally reaches the beginning of Golden Batch (line 4), we first calculate the exact discord score of the current subsequence using all the data in *GoldenBatch* (line 5), and then determine whether the current highest discord score *BSF* needs to be updated (line 6). If the discord score of the current subsequence is still greater than *BSF*, it means that the subsequence at position *i* does not have a nearest neighbor similar enough to it in the Golden Batch and it is a discord, at which point we should update the current highest discord score *BSF* in line 7.

This discussion of Golden DAMP is a good place to highlight an interesting and important property of the general DAMP algorithm. Virtually all other TSAD algorithms, including USAD [2], AE [2], *Telemanom* [14], NORMA [4] and LSTM-VAE [27], exist *only* as the implicit equivalent of Golden Batch algorithm. Here the training data given to the algorithm acts as the Golden Batch. This can be a problem if the period of the data changes, and we wish to be invariant of that. For example, a healthy human heartrate can vary between about 40 to 120 beats per minute (bpm). If a batch algorithm is trained on

one heartrate, it may have difficulty generalizing to a different heartrate. In contrast, classic DAMP will be unaffected, because at every time step it is using *all* previously seen data as training data. Thus, so long as the heartrate change is not instantaneous, it can adjust to the new periodicity.

To illustrate this, in Fig. 12 we perform an experiment comparing classic DAMP with *Telemanom* [14], on a dataset that has a changing periodicity.

In a sense, the news is even worse for *Telemanom* than Fig. 12 suggests. The algorithm has a stochastic element. We ran it three times, and this is the *best* of the three runs. In addition, note that this is an offline experiment. However, as we discuss in Section 2.5.3 Comparison to LSTM Deep Learning, all algorithms except DAMP have a period between the time they are given the training data, and the time they are ready to begin monitoring (we call this "linger"). Thus, in a real-time situation, there would be a period of a few tens of seconds, for which *Telemanom* would be undefined.

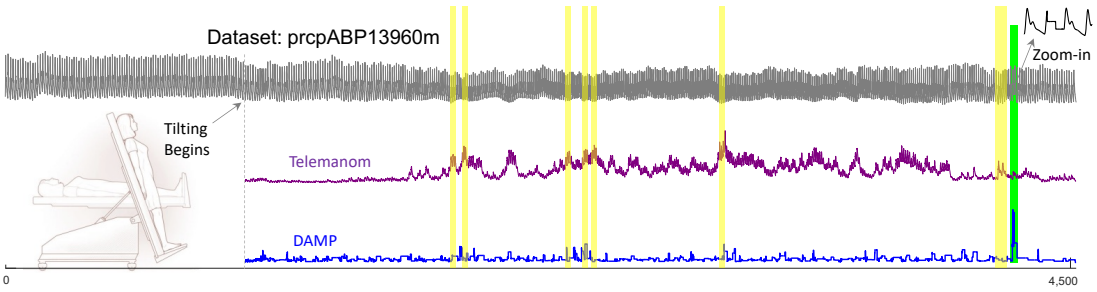

Fig. 12 *top*) A snippet of arterial blood pressure (ABP) data from a healthy patient undergoing a tilt-table test. There are no biological anomalies in the dataset, but near the end there is a short disconnection artifact (highlighted in green). *middle*) If we train *Telemanom* on the prefix of the snippet, before the table was tilted, it has a hard time adjusting to the post-tilt increased heartrate. It flags eight anomalies (highlighted in yellow), all false positives, and fails to discover the single true anomaly. *bottom*) In contrast, because DAMP is using *all* previously seen data, it can adjust to the changing heartrate, and it strongly peaks at the location of the true anomaly. Here $m = 33$.

These observations do open an interesting issue, should we be invariant to changes in periodicity? This is a domain dependent question. Most biological signals can vary innocuously within a certain range. For example, heartbeats, respiration, gait cycles etc. In contrast, cycles guided by the circadian progression of the Earth's rotation, traffic patterns, electrical power demand, web traffic etc., will not be expected to have a change of periodicity, and any apparent change of periodicity probably warrants flagging as an anomaly. The Golden Batch implementation of DAMP allows the user to create a curated dataset that reflects the domain constraints. For example, suppose a user is given normal heartbeats at say 60 bpm. If she wants to be invariant to the heartrate varying between say 50 and 70 bpm, she can just create such rescaled time series and add them to her Golden Batch.

2.4.7 Multidimensional DAMP

The previous sections have shown how to find anomalies in a one-dimensional time series. We believe that in many cases, anomaly detection of *all* the one-dimensional data is sufficient for user demands. For example, in a hospital setting, a doctor may monitor a patient's ECG, blood pressure, and respiration. Most life-threatening situations will show up in at least one of the above. For example, a myocardial infarction, will first show up in the patient's ECG, septicemia will first show up in the patient's blood pressure, and tracheomalacia will first show up in the patient's blood respiration.

However, there are also special cases where anomalies occur in only two or more dimensions. For example, in the low-latitude Pacific West Coast region, typhoons accompanied by heavy precipitation occasionally make landfall in summer. In order to

identify such unusual weather events, it is insufficient to monitor *only* precipitation or wind speed. This is because these areas may have strong winds but sunny weather, or extreme rainfall but still air. As a result, we need to combine wind speed and precipitation as two-dimensional data to find out which day has both precipitation and wind speed anomalies. If such anomalies can be identified in two dimensions, there is a high chance of typhoon weather on that day. Therefore, it is necessary to generalize our DAMP algorithm to support searching in high-dimensional spaces. We refer to the DAMP algorithm for multidimensional data anomaly detection as *multidimensional DAMP*. We note that there are several ways in which the information from multiple time series can be combined. This issue is perhaps worthy of a detailed investigation. Here we show one simple and obvious method and demonstrate that DAMP can easily support it.

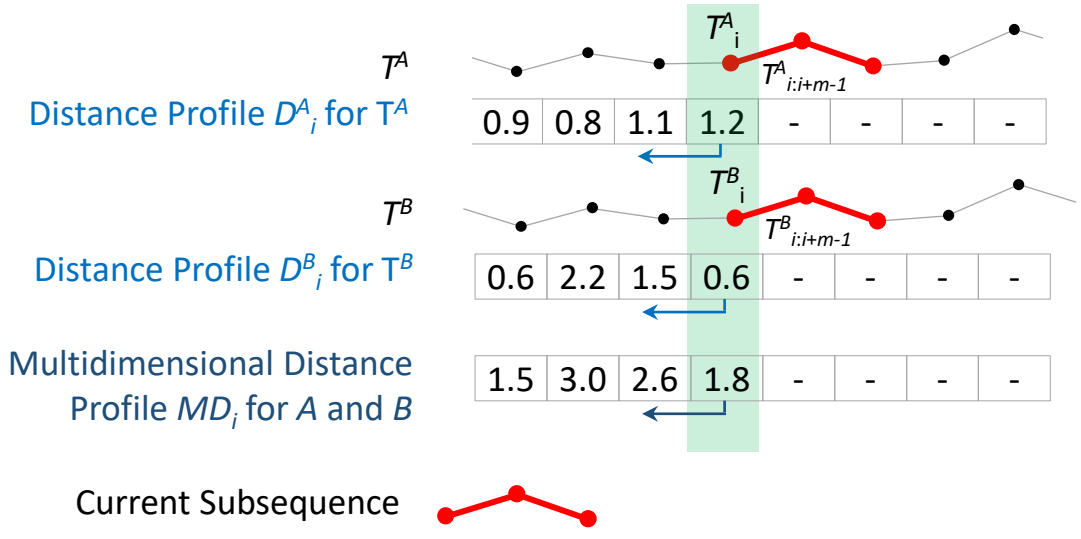

Fig. 13 Multidimensional distance profile for position *i*.

The basic idea of multidimensional DAMP is the same as the one-dimensional DAMP we introduced in Section 2.4.1 Intuitive Overview of DAMP, which retains the procedure of

backward iterative doubling and forward pruning. The difference between them is reflected solely in the calculation of the discord score.

Fig. 13 illustrates how the multidimensional DAMP calculates the discord score for position i. Let T^A be the time series of dimension A in a two-dimensional time series, while T^B corresponds to dimension B, and the length and frequency of T^A and T^B are equal. For position i, we first compute the distances between the current subsequence of T^A and T^B and the subsequences before position i in their respective dimensions, forming two distance vectors $D^A{}_i$ and $D^B{}_i$ (see Definition 4). After that, we add the elements of the two distance vectors two by two according to their positions to produce a new vector MD_i, which contains the distance information in both dimensions A and B. Finally, the minimum value on MD_i is the discord score at position i. As the algorithm progresses, the *BSF* continuously tracks the current highest discord score that combines information from both dimensions. Table 7 presents the multidimensional backward processing algorithm. As it is primarily similar to Table 2, we refer the reader to Section 2.4.4 Formal Pseudocode for DAMP for more details on the iterative doubling backward algorithm. Here we only highlight the parts that have changed. Compared to Table 2, we add two new inputs *TA* and *TB*, the time series in dimensions A and B. In lines 5 and 10, we change the calculation of the discord score at position i *aMPi*. In line 5, to obtain *aMPi*, we call MASS twice to calculate the distance between the current subsequence of *TA* and *TB* and all subsequences before position i respectively. Next, we add the elements in the two distance vectors returned by MASS two by two according to the positions to obtain the multidimensional distance profile. Finally, the minimum value of the

multidimensional distance vector is taken as the exact discord score of position *i*. Line 10 is similar to line 5. The only difference is that line 10 only finds the nearest neighbor in the prefixes of *TA* and *TB* before position *i* and *aMPi* is the approximate discord score for position *i*.

give the implementation details of multidimensional DAMP. Here we only demonstrate the two-dimensional version, however the reader can easily modify it to work with higher dimensional data. Since the basic steps of multidimensional DAMP and one-dimensional DAMP are the same, the framework of multidimensional DAMP is identical to Table 1.

Table 7 presents the multidimensional backward processing algorithm. As it is primarily similar to Table 2, we refer the reader to Section 2.4.4 Formal Pseudocode for DAMP for more details on the iterative doubling backward algorithm. Here we only highlight the parts that have changed. Compared to Table 2, we add two new inputs T^A and T^B, the time series in dimensions *A* and *B*. In lines 5 and 10, we change the calculation of the discord score at position *i* $a\text{MP}_i$. In line 5, to obtain $a\text{MP}_i$, we call MASS twice to calculate the distance between the current subsequence of T^A and T^B and all subsequences before position *i* respectively. Next, we add the elements in the two distance vectors returned by MASS two by two according to the positions to obtain the multidimensional distance profile. Finally, the minimum value of the multidimensional distance vector is taken as the exact discord score of position *i*. Line 10 is similar to line 5. The only difference is that line 10 only finds the nearest neighbor in the prefixes of T^A and T^B before position *i* and $a\text{MP}_i$ is the approximate discord score for position *i*.

Table 7: Multidimensional DAMP Backward Processing Algorithm

```
Function:  [aMP_i, BSF] = BackwardProcessing(T^A, T^B, m, i, BSF)
Input:  T^A:  Dimension A of the multidimensional time series
        T^B:  Dimension B of the multidimensional time series
        m:    Subsequence length
        i:    Index of current query
        BSF:        Highest discord score so far
Output: aMP_i:Discord value at position i
        BSF:        Updated highest discord score so far
```

```
1  aMP_i = inf
2  prefix = 2^nextpow2(m)  // Initial length of prefix
3  While aMP_i ≥ BSF
4     If the search reaches the beginning of the time series
5        aMP_i = min(MASS(T^A_1:i,T^A_i:i+m-1) + MASS(T^B_1:i,T^B_i:i+m-1))
6        If aMP_i > BSF  // Update the current best discord score
7            BSF = aMP_i
8        break
9     Else
10       aMP_i = min(MASS(T^A_i-prefix+1:i,T^A_i:i+m-1) + MASS(T^B_i-prefix+1:i,T^B_i:i+m-1))
11       If aMP_i < BSF
12           break      // Stop searching
13       Else           // Double the length of prefix
14           prefix =2*prefix
15 return aMP_i, BSF
16
```

Multidimensional DAMP also has a similar forward pruning process to that of one-dimensional DAMP, as shown in Table 8. Compared with Table 3, we need to only change line 5. In the range of *lookahead*, the distances between the current and future subsequences of T^A and T^B are calculated separately. Then the distance vectors of A and B dimensions are summed to yield a distance vector MD'_i containing two-dimensional information. Our pruning decisions are made based on this two-dimensional distance vector.

Table 8: Multidimensional DAMP Forward Processing Algorithm

```
Function:  PV = ForwardProcessing(T^A, T^B, m, i, BSF, PV)
Input:  T^A:  Dimension A of the multidimensional time series
        T^B:  Dimension B of the multidimensional time series
        m:    Subsequence length
        i:    Index of current query
        BSF:        Highest discord score so far
        PV:   Pruned Vector
Output: PV:   Updated Pruned Vector
```

```
1   lookahead = 2^nextpow2(m)        // Length to peek ahead
2   If the search does not reach the end of the time series
3     start = i + m
4     end = min(start + lookahead - 1,length(T))
5     MD'_i = MASS(T^A_start:end,T^A_i:i+m-1) + MASS(T^B_start:end,T^B_i:i+m-1)
6     indices = all indices in MD'_i with values less than BSF
7     indices = indices + start - 1      // Convert indices on distance
8     profile to indices on time series
9   PV_indices = 0                     // Update the Pruned Vector
10  return PV
```

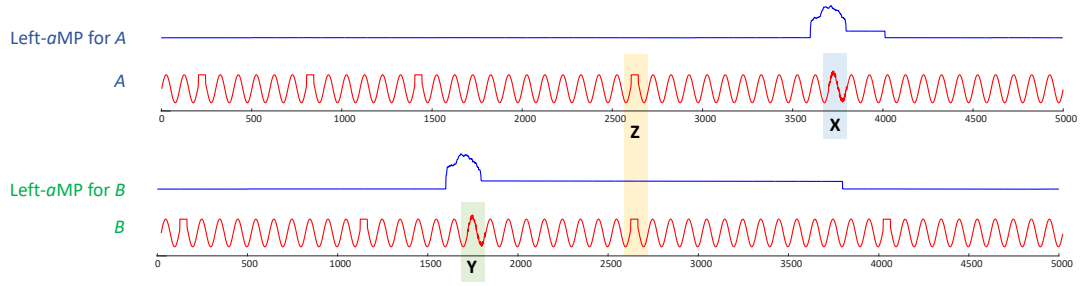

Fig. 14 Synthetic time series *A* and *B*. *top*) Synthetic dataset *A* and its corresponding one-dimensional Left-*a*MP. *bottom*) Synthetic dataset *B* and its corresponding one-dimensional Left-*a*MP. Here we set the window size to be the minimum positive period of the sine wave, i.e., $m = 100$.

Let us start with a toy data set to understand the difference between multidimensional DAMP and one-dimensional DAMP. The red curves in Fig. 14 illustrate two synthetic time series *A* and *B*. These two time series consist mainly of sine waves. Specifically, for time series *A*, the data at positions 3700-3799 (**X**) are noisier than the other parts, while for time series *B*, the data at positions 1700-1799 (**Y**) are noisier. If you look closely, you will find that the two time series will have a square wave at random positions from time

to time. It so happens that at positions 2605-2644, both time series show a square wave simultaneously, which is where our real anomaly lies. We denote it as **Z**. We tested the time series *A* and *B* with one-dimensional DAMP and two-dimensional DAMP respectively to see if they could find the true anomaly **Z**. Fig. 14 also gives the results of performing a one-dimensional DAMP on time series *A* and *B*. It is easy to see by the highest point of the blue curve in Fig. 14.*top* that one-dimensional DAMP is attracted to the noisy sine wave in *A* and does not notice the anomaly at position **Z**. Similarly, as illustrated in Fig. 14.*bottom*, one-dimensional DAMP on *B* also fails to detect the anomaly at **Z**, instead considers the nosier **Y** as the anomaly. Missing information in another dimension, the one-dimensional DAMPs mistakenly believe that the presence of the square wave at **Z** is justified because they observe similar patterns before **Z**.

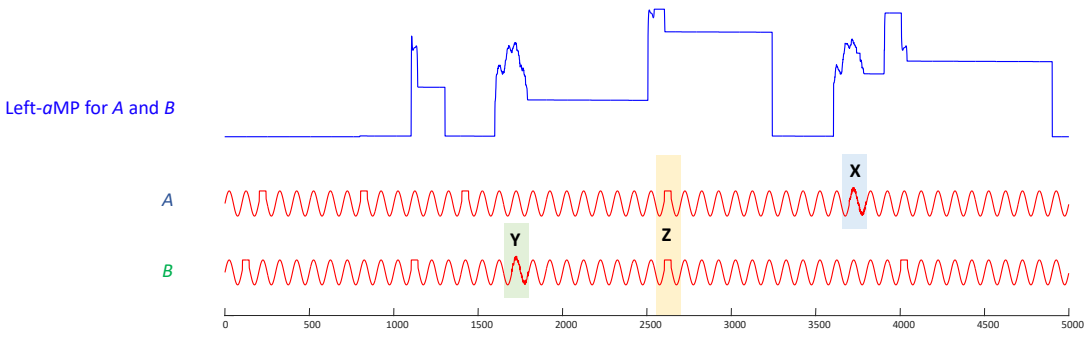

Fig. 15 Left-*a*MP generated by two-dimensional DAMP. Here we set the window size to be the minimum positive period of the sine wave, i.e., $m = 100$.

Next, we combine *A* and *B* into a two-dimensional time series and feed it into the two-dimensional DAMP to see if the results will be different. The Left-*a*MP generated by two-dimensional DAMP is shown in Fig. 15. Note that compared with the Left-*a*MP generated by the one-dimensional DAMP in Fig. 15, the two-dimensional Left-*a*MP

captures more anomalies with more “bumps” on its curve. All these bumps can be interpreted intuitively. For example, when both square and sine waves are present, or when one of the sine waves is noisier, they are recognized by the algorithm as a potential anomaly and correspond to a bump in the Left-*a*MP. What is more, the position of the highest point of Left-*a*MP in Fig. 15 corresponds to the coincidence of two square waves, that is, **Z**. This is because if you look at the entire time series of *A* and *B*, you will see that the square wave only appears at **Z** in both dimensions simultaneously, which cannot be observed at other locations.

We have seen that we can create a synthetic dataset that has an anomaly that can be discovered only by considering two time series simultaneously. However, can we discover two-dimensional anomalies in real data? Surprisingly, we are not aware of any such benchmark dataset. Most datasets in the space are synthetic, or are multidimensional, but have anomalies that are so obvious that it suffices to examine any *single* dimension [2][14]. However, we can explore energy grid data published by a consortium of Texas A&M and USC in 2021 [38], and use out-of-band data to evaluate the returned anomalies. Fig. 16.*top* shows three years of wind speed and relative humidity data from the New York area between 2018 to 2020 [38].

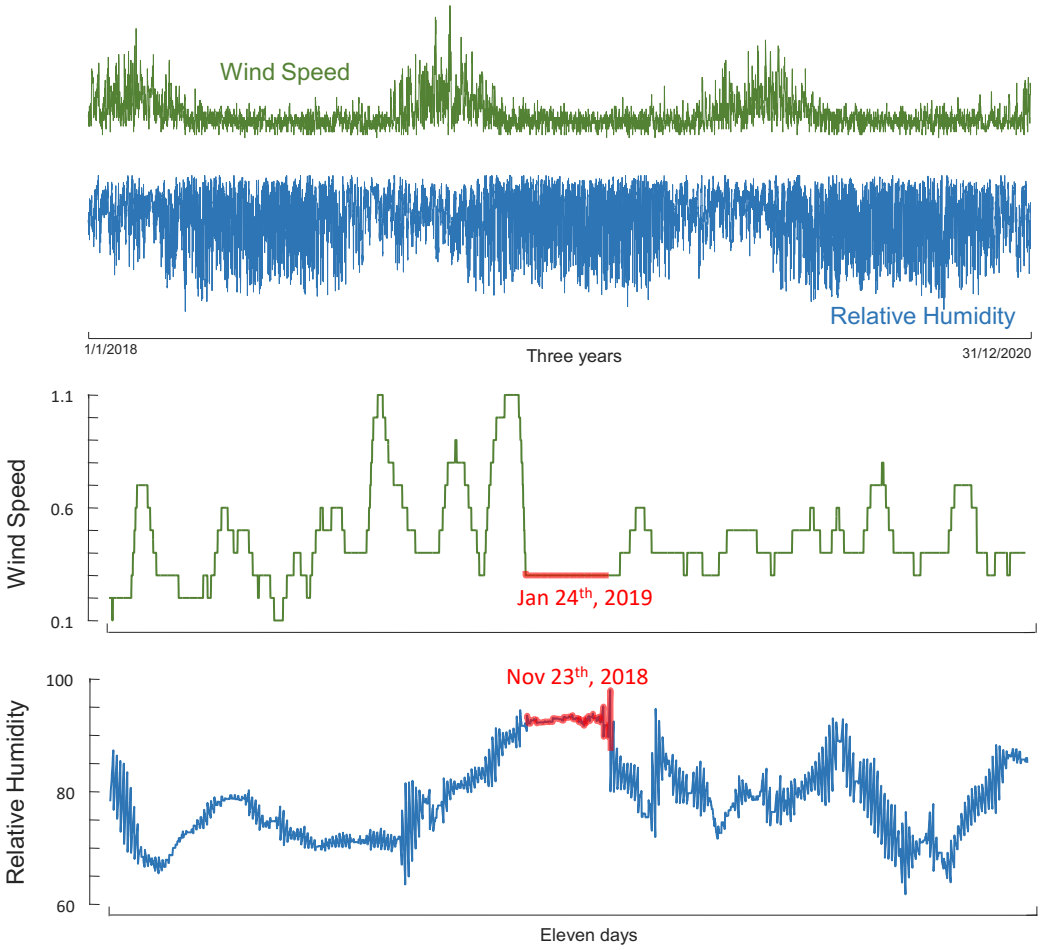

Fig. 16 *top*) Three years of wind speed and relative humidity data for the New York area from [38]. *bottom*) The two corresponding top 2D discord in this dataset. Here m = 1440 (one day in minutes).

Fig. 16.*bottom* shows the results of our search on the one-dimensional data of wind speed and relative humidity, respectively, and the anomalies identified by one-dimensional DAMP are marked in red. First, for wind speed, the one-dimensional DAMP reports the constant interval occurring on January 24, 2019, as an anomaly; however, we do not find any reported climate anomaly in New York State on that date. That is to say, although the algorithm finds an anomaly with a pattern that is different from its context, it does not seem to noticeably affect people's lives. As a result, we can conclude that the wind speed

anomaly is trivial. Second, for relative humidity, the one-dimensional DAMP identifies the continuous peak occurring on November 23, 2018, as an anomaly. Through a Google search, we found reports of heavy rainfall and flooding that occurred in New York State on that day [22], which confirms that the anomalies identified in the dimension of relative humidity are informative and that the one-dimensional DAMP is effective.

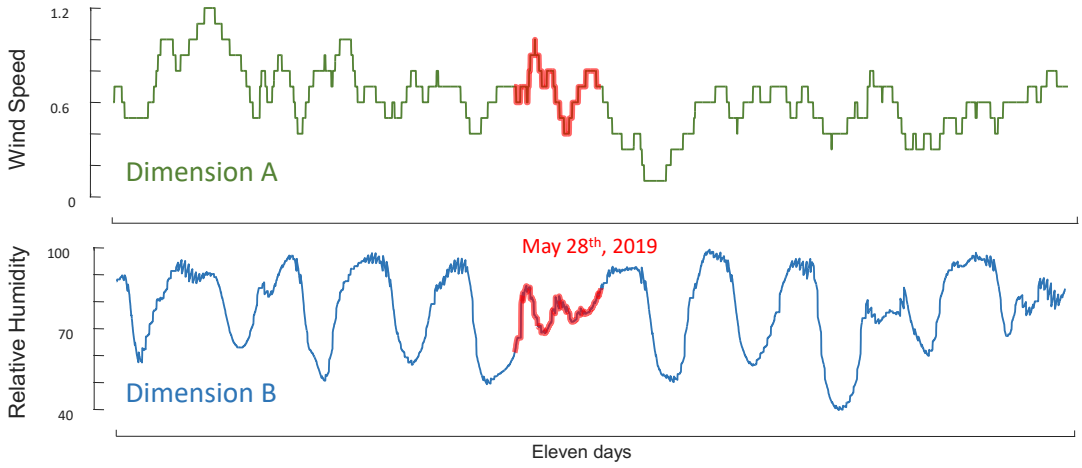

Fig. 17 Top discord for two-dimensional DAMP. Here m = 1440 (one day in minutes).

However, if we combine wind speed and humidity and search in two dimensions, will the algorithm give us more interesting results? To investigate this, we took wind speed as dimension A and relative humidity as dimension B and re-executed this two-dimensional data using multidimensional DAMP. The results are presented in Fig. 17. Note that the two-dimensional DAMP reports a different date to either of the one-dimensional DAMP runs, May 28, 2019. This means that *both* humidity and wind speed in New York City showed anomalous patterns on this date. This anomaly is confirmed by the news "*A powerful thunderstorm slammed Staten Island Tuesday night, pounding the borough with large hail, heavy rain and the threat of a tornado.*" [29].

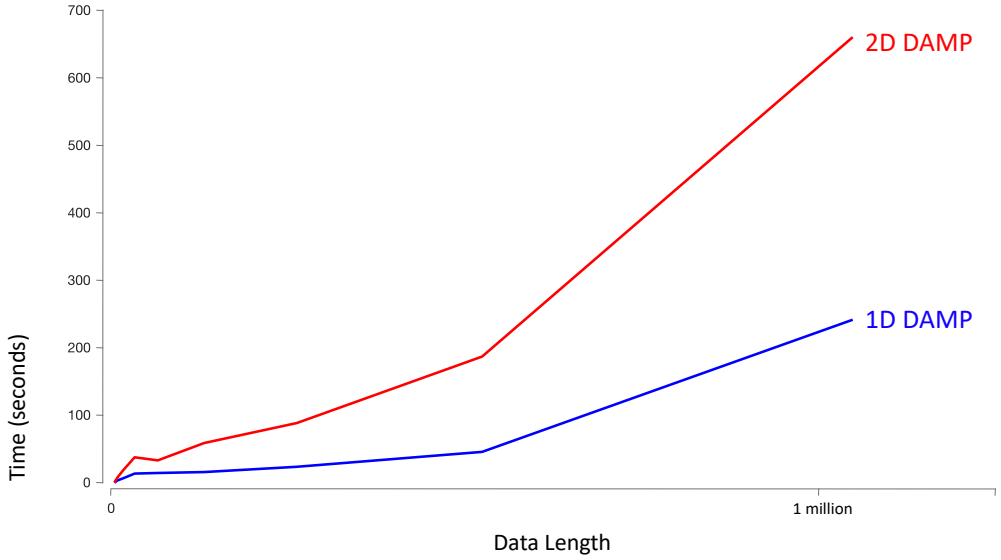

Fig. 18 The scalability of 1D and 2D DAMP over increasingly large datasets. The cost to double the number of dimensions considered is only slightly worse than double the time, suggesting that multidimensional DAMP search inherits the efficiency of the 1D version. Here m = 1440 (one day in minutes).

We have demonstrated the utility of multidimensional DAMP. However, readers may wonder if it will pay a large time overhead for it. To investigate this, we used the data shown in Fig. 16.*bottom* (wind speed) and Fig. 17 and recorded the time cost of the one-dimensional and two-dimensional algorithms for increasingly long subsets. The experimental results are shown in Fig. 18. It can be seen that the time cost of a two-dimensional DAMP is only a small constant ratio of approximately 3.0 slower than the cost of a one-dimensional DAMP, which suggests the good scalability for multidimensional DAMP.

2.5 Empirical Evaluation

To ensure the reproducibility of our experiments, we have built a website [10] containing all the data/code used in this chapter. All experiments were conducted on an Intel® Core i7-9700CPU at 3.00GHz with 32 GB of main memory, unless otherwise stated.

There are two things one normally needs to establish to validate an anomaly detection algorithm.

- **Effectiveness**: Here we feel less of an obligation. As we noted in Section 2.1, there are at least one hundred independent papers that have used discords to solve a real-world problem and that have shown that discords are the only technique that seems to be able to discover anomalies that are not visually obvious (Fig. 3, Fig. 4 and Fig. 5). Nevertheless, for completeness we will show examples in Sections 2.5.1 Energy Grid Dataset and 2.5.2 Machining Dataset that further demonstrate the excellent effectiveness of discords in diverse domains, and Section 2.5.3 Comparison to LSTM Deep Learning and Section 2.5.4 Comparison on the KDD Cup 2021 datasets offer comparisons to several deep learning-based methods.
- **Efficiency**: As this is the main contribution of the paper, here we will attempt an ambitious set of anomaly detection experiments in terms of both throughput and scale.

2.5.1 Energy Grid Dataset

Recently, a consortium from Texas A&M and USC released a large dataset on decarbonized energy grids [38]. The dataset contains files representing three years of measurements of various metrics in sixty-six electrical zones in the continental USA. As

Fig. 19 suggests, each file represents eleven measurements, ten of which are *measured* (temperature, wind speed etc.), but one is *computed* from the first principles of astronomy, the Solar Zenith Angle.

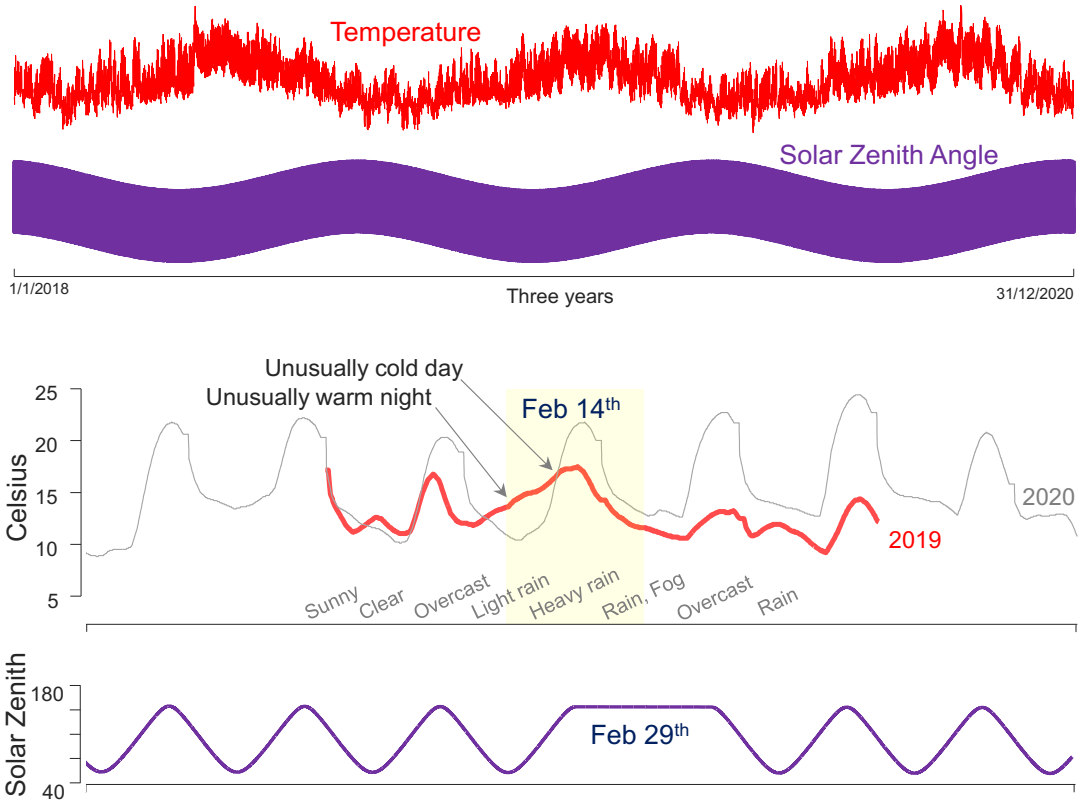

Fig. 19 *top*) Two examples of time series from [38]. Most, like temperature are *measured*, but Solar Zenith Angle is *computed*. *bottom*) The two corresponding top discords in these datasets. Here m = 5760 (four days in minutes).

The total size of this dataset is 12 GB, representing 2,174 years of data with 1,142,668,098 datapoints. As such, we believe that it is the largest real dataset ever searched for anomalies. This complete search took only 2.06 days.

As Fig. 19 shows, most of the anomalies discovered do have a semantic meaning that can be traced. For example, a temperature trace from California had a discord that reflected "*Valentine's Day Storm Slams California*" [33]. Even the *computed* time series reveals a

strange anomaly echoing a biblical event. Joshua persuades God to stop the sun from moving for a day "*There has never been a day like it before or since* (Joshua 10:14)". In our dataset there is a similarly unique day in which the sun apparently does not move! The reader will readily appreciate the cause of this anomaly, after noting it occurs on the 29th of February [34]. It is a classic leap year bug. Note that we informed the Texas A&M and USC team of this bug, so presumably it will be fixed in upcoming releases.

2.5.2 Machining Dataset

The example in Section 2.5.1 Energy Grid Dataset demonstrates the utility of anomaly detection in batch data exploration. However, in some cases if we can do anomaly detection in real-time, we may be able to perform an intervention to improve an outcome. For example, consider the process of making parts using a CNC milling machine. Occasionally a problem arises where an item being machined is not held correctly and it moves. This can cause a milling machine to "crash" [8]. High-end CNC mills can cost over one million dollars, and crashes resulting in more than $20,000 in damage are known. Many (but not all) machining processes can be paused by an operator, so in principle it may be possible to stop a machine before it crashes. However, with the speed at which these machines operate, it is unlikely that the operators' reflexes would be fast enough.

This suggests the question, could we monitor the process with telemetry, and pause the process if we detected an anomaly? In order to test this, we recreated a common scenario in Fig. 20.

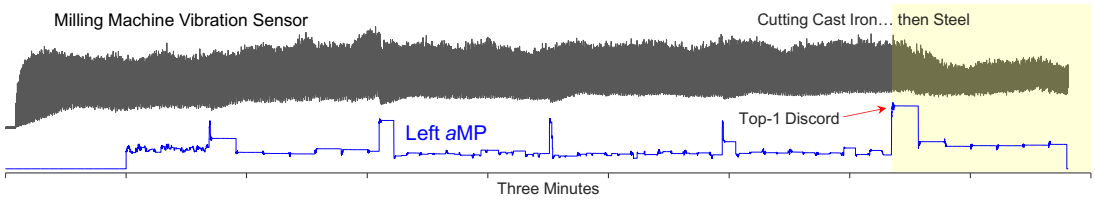

Fig. 20 *top*) Vibration telemetry from a milling machine that was cutting cast iron, but then overshot to start cutting the steel jaws of the vice. *bottom*) The Left-*a*MP discovers the transition. Here $m = 16$.

A common CNC programming error is to give the wrong coordinates for a cutting pass, and have the cutter overshoot the intended material to be machined, and inadvertently attempt to remove material from the jaws of the vice. Because the jaws are typically harder than the material they hold, and more resistant to cutting, two things can happen:

- The milling cutter itself will break. This is a $20 to $200 error.
- A much worse possibility is that the cutter will move the vice. If it happens to push it into the path of later traversal, this could cause a head crash, which is a $2,000 to $20,000 error.

As Fig. 20 shows, the *a*MP can detect the change of material, and this could be used to sound an alarm, or pause the machining process until the operator can inspect this.

Note that before the true anomaly there are other areas with high discord scores. They are when the milling cutter changes direction (from *Climb* milling to *Conventional* milling). Under our proposed scheme these would have a small cost, the process would pause until the operator visually confirms all is well, and hits *continue*.

2.5.3 Comparison to LSTM Deep Learning

Although dozens of competing deep learning anomaly detection (DLAD) algorithms now exist, it is impossible to say which is the state-of-the-art. This is because, as Wu and Keogh have demonstrated, the amount of mislabeling in the benchmark datasets dwarfs the reported differences between algorithms [35]. It makes no sense to say that algorithm **A** is 5% better than algorithm **B**, when up to 30% of the ground truth labels are suspect.

To bypass this issue, here we will compare to just *Telemanom*. It is the most cited anomaly detection paper of the last five years [14], and several independent papers have also found it to be effective. The general idea of this work is to use LSTM to predict future values, then detect anomalies based on the difference between predictions and actual data. Can *Telemanom* detect the anomalies we consider in this work?

- **ECG** (Fig. 4) **No**. Given the same 500 datapoint prefix as training data, it fails to find the anomaly. If we give it ten times as much training data (the first 5,000 datapoints), it *still* fails.
- **Bearing** (Fig. 3): **Yes**. However, *Telemanom* took a total of (517.6 training + 700.4 testing) 1,218 seconds. This is two orders of magnitude slower than DAMP, which took 16.1 seconds. More importantly, *Telemanom* is an order of magnitude slower than real-time, precluding any possibility of online monitoring.
- **Energy Grid** (Section 2.5.1 Energy Grid Dataset) **Maybe**. There are only *objective* labels for Solar Zenith Angle (this anomaly was discovered with DAMP but *confirmed* with the data creators). If *Telemanom* sees only the first week as training data (as

DAMP did), then it only learns that the Solar Zenith Angle can decrease over time, and it will flag as anomalous anything that happens after the summer solstice. A solution to this problem is to allow *Telemanom* to train on the full first year, then test on the subsequent years. Then it *may* find the "Joshua" anomaly. However, this will take 59.1 hours, over 1,300 times slower than DAMP.

- **Milling Data** (Fig. 20) **No**. Actually, *Telemanom* can detect the same anomaly as DAMP. But recall it can only start training when the first 5,000 datapoints arrive, and it takes 411 seconds to train the model. However, 127 seconds after it begins training, we encounter the anomaly, and about 21 seconds after that, the endmill snaps off. *Telemanom* is just too slow to be useful here.

These comparisons suggest that the most cited deep learning anomaly detection algorithm is not as accurate as DAMP, requires more training data, and is much slower.

2.5.4 Comparison on the KDD Cup 2021 datasets

To further see the limitations of deep learning time series anomaly detection, we can compare DAMP to DLAD algorithms on publicly available benchmarks. Wu and Keogh have shown that most benchmarks in this space are too trivial to be interesting, and in any case are plagued by mislabeling and other problems [35]. Instead, we consider the KDD Cup 2021 dataset consisting of 250 univariate time series [11]. This archive was designed to be diverse, have a spectrum of difficulties ranging from easy to essentially impossible, and has a detailed provenance for each of the 250 datasets, giving us some confidence that the ground truth is correct. Moreover, the datasets include a wide range of domains,

including cardiology, industry, medicine, zoology, weather, human behavior, etc. We use the accuracy metric that was suggested by the dataset's creators. In brief, each of the 250 datasets has a single anomaly. Each algorithm is tasked with predicting the location of that anomaly. Let the length of the anomaly be *L*. If the prediction is within plus or minus *L* data points of the anomaly's true location, it is judged correct. If *L* is less than 100, then it will be set to 100. The scores in Table 9 show the ratio of correct predictions for the 250 datasets.

Table 9: Accuracy and Time for Eight TSAD Methods

Method	Accuracy	Train and Test Time
USAD [2]	0.276	8.05 hours
LSTM-VAE [27]	0.198	23.6 hours
AE [2]	0.236	6.11 hours
Telemanom [14]	*Out of memory error on longer examples*	
NORMA [4]	0.474	17.8 minutes
SCRIMP (Full-MP)	0.416	24.5 minutes
DAMP (Left-MP) out-of-the-box	0.512	4.26 hours
DAMP (Left-MP) sharpened data	0.632	4.26 hours

Once again, these results show that DAMP is more accurate and faster than deep learning-based methods. It is important to note that the results for DAMP are completely free of *any* human intervention or tuning. We use four hardcoded lines of Matlab (see [10]) to find the approximate period in each training dataset, and used that as the value of *m*. Likewise, we simply hardcoded a *single lookahead* value for all 250 datasets. Further optimizing the former would improve accuracy and personalizing the latter for each individual problem would improve the speed. However, we wanted to show that even the most naïve out-of-the-box use of DAMP is highly competitive. As an example of a small

intervention that can further improve accuracy, if we run DAMP on *sharpened data* (a single extra line of code, see [10] for details) the accuracy improves to 0.632.

The left-discords of DAMP are significantly more accurate than the full-discords computed by SCRIMP, because some anomalies have near "twin-freaks" that suppress the distance of the anomaly to its nearest neighbor. Note that the time for SCRIMP and NORMA here is relatively good, as there are 250 *short* time series. In Fig. 23 we will see that for longer time series this advantage of SCRIMP/NORMA rapidly inverts.

We included a comparison to the recently published NORMA [4], which can be seen as a sort of Matrix Profile that uses an automatically discovered subset of the training data as the reference data. Here we used the original authors' tools and suggestions to set the parameters (we were able to make the results *slightly* better with our own parameter settings [10]). The time for NORMA is good, but it is important to note the following:

These datasets have tiny training data splits (they were deliberately made that way, to allow the deep learning community to consider them in a tenable fashion [11]). But as Fig. 24 shows the NORMA algorithm scales poorly for large datasets.

On these datasets, we can easily close all of the time gap by using either X-Lag-Amnesic DAMP (Section 2.4.6.1 X-Lag-Amnesic DAMP) or Golden DAMP (Section 2.4.6.2 Golden DAMP), with only a minimal decrease in accuracy. Indeed, the Golden DAMP algorithm essentially subsumes NORMA as a special case.

The results in Table 9 mask a unique timing advantage that DAMP has over not only NORMA, but all other non-trivial anomaly detectors5. We believe that DAMP is the only instantaneous TSAD in the literature. To see this, consider the situation in Fig. 21.

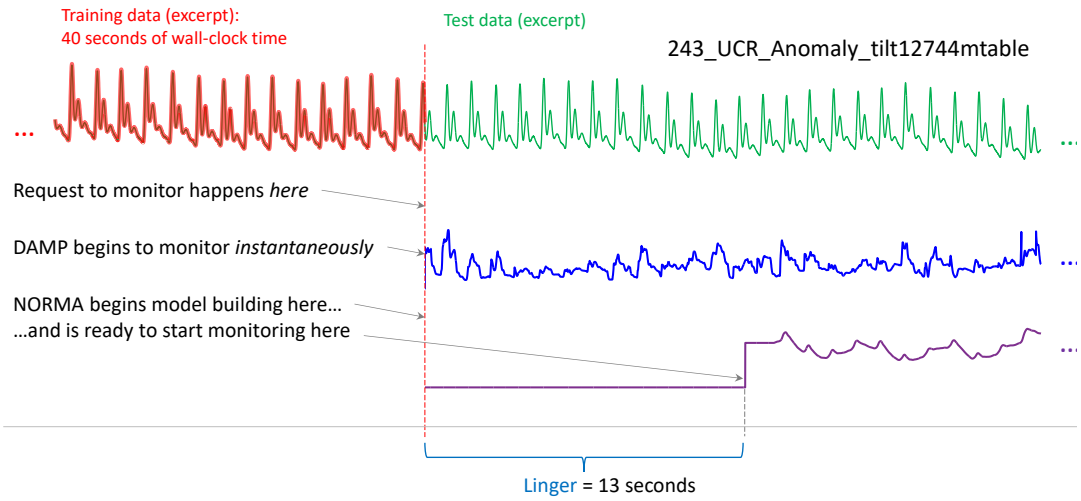

Fig. 21 An excerpt from the 243_UCR_Anomaly_tilt12744mtable dataset. The task is to exploit information in the training split, to detect the most significant anomaly in the test split. When requested, DAMP can instantaneously begin to monitor. However, NORMA (and all other TSAD algorithm), must have a period of inaction or "linger" while they build their models. Here $m = 276$.

The figure shows a dataset from the KDD Cup 2021. The first forty seconds of wall-clock time pass, and then we are invited to monitor for anomalies in the remainder of the data. We define "linger" as the time a TSAD algorithm requires to ingest the training data, build its model, and be ready to start monitoring. As shown in Fig. 21, the linger for NORMA on this problem is thirteen seconds. This means that any anomaly that occurs in the first thirteen seconds will not be detected (or will only be detected post-mortem).

[5] Here we explain "non-trivial anomaly detector". Simple rule-based conditionals such as: "**`if`** `the time series ever reports a value that is higher than any value you have seen before,` **`then`** `flag anomaly`" could be used as an anomaly detector, and could be instantaneously instantiated. By *non-trivial* we mean any TSAD algorithm that examines each subsequence for any information about shape, autocorrelation, Markov properties etc., and compares this information (in the most general sense), to a model gleaned from training data. The reader will appreciate that this includes essentially all proposed anomaly detectors in the literature.

Note that DAMP appears to be unique among TSAD algorithms in having zero linger. In this example, the linger of NORMA may not be too consequential (although it grows rapidly with more training data, see Fig. 24). Perhaps the attending physician can wait with the patient while the model is being built. However, recall our machining example in Section 2.5.2 Machining Dataset. Here, if the linger is more than 127 seconds, the TSAD algorithm would not be able to avoid the expensive head-crash.

Recall that Table 9 notes "*Out of memory error on longer examples*" for *Telemanom* [8]. There does not seem to be any simple way to fix this issue, so we did the following. We sorted all the datasets from smallest to largest, and kept evaluating increasingly longer datasets until the first failure. *Telemanom* failed at the 63rd smallest dataset (114_UCR_Anomaly_CIMIS44AirTemperature2). On the first 62 datasets it correctly found the anomaly on 29, giving an accuracy of 0.468. This took *Telemanom* 3.4 hours. When we run DAMP on just these 62 shorter datasets, it takes 64.9 seconds. In general, the 62 shorter test cases are the easier ones (they certainly have a much higher default rate), yet both flavors of DAMP are still significantly more accurate.

Finally note that Table 9 does not include any comparisons to the algorithms that entered the KDD Cup in 2021 [11]. The best performing algorithms scored an impressive 88.4%. However, note that none of the top performers have made code publicly available. Moreover, all the top performers use *meta-algorithms*. For example, the top place algorithm, DeepBlueAI, used a meta-algorithm that included at least four different algorithms ("Fourier Transformation based methods", Matrix Profile, LightGBM and Dilated CNN). In all cases, the logic used to switch between or combine the atomic

algorithms is not clear (We hope that in at least some cases, the participants with publishing a publication will make that clear). In contrast, Table 9 compares the leading *single* algorithms, which have usable public implementations. Combining them in a meta-algorithm or ensemble would be an interesting project but is beyond the scope of this chapter.

2.5.5 Threshold Learning for DAMP

Up to this point, we have experimentally demonstrated that DAMP can *locate* the most anomalous subsequence. However, we have not shown how the algorithm makes a binary decision thereafter to flag the subsequence as anomalous or not. For this purpose, we simply need to learn a *threshold*. To demonstrate, consider the following experiment. We created 200 random walk time series of length one million. As shown in Fig. 22.*top*, into half of them we randomly inserted a subtle anomaly, a low amplitude random section of length 950 (Why length 950? We found that if we used length 1,000 we got perfect accuracy, which is uninteresting for this experiment. So, we tuned the value to give an error rate of about 10%). In Fig. 22.*left*, we show the top-1 discord score (for m = 1,024) for all 200 time series, divided into the two cases. This plot suggests that a threshold of 36.0 is the optimal value to maximize the accuracy on future occurrences. To test this, we created and tested an additional million examples, all of which are also of length one million, classifying an actual anomaly as a true positive if the correct location of the anomaly was discovered *and* the top-1 discord score was above the threshold. Fig. 22.*right* shows the confusion matrix.

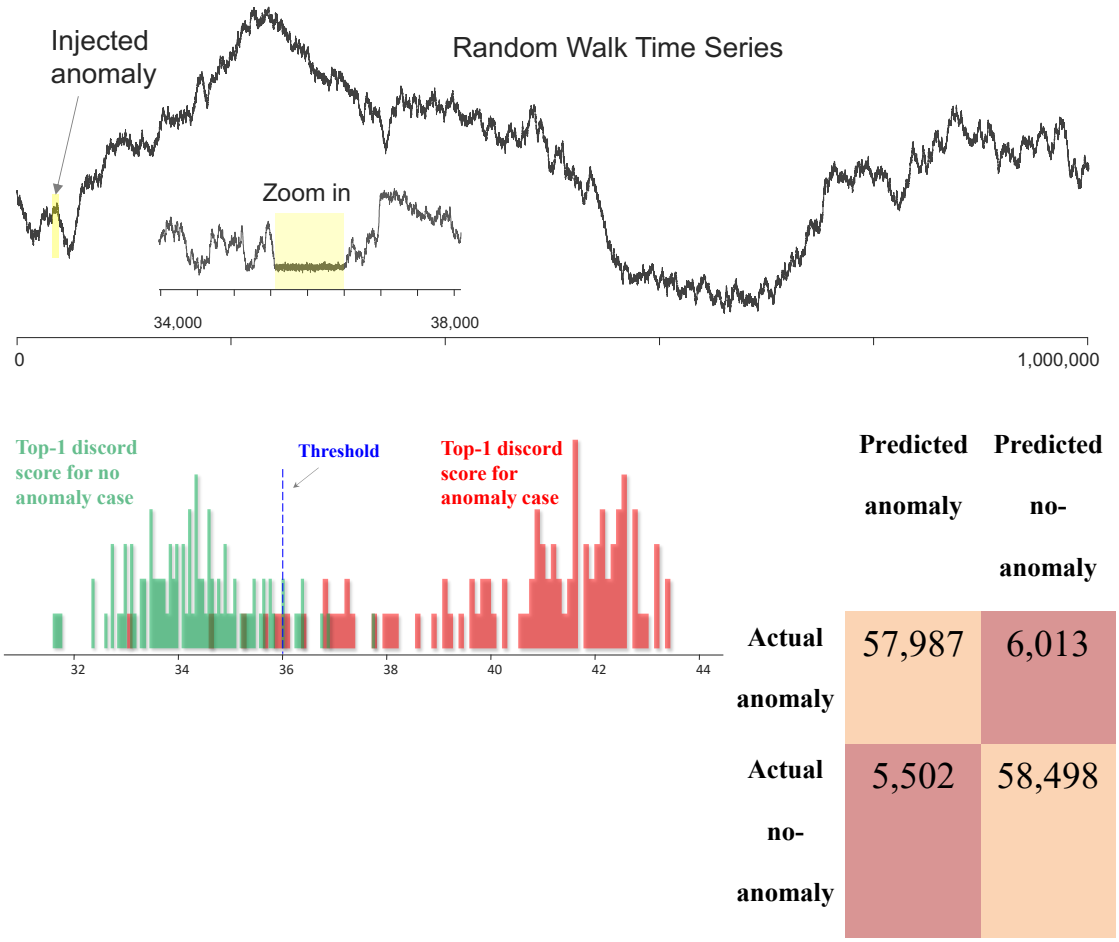

Fig. 22 *top*) A sample random walk with an anomaly embedded. *left*) The distribution of top-1 discord scores for the two cases of interest. *right*) The confusion matrix for this task. Here $m = 1024$.

We note in passing that this experiment (which took several days distributed across commodity laptops and desktops), trained on time series with a total length of 200 million, and tested on time series with a total length of 128 billion. To the best of our knowledge, this is the largest scale time series anomaly detection experiment ever conducted. Could deep learning do this? We estimate that *Telemanom* [14] would take about twelve years to do this, although in practice it gives *out-of-memory* errors.

2.5.6 Scalability Comparisons

To find out which elements of our proposed method contribute most to its efficiency, we have performed an ablation study, in which various elements of DAMP were progressively crippled. As a baseline, we also compare to SCRIMP [39]. This comparison to SCRIMP is a little unfair, as it discovers motifs as well as discords. However, it seems to be the most used discord discovery algorithm in recent years. Fig. 23 summarizes our findings.

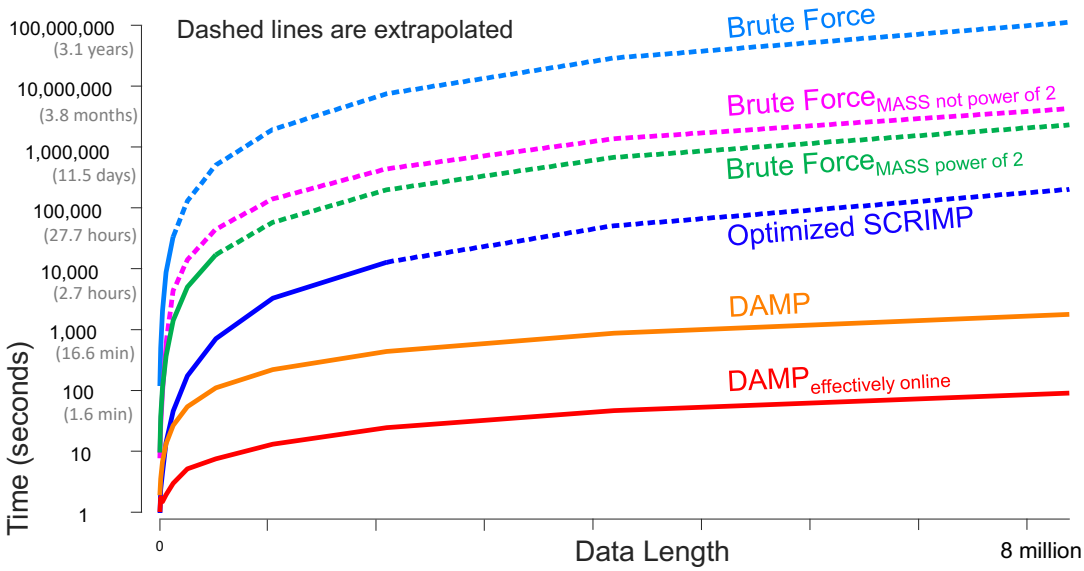

Fig. 23 The CPU time vs time series length for various discord discovery algorithms. Note the Y-axis is in log scale. Note that $\text{DAMP}_{\text{effectively online}}$ means that the forward processing algorithm introduced in Table 3 was used. Here $m = 94$.

It is clear that each element we proposed does actually contribute to speed up, and that DAMP is effectively linear in n.

As we earlier noted, most of the benchmark datasets are only hundreds to thousands of datapoints long [35], and that seems to have set the limit of the ambition of most of the

community when it comes to scalability. However, a recent paper pushed that envelope by considering a two million length ECG dataset [4]. In fact, these authors graciously gave us the *exact* dataset they used, (which was in fact even longer than they considered in [4]), and helped us create a perfectly commensurate experiment, as shown in Fig. 24. A real-time video trace of this experiment is at [10].

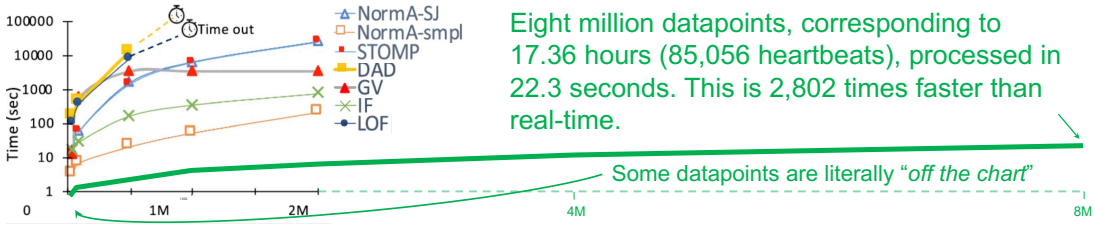

Fig. 24 (Most of this figure is taken from [4] with permission, only the green elements are new). The scalability of various algorithms on increasing large subsets of a long ECG trace. All algorithms except DAMP are limited to the first 2M data points by [4]. Note that the Y-axis is logarithmic. Here $m = 94$.

Note that of the many approaches considered, some time out (i.e., are not finished in a four-hour cutoff) at length 500K. In contrast, DAMP can handle eight million datapoints in just 22.3 seconds, this is over 358,000 Hz. In fact, DAMP is so fast, that the time it reports for the 50K length trial is literally off the original chart, taking less than one second.

As eight million datapoints are about the longest publicly available ECG, in Fig. 25 we conclude this section by searching a single random walk time series of length 2^{30}.

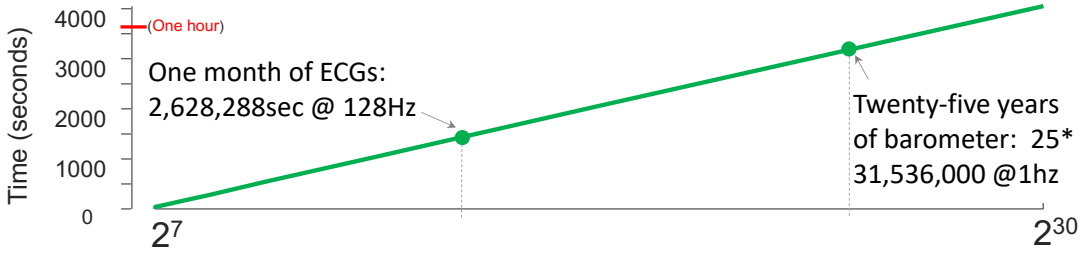

Fig. 25 The time taken for DAMP to process a random walk time series of length 2^{30} (just over one billion). For context, we have labeled the size of two concrete tasks, processing a month of ECGs and twenty-five years of sensor data. Here m =128.

2.5.7 Scalability and Stability of DAMP

One of Wu and Keogh's criticisms of common benchmarks is *unrealistic anomaly density* [35]. They noted that over 20% of the data is labeled anomalous in many benchmarks, which poses a real problem for the evaluation. Suppose that an algorithm has near perfect sensitivity, but it will randomly give out a false positive once in every million datapoints (perhaps due to the numerical instability of streaming algorithms [13]). Note that because most benchmarks in the literature only have a few thousand datapoints, this issue would almost certainly not be observed during testing. However, it clearly would be a problem for any real-world deployment. For example, for a continuous processing system with telemetry reporting every second, this would give us about thirty-one false positives a year.

To demonstrate DAMP does not have this issue, we did the following test. Recall the subtle anomaly shown in the 100,000 datapoint MGAB dataset in Fig. 5. We can append anomaly-free data from the same Mackey-Glass model (but free of the embedded anomalies [31]) to make it one thousand times longer, i.e., a total length of 100 million.

When we search this with DAMP (m = 40), we count a trial successful if the top-1 discord is found in the first 100,000 datapoints (created by [31]), rather than from the appended ninety-nine million nine hundred thousand datapoints. Each of the coauthors of this work ran this experiment multiple times in the background of their desktops over a week, and in total conducted over 16,000 such trials, finding a total of zero false positives.

Note that this experiment required performing anomaly detection on time series with a total length of 1.648 trillion datapoints, using off-the-shelf hardware. This is something that would be inconceivable with any other anomaly detection method.

3. MADRID: A Hyper-Anytime Algorithm to Find Time Series Anomalies of all Lengths

In this chapter, we introduce MADRID, an algorithm to efficiently solve the discords-at-all-lengths problem. We first show that we can reduce the absolute time to compute all-discords by passing information between computations of different lengths.

We then demonstrate that we can cast the search as an *anytime algorithm* [63]. Previous work has shown that some discord discovery algorithms, most noticeably the Matrix Profile, are amenable to being cast as anytime algorithms [67]. However, we will show that MADRID is able to produce very efficient anytime convergence, a property we have named (and will formally define) a *Hyper*-Anytime Algorithm.

Our newfound ability to *compute* anomalies of all lengths produces the downstream issue of *ranking* anomalies of all lengths. We further introduce novel algorithms for this task.

The rest of this chapter is organized as follows. In Section 3.1 we review our motivation and state our assumptions. Section 3.2 introduces all needed definitions and notation, allowing us to introduce MADRID in Section 3.3, which in turn allows us to offer a discussion of the implications of all-lengths anomaly search in Section 3.4. We empirically evaluate our work in Section 3.5. We omit a separate related works section, instead we discuss such material inline and in context.

3.1 Motivation and Assumptions

There has been a recent explosion of research efforts in Time Series Anomaly Detection (TSAD) [42][54][58][62][65]. To make our contribution clear in this cluttered literature, we will begin by stating and justifying our assumptions:

Discord-based anomaly detection algorithms are competitive with the SOTA. This claim seems unimpeachable, based on recent empirical results from multiple independent groups [65]. Some researchers have expressed surprise that such a simple algorithm could be competitive. However, we suspect that discord's simplicity is the *cause* of its competitiveness. Most deep learning TSAD algorithms have at least ten parameters to set, typically using limited data, that is a recipe for overfitting.

When using discords, the choice of subsequence length matters. We have already shown them in Fig. 1. Moreover, this has been empirically observed by several independent research groups [42][46][54][60], and we demonstrate it in Fig. 26 and elsewhere in this chapter.

Creating a heuristic to find the "optimal" window size (even assuming that such a task is well-defined) is a very hard task. Once again, multiple groups have observed this: "*Finding the optimal window size has remained to be one of the most challenging tasks in TSDM domains, where no domain-agnostic method is known for learning the window size*" [42].

As we hinted at above, it is not clear that a single optimal window size is meaningful. A single dataset may have structures and anomalies on multiple scales. For example, in [46] the researchers acknowledge the attractiveness of the subsequence-base anomaly

detection algorithms but bemoan "*a fixed length must be specified in advance, making it a clearly sub-optimal approach for applications dealing with climate data events of varying length*".

Given the above, there is a simple and obvious answer to all these issues. Simply compute the anomaly score for every possible subsequence length! Or equivalently, every possible subsequence length between some widely-spaced minimum and maximum lengths. This completely solves the issue. If the location of the anomalies agrees at all lengths, we are done. However, if multiple anomalies emerge at different lengths (as in Fig. 1, a downstream algorithm can be used to rank/summarize/process them.

Before explaining how we can tackle the apparent untenable time complexity, we will take the time to dismiss the apparent solution to the problem, using some heuristics to find a good compromise length.

3.1.1 On the Difficulty of Finding a Compromise Length

We noted that the community has recognized the difficulty of finding the best value of m for anomaly detection. The basic solution proposed in the literature seems to be to find a "compromise" length. For example, if you suspect that anomalies might manifest themselves at a scale of about fifty to seventy minutes, you could set m = 60 minutes. However, there does not appear to be an understanding as to *why* finding a good compromise value for m is so hard. Here we present some novel observations that show in at least some cases, the task is actually futile. There are simple and obvious anomalies that can only be discovered for a very *particular* value of m. Concretely, our observations are that:

- There exist anomalies that are trivial to find for some value of $m = t$, but completely disappear any value $m < t$.
- There exist anomalies that are trivial to find for some value of $m = t$, but completely disappear any value $m > t$.
- In Fig. 26 we illustrate both cases by using the Matrix Profile to find anomalies for consecutive values of m.

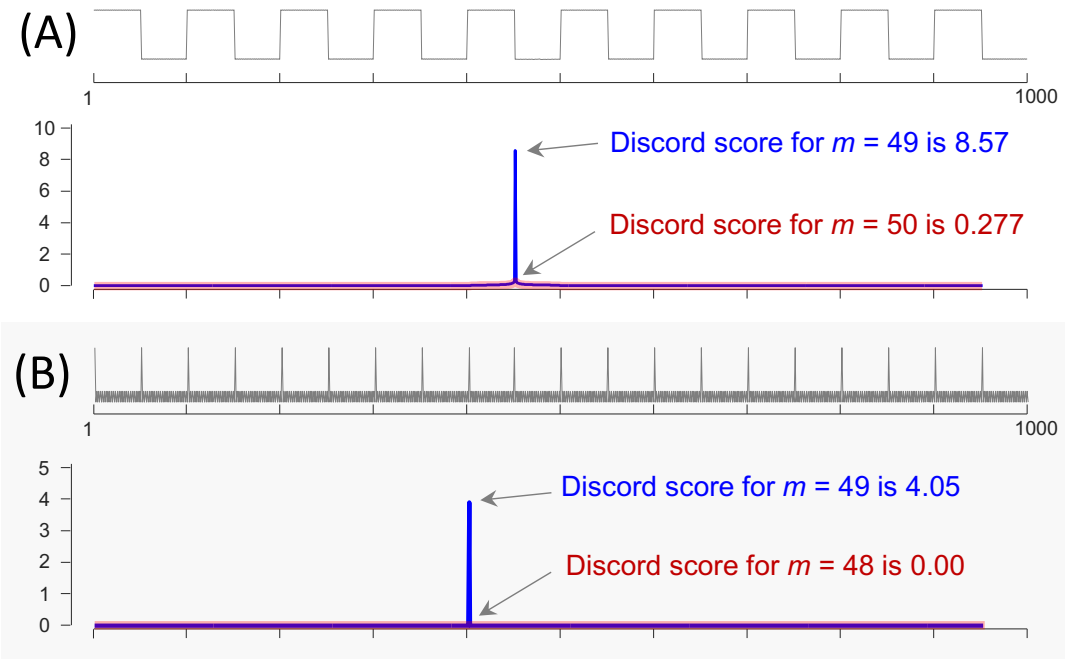

Fig. 26 Two examples to show that anomaly detection can be hypersensitive to the sliding window length. In (A) we can correctly find a subtle anomaly when $m = 49$ but increasing m by just one causes the discord score to plunge. In (B) we can correctly find a subtle anomaly when $m = 49$ but decreasing m by just one causes the discord score to plunge to zero.

Note that for clarity we are showing examples on synthetic datasets, however the effect is seen on real data. Further note that we can combine these observations to contrive a dataset where the value of m must be *exactly* 49, or any other number.

To be clear, we are pointing out a worst-case scenario, the evolution of the discord score with changes m is generally smooth (for example, see Fig. 1, from 12 to 24 hours). However, these observations strongly motivate MADRID's philosophy of testing every length. We can simply bypass the difficulty of finding a good value by testing *all* values.

3.1.2 On the Computational Demands of "all-lengths"

There appears to be a flaw in our proposed "test-all-lengths" solution. Until recently, processing even a *single* subsequence length was considered challenging [65], thus processing perhaps a thousand subsequence lengths appears to be an insurmountable task for realistic deployments. In this chapter we show that we can overcome this apparent bottleneck. We introduce MADRID, an algorithm that is absolutely fast, and for large datasets can be computed in a *Hyper-Anytime* fashion. The term *Hyper-Anytime* will be formally defined later. In brief it means an algorithm that can converge to within 10% of the optimal answer, after using less than 10% of the time needed for full convergence.

3.2 Definitions and Notations

We begin by introducing the necessary definitions and notation. This chapter continues to use all definitions and notations given in Section 2.2 of Chapter 2. In addition to these, there is one new definition as follows.

Definition 9: A *multi-length discord table M* is a two-dimensional array that holds the normalized left Matrix Profiles calculated at multiple scales. Each row represents a left Matrix Profile derived from executing queries of a different length, arranged row by row in a descending order based on query length. Let P_m^L be the left Matrix Profile of a query subsequence of length m and is normalized according to query length m, with a user-

defined query length range $[minL, maxL]$ and a step size S, then $M = [P^L_{minL}; P^L_{minL+S}, P^L_{minL+2S}; \ldots; P^L_{maxL}]$.

In Fig. 27 we illustrate a multi-length discord table *M*.

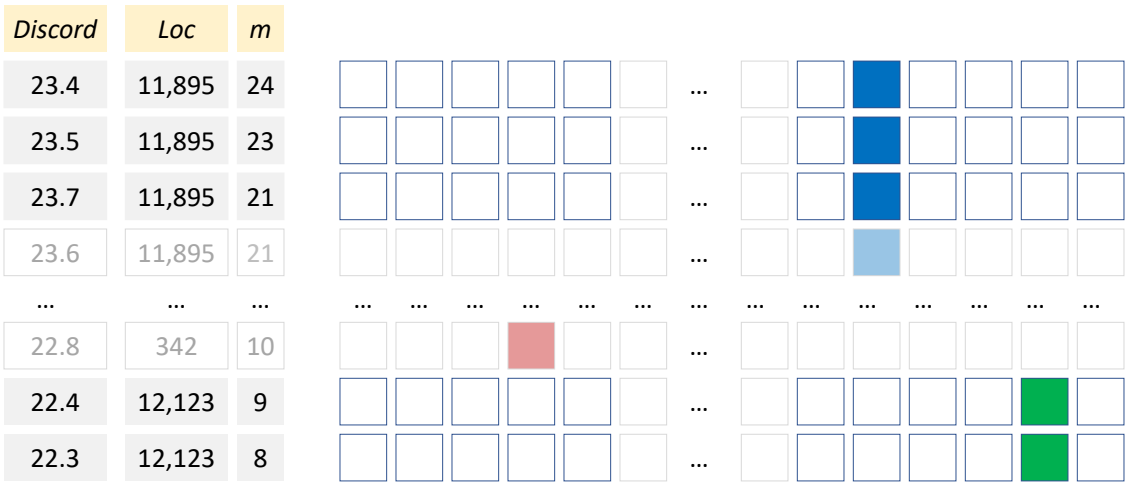

Fig. 27 An illustration of a multi-length discord table M. To ground it in a familiar setting, it is loosely based on Fig. 1.

The left side holds the table's meta-data. Reading from left to right, the top *Discord*, is at the given *Loc,* for the anomaly length of *m*. Computing this meta-data is the task-at-hand. To do so, we can fill in all the values on the right side, record the maximum value in each row (illustrated by the colored cell) and copy that information to the meta-data. As we will show, we can in fact prune most such calculations.

Note that in our example above we consider every value of *m* from 8, the *minL* to 24, the *maxL*. If this is too fine a granularity for a user, they can set a step size, say $S = 2$, and only consider *m* = [8, 10, ..., 22, 24].

3.2.1 A Brief Review of DAMP

MADRID exploits some ideas from the recently introduced DAMP algorithm [65], which we will review here. To be clear, we can think of DAMP as the currently fastest known algorithm to compute *a single* row of multi-length discord table *M*.

DAMP is an ultra-fast time series anomaly detection algorithm that operates on a fixed sliding window size. It defines the discord score by measuring the z-normalized Euclidean distance between the current subsequence and its nearest neighbor in history. That is, its nearest neighbor in the *past*; the algorithm is not allowed to compare to subsequences that occur after its arrival.

DAMP employs two key strategies to speed up the process of anomaly detection, which are also adopted by MADRID:

- **Iterative doubling**: Instead of searching the entire time series in history, DAMP searches back for the nearest neighbor only as far as needed. Initially, it looks back a small, power-of-2 distance for a subsequence with a distance from the current subsequence less than the *Best-So-Far* (*BSF*) discord score. If such a subsequence is found, the current subsequence is disqualified from being a discord, and the search stops. Otherwise, the search length doubles iteratively. Due to the dependency of the time series data, in most cases, iterative doubling requires only one or two attempts to disqualify the current subsequence, pruning over 99% of the search space. The reduction of the search space significantly enhances the speed of the backward search. In addition, efficiency is gained using the MASS algorithm [44], a fast

Euclidean distance computation algorithm based on the Fast Fourier Transform (FFT), which performs optimally for input data lengths is a power of two.

- **Forward pruning**: DAMP integrates a forward search procedure to further improve its speed. The principle is akin to that of the backward search; if there exists a future subsequence that is similar to the current one, then it is unlikely to be a discord and will be skipped in subsequent iterations. Similarly, *BSF* is used as a criterion to prune future subsequences that do not qualify as discords. This method allows DAMP to bypass processing unnecessary subsequences, thereby saving computational time and improving the overall efficiency of the algorithm.

DAMP *is* a fast algorithm, thus a strong starting point for the task-at-hand is to simply loop over all values of *m* using DAMP. However, there are two ways we can improve upon this. The first is to share information across multiple runs, and the second is to create an *anytime* algorithm [63].

3.2.2 Measuring Anytime Algorithm Efficiency

In the title of this chapter, we used the phrase *Hyper-Anytime Algorithm*. This is a definition that we invented. To our knowledge, there is no existing taxonomy or hierarchy to reference the performance of anytime algorithms [63]. We propose the following (loosely inspired by the h-index).

Assume we are measuring the performance of an anytime algorithm in terms of percentages, both in computational resources used (generally wall clock time) and current quality of solution (typically measured in *Root Mean Square Error* (RMSE)). An anytime algorithm is said to be *E*-efficient for the largest number that the following sentence is

true: The algorithm can converge within 1-*E* percent of the final solution using just *E* percent of the computational resources.

For example, in Fig. 28, the performance of the algorithm shown in green is 70%-efficient, as it has converged to within 30% of the final solution using only 30% of the time required by the batch algorithm. As such, we would call it *Ultra-Anytime*. Note that to compute the *E*-efficiency of an algorithm, we can simply find where it intersects a line that connects the two "100%" corners.

With *E*-efficiency defined, we further propose the semantic hierarchy shown in Fig. 28. As we will show in Section 3.3.1 Setting a Strong Baseline for Efficiency, we will propose a variety of algorithms to solve the task-at-hand, culminating with MADRID, which is a *Hyper*-Anytime Algorithm.

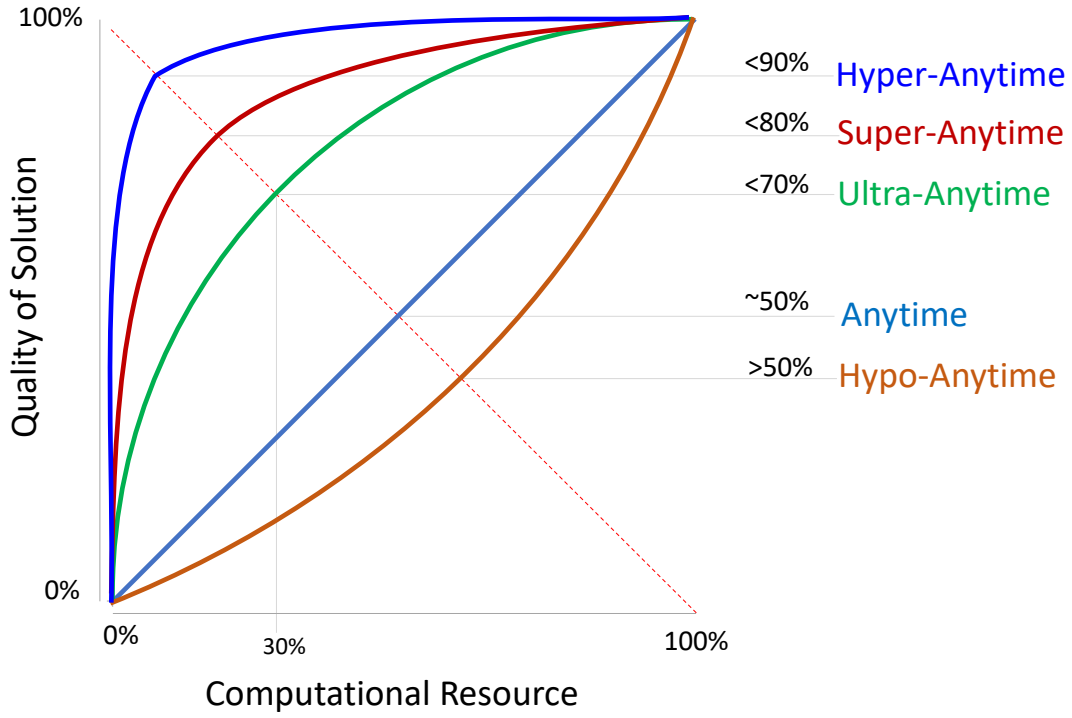

Fig. 28 An illustration of the performance of five hypothetical anytime algorithms. The point where they intersect the red dashed line allows us to rank them with the proposed hierarchy.

3.3 MADRID

We are now in a position to introduce MADRID. The reader will recall the *completed* multi-length discord table *M* shown in Fig. 27. In Fig. 29 we show what this data structure looks like when it is initialized.

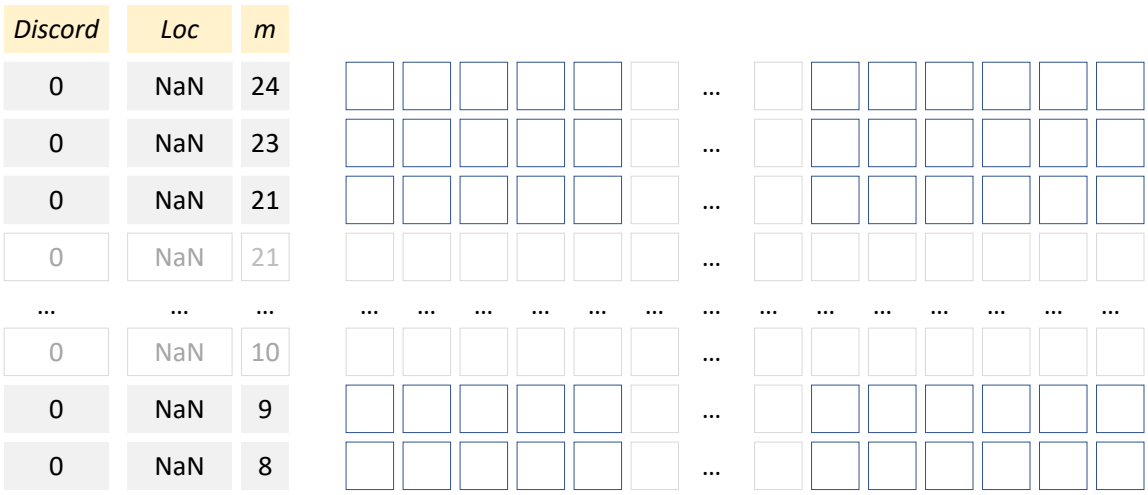

Fig. 29 An initialized multi-length discord table *M*. Compare to Fig. 27. Here the user chose *minL* = 8, *maxL* = 24 and *S* = 1.

Note that there is a simple way we can measure progress when completing this data structure. The current sum of all the numbers in the *Discord* column is zero, let us call this number the *discord-sum*. As we begin to incrementally fill in cells with either their final values or increases in their current *BSF* values, this sum will rise, until it reaches its final value, which happens to be 374.2. This is the variable we will use to track progress in our anytime algorithm framework.

3.3.1 Setting a Strong Baseline for Efficiency

As shown in Fig. 30, before we formally introduce MADRID, we will introduce four baselines, each one progressively better than the last. All four sequentially visit each cell in the multi-length discord table *M* from left to right and top to bottom. However, due to the different search strategies they employ, their processing speed varies greatly.

The first technique is pure brute force. It uses a naïve nearest neighbor algorithm to search for the nearest neighbor in *all* historical data for every subsequence in the sliding window. As shown in Fig. 30.*top* it takes 15.5 hours to finish.

The MASS Brute Force was introduced as a baseline in [65]. It also searches across all historical data, however it avails of MASS [44], which is a highly optimized subsequence search algorithm. As Fig. 30 shows, it is 14 times faster.

By using DAMP, we can further improve efficiency. DAMP accelerates the search by using the iterative doubling and forward pruning techniques we reviewed in Section 3.2.1 A Brief Review of DAMP, significantly reducing the search space and thereby greatly accelerating the process. As Fig. 30, shows it is about 2.3 times faster than a straightforward application of MASS.

While DAMP itself is fast, we can further accelerate it by taking advantage of a unique feature of our problem setting. The efficiency of DAMP greatly depends upon the *BSF* discord score. The larger it is, the more effective the pruning is. For a single run of DAMP at a given value of m, we have no control over how fast m rises to its final value, the discord score. However, in our setting, we expect that in general, the discord score of cell $M_{[Loc,m]}$ will be highly correlated with cell $M_{[Loc,m+1]}$. We compute the first row of M using classic DAMP, which computes (calculating or pruning) the cells from left to right. But for all subsequent rows, we first compute the value of the cell that had the highest value in the previous row. This gives us a large *BSF* discord score from the beginning. We call this optimization *warm*-start. Fig. 30 shows that it gives a further 1.5 times speed-up.

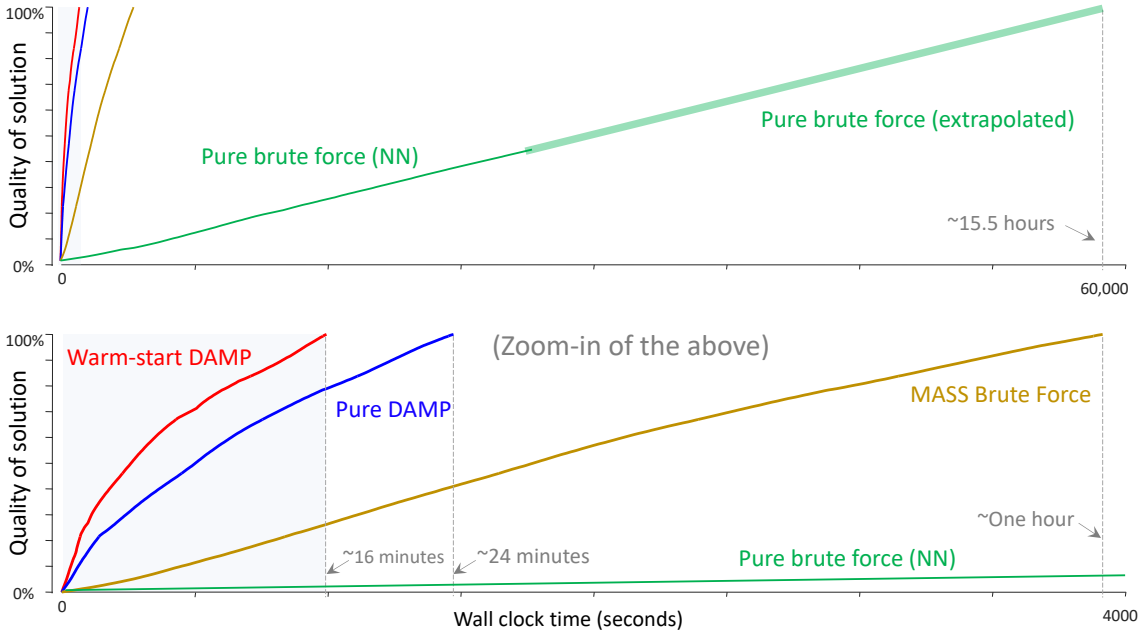

Fig. 30 *top*) The performance of four algorithms tasked with computing *M* for a time series of length 8,192, with *minL* = 128, *maxL* = 768, and step size *S* = 1. As pure brute force visually dominates, in (*bottom*) we show a zoom-in on the other three approaches.

For the rest of this chapter, we will only show results corresponding to the gray shaded area in Fig. 30, having established it as our strong baseline. In addition, in future figures in order to factor out the influence of the hardware being used, we will plot not the absolute time, but the percentage of cells in *M* computed or pruned.

Although warm-start DAMP can achieve further acceleration by leveraging higher *BSF* discord scores, each iteration depends on the results of the previous one, meaning we have to wait for the search on *m* to complete before jumping to *m* + 1. As a result, this is a batch-only algorithm.

However, when dealing with very long time series and/or a wide range of subsequence lengths, warm-start DAMP may take a long time to complete all iterations and report to the user. Is there an anytime algorithm approach [63] that can offer an approximate

answer early in the execution of the algorithm? In response to this, we introduce MADRID, which employs an additional initialization strategy to provide a quick estimation of the top discord for each subsequence length before launching warm-start DAMP.

Fig. 31 illustrates the initialization process of MADRID. Since the DAMP search is very fast for a single window length, we can take advantage of this fact to quickly calculate several top discords of different lengths that are representative of the full range we are tasked with searching. This will provide strong clues for inferring the location of remaining discords at other lengths. But what makes top discords representative? We hope that the most representative top discords can provide more information for subsequent inferences, which means there should be as much diversity among them as possible; they should exhibit the maximum possible differences either in length (m) or location (Loc). As in most cases, the discord location (Loc) of m and $m + 1$ can be adjacent, by maximizing the differences in lengths of three top discords, we are more likely to obtain three widely separated Locs. Therefore, as the first step in initialization, we let DAMP perform searches on the maximum length $maxL$ (here 24), the minimum length $minL$ (here 8), and the mid-length ($maxL + minL$) / 2 (here 16). As shown in Fig. 31, the cells visited by DAMP are marked in gray, and the top discords found by DAMP are highlighted in red.

Discord scores/locations are also updated in the meta-data (red text). Following this, we can utilize the known information, i.e., the red text, to make quick and reasonable inferences about the unknown information in the meta-data.

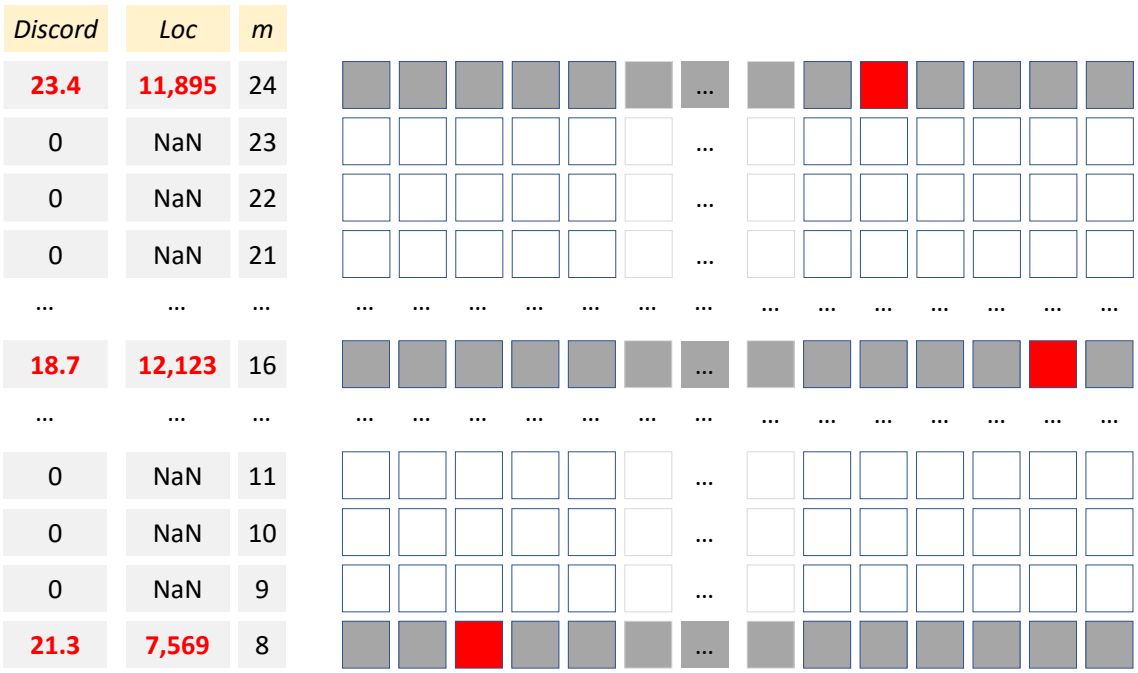

Fig. 31 MADRID's multi-length discord table M after the first step of initialization. As a follow-up to Fig. 29, here *minL* = 8, *maxL* = 24 and *S* = 1.

The second step of initialization builds on the same insight that informs warm-start DAMP. Since the discord score of cell $M_{[Loc,m]}$ will generally be highly correlated with cell $M_{[Loc,m+1]}$, we postulate that cells located in the same column as the red points, while may not end up as the top discord for that row, have a high likelihood of being in close proximity to the top discord, with a score near that of the top discord. Consequently, we compute the scores for all cells in the same columns as the three top discords. In Fig. 32, the columns accessed during this process are colored in yellow. Finally, apart from the rows selected in the first step, each row computes three discord scores at three positions, from which we choose the cell with the highest score as the approximated solution and use it to update the meta-data for that row. The left side of Fig. 32 displays the meta-data after initialization, where we distinguish between exact and approximate values using red

and black colors respectively. The red fonts in the *Discord* and *Loc* columns represent exact values that match the final results and will not be modified in subsequent computations. Meanwhile, the black fonts are approximations, which may be corrected in subsequent calculations.

After the initialization process, MADRID initiates warm-start DAMP to calculate the exact scores and positions of top discords row by row. This time, warm-start DAMP can utilize the discord scores obtained during the initialization stage, compare them with the *BSF* discord score returned after processing the previous row and select the higher score as the initial *BSF* score to start searching the current row.

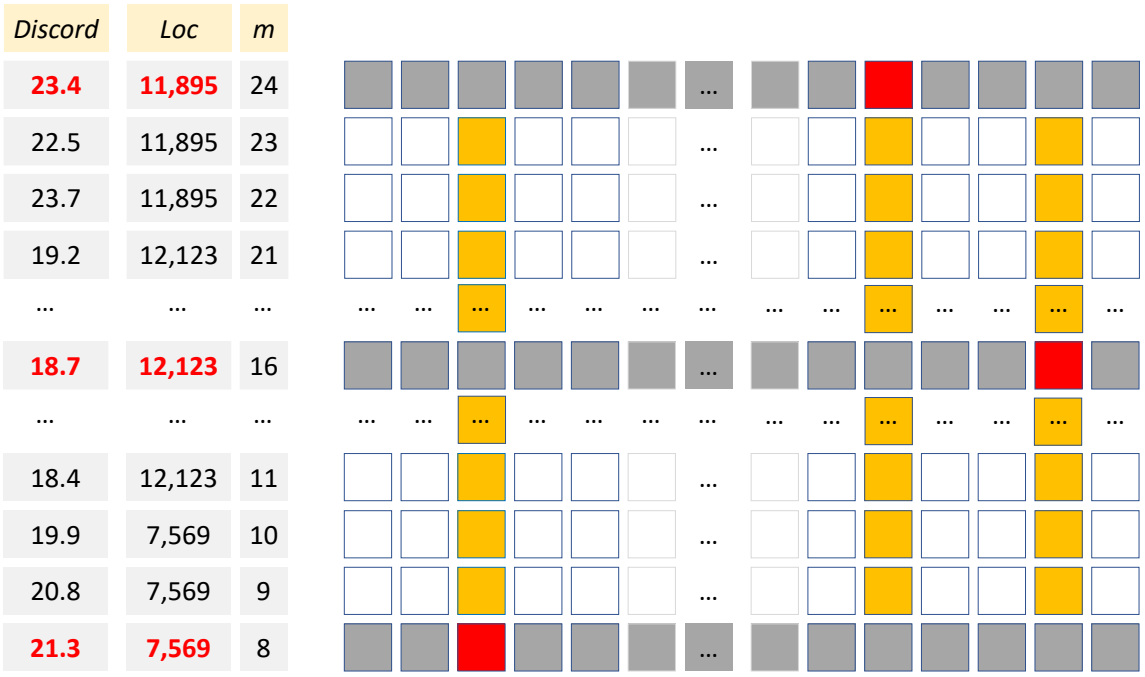

Fig. 32 MADRID's multi-length discord table M after the second step of initialization. As a follow-up to Fig. 31, here *minL* = 8, *maxL* = 24 and *S* = 1.

3.3.2 Making Different Length Subsequences Commensurate

We have glossed over one important issue in the above. We propose to find anomalies at lengths that may differ by more than an order of magnitude. Naturally, all things being equal, longer anomalies will tend to have higher discord scores. However, we would like the discord scores to be commensurate across different lengths for two reasons.

- The effectiveness of MADRID's search strategies is based on passing information between subsequence lengths; this is most effective if the subsequence lengths are commensurate.
- We may wish to report the most *natural* length anomaly. However, unless we can make the different lengths commensurate, the longest discord will almost certainly have the highest discord score.

Several research efforts have proposed normalizing different length Euclidean distances by dividing by the subsequence length, however this overcompensates. As Fig. 33 shows, the correct normalization is to divide by the square root of the subsequence length.

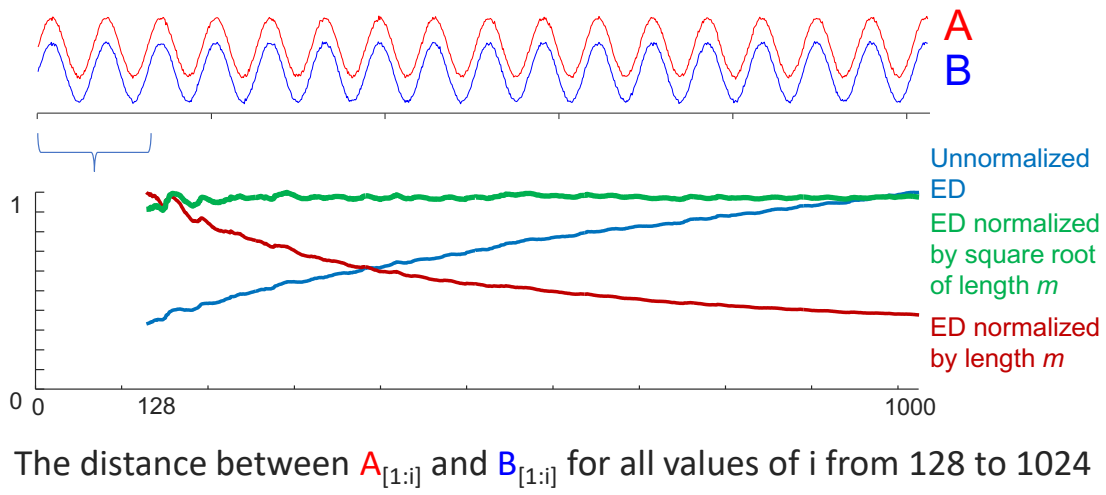

Fig. 33 *top*) Two slightly noisy sine waves. *bottom*) We measured the distance between the prefixes of these sine waves of every length from 128 to 1024. The unnormalized distance grows, the length normalized distance plunges, but the proposed "divide by the square root of m" normalized distance remains almost constant.

For brevity, we relegate the derivation of this normalization factor to [51].

3.4 Discussion

Before beginning the experimental evaluation, we will take the time to discuss the implications of an all-lengths search for anomalies.

3.4.1 MADRID Solves the Core TSAD Problem

We have seen that MADRID is an effective TSAD algorithm, here we make a stronger claim. MADRID solves (or rather *bypasses*) *the* core TSAD issue, parameter tuning.

Most researchers who attempt to evaluate TSAD algorithms noted the extreme difficulty of setting parameters. For example [62] notes "*specifying the hyperparameters of anomaly detection is particularly difficult because it requires an in-depth understanding of the data and the algorithm*", [61] bemoans "*a full parameter grid search is clearly*

infeasible" and [42] laments "*..window size for extracting such subsequences is a crucial hyper-parameter*".

Note that these are the comments of *experts* in TSAD, presumably the technicians tasked with using the algorithms in factories/hospitals would have even more difficulty. The reader will appreciate that MADRID completely solves this issue, there are simply no parameters to set[6].

Many other research efforts make an unspoken assumption before introducing their proposed approach. Here we will make that assumption concrete, and show it is unwarranted. The (near universal) unspoken assumption of TSAD is:

If there is an anomaly of length *K* in a dataset, and we set the sliding window length of a sensitive algorithm to be *L*, a number smaller than, but still a large fraction of *K*, then we will probably find the anomaly. This is because a strong anomaly of length *K* surely is comprised of sub-regions that are at least somewhat anomalous.

To show that this is an unwarranted assumption, let us create a simple synthetic dataset. As shown in Fig. 34.*top*, the normal data simply consist of *two* small bumps, followed by a large bump. The anomaly we created is similar, but there are *three* small bumps. This anomaly is visually obvious.

[6] For the pedant, *minL* can be set to 3, the smallest subsequence that can be z-normalized, and *maxL* can be set to an arbitrarily high number, and *S* is defaulted to one. Thus MADRID is truly parameter-free.

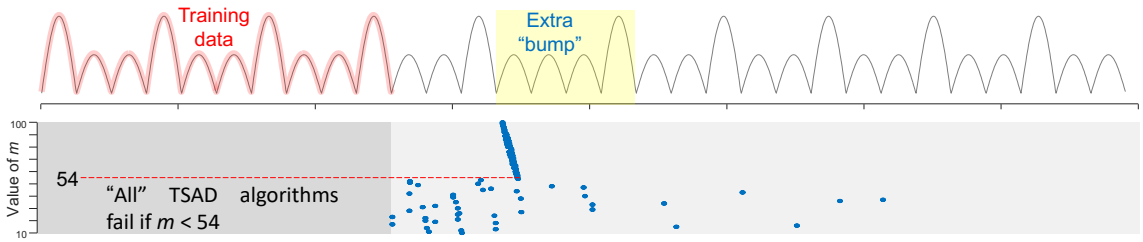

Fig. 34 *top*) A trivially simple anomaly detection problem is unsolvable for any algorithm that considers a sliding window length m < 54. *bottom*) However, MADRID with *minL* = 10, *maxL* = 100 and *S* = 1 easily solves this.

As shown in Fig. 34.*bottom*, MADRID$_{[10,100,1]}$ solves this problem. It finds the anomaly in two ways. The random scattering of the top discord location for m = 10 to 53 suggests that these are not true anomalies, whereas the stability of the top discord location for m = 54 to 100 is evidence of a strong significant anomaly. Moreover, the discord scores (see [51] for a 3D version of this plot) rise dramatically as we transition from 53 to 54 and beyond.

We have tested DAMP, the Matrix Profile, OmniAnomaly and Telemanom on this dataset with a sliding window of length 50, and they all fail to find this obvious anomaly. Moreover, although we clearly have not tested all of the dozens of TSAD algorithms out there, from their descriptions we believe that most or all of them will fail here.

In retrospect this finding seems obvious. By analogy, the last decade has seen extraordinary progress in facial recognition algorithms. However, Fig. 35 invites us to imagine we forced the algorithms to use a single-sized bounding box for all images.

Fig. 35 The futility of the "one-size-fits-all" unspoken assumption of TSAD is obvious if we consider its analogue in image processing. Any face processing algorithm would clearly find the images at the left or right extremely challenging.

This observation has implications not only for our championing of MADRID, but even for a retrospective review of previous comparisons of TSAD algorithms. By carefully choosing the right datasets and the "right" value for *m*, it is possible to achieve almost any relative ranking of any competing approaches. No empirical comparison we are aware of seems to have carefully considered this issue.

Finally, here we used a synthetic dataset for clarity, however the empirical results in the next section suggest that this is a real issue. In many settings, a single-length TSAD algorithm can be effective if the perfect subsequence length is chosen but can fail if the value is even slightly off.

3.5 Experimental Evaluation

To ensure experiments are easily reproducible, we have built a website [51] that contains all the data/code used in this chapter. All experiments were conducted on an Intel® Core i7-9700CPU at 3.00GHz with 32 GB of main memory.

In our experiments we need to demonstrate two things. First, that there are anomalies that MADRID can discover that will escape our attention if we use a *single* window length application of the MP, or *any* competitive TSAD algorithm.

Second, we need to show that our casting of MADRID as an anytime algorithm allows us to find discords quickly. So quickly, that for many realistic scenarios a user can interact with historical archives in real-time interactive sessions.

3.5.1 Revisiting Melbourne Dataset

We revisit the Melbourne dataset shown in Fig. 1, this time considering foot traffic in *Waterfront City*, a popular shopping area, from April 2009 to January 2018.

In [65] the authors noted that deep learning approaches have a hard time *generalizing*. For example, if a deep learning model is trained in the *winter*, it may fail to generalize to the *summer*. This is not an issue for MADRID; once a subsequence is inspected, it is instantly ingested into the model, thus we are largely invariant to concept drift. Thus, to be fair to deep learning models, we use the first full year as training data, and the remaining eight years as test data. This way, the model has seen all the annual seasonal variability and all annual cultural events (Xmas, national holidays, etc.)

We tasked MADRID to return only the top-1 anomaly at *each* length. Because we are considering 44 distinct lengths, the algorithm *could* have returned between 1 and 44 different anomalies, however it actually reports *eight* district anomalies. Because we have made the different lengths commensurate, we can rank them. The top three anomalies (Fig. 36) are:

- **15-hours**: Police say widespread **flash flooding** is beginning to affect central areas around Melbourne [56]
- **19-hours**: Thousands of AFL fans have created a carnival-like atmosphere as they... **final parade** [40]

- **39-hours:** ...public holiday in **Remembrance Day** [41]

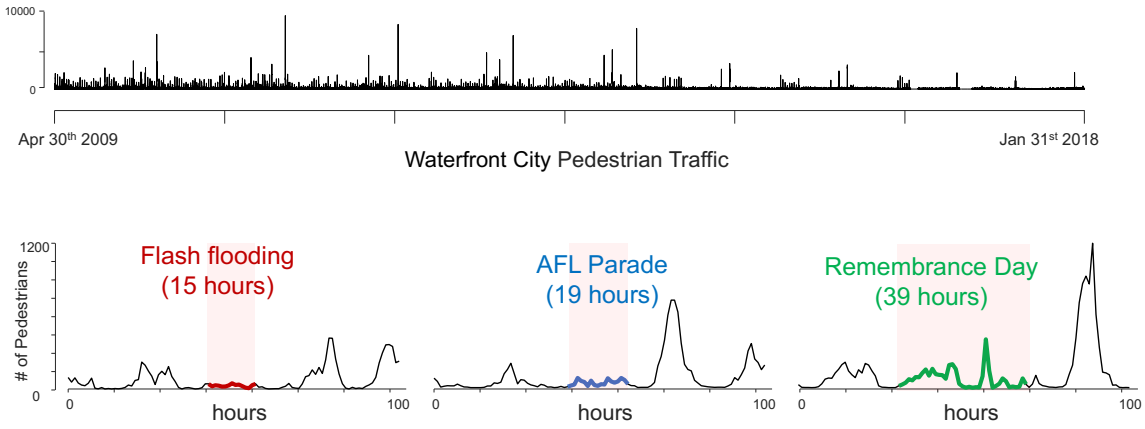

Fig. 36 *top*) About eight years of pedestrian traffic. *bottom*). MADRID with *minL* = 4, *maxL* = 48 and step size *S* = 1, finds eight distinct anomalies, the top-3 are shown.

Interestingly, these three anomalies are from different causes: an unpredictable weather event, a sporting celebration, and a somber cultural event.

We compared to Telemanom, one of the most cited TSAD deep learning models [52]. It would take an estimated 8.6 hours to test all 44 lengths, Instead, we tested just the shortest, longest, and median lengths. All three lengths found the same anomaly, the 2014 national holiday. There is a sense in which this could be seen as a positive feature, that the algorithm is "stable". However, it also hints at the fact that the algorithm is biased towards certain types of anomalies, independent of the user's selection of window length, meaning that certain types of anomalies could be hard to find.

MADRID took 40.5 minutes to fully converge and is semantically converged after just 6.5% of the computations have been completed. Telemanom is an order of magnitude slower. That is not untenable, especially for data that took nine years to collect, but it does prohibit *real-time* interaction with the data. As [47] notes "*In interactive data*

analysis processes, the dialogue between the human and the computer is the enabling mechanism. It is of paramount importance that this dialogue is not interrupted by slow computation". Using MADRID, especially with its anytime feature, a user could run TSAD for a few minutes and then peek at the results. Based on what she sees, she could perhaps edit the data "*Oh, I should delete all of the summer of 2012, due to that construction period"*, and search again. This type of interaction is simply not tenable if each cycle takes hours.

3.5.2 Revisiting HEX/UCR Anomaly Dataset

Recently, the HEX/UCR Anomaly Dataset has emerged as a standard benchmark for time series anomaly detection. One of its key advantages is that it provides a standardized scoring function to facilitate reproducible and fair evaluation among different papers. As demonstrated by several independent papers [58][65], DAMP is highly competitive in the domain. Since DAMP is the core subroutine of MADRID, that bodes well for our proposed approach.

DAMP offers a heuristic for automatically suggesting a value for *m*. The method is simple, and claimed to be effective for most datasets, but it is not perfect and can fail with datasets that do not have a single distinct periodicity. Because MADRID considers a *range* of window lengths, we can increase the chance of successfully detecting anomalies. Based on the *m* value suggested by DAMP, we simply set the *minL* for MADRID to be one-third of this value and *maxL* to be three times this value (rounding to the nearest integer).

Consider the following two examples:

We begin by testing the UCR-202 dataset, for which the DAMP's heuristic suggested a window length of 32. Therefore, we set the range of MADRID's window lengths from 10 to 96. Fig. 37 displays the results of MADRID and DAMP algorithms executed on this dataset. The x-axis represents the window lengths used by MADRID, the y-axis represents the positions in the input time series, and the z-axis denotes the anomaly scores. We plotted the top discords found using different window lengths as "stems" in the figure, where successful MADRID predictions are marked with red stems, failed MADRID predictions with blue stems, and the DAMP prediction with a green stem.

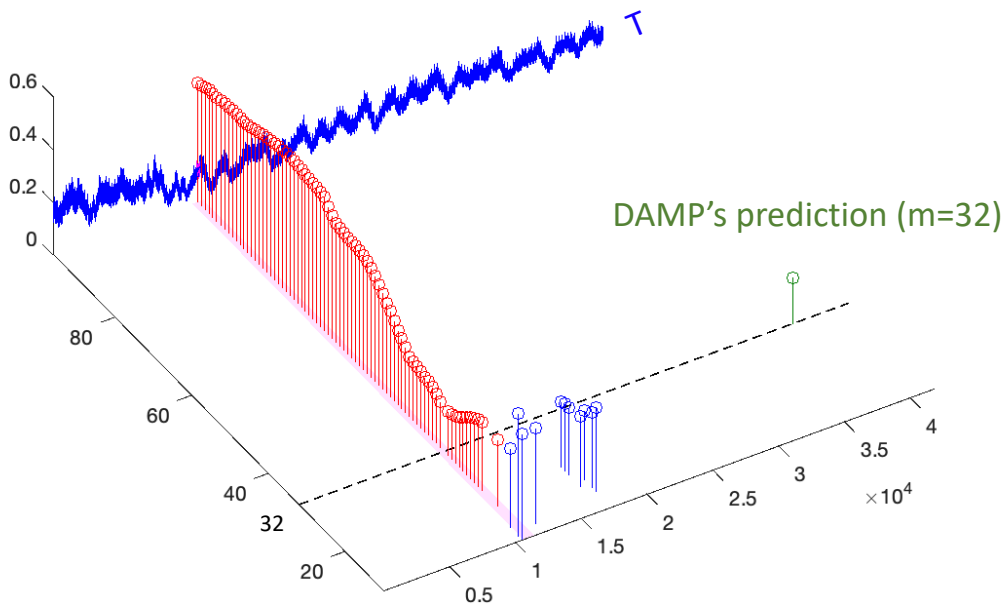

Fig. 37 The 3-dimensional multi-length discordance table of dataset UCR-202, where *minL* = 10, *maxL* = 96, and step size *S* = 1.

From Fig. 37, we can easily see that most of MADRID's predictions successfully identified the ground truth anomaly. Of the 87 predictions of MADRID, 75 are

objectively correct. Or if we report only the location with the highest (length normalized) discord score, we are also correct.

In contrast, DAMP uses a window length of 32, which is only one less than the correct window lengths used for successful predictions. However, this small difference of only one value caused DAMP to generate a completely wrong prediction result that is far from the ground truth.

The UCR-202 dataset is not the only such example. For example, we also observed nearly identical results for UCR-113. For that dataset, MADRID used a window length interval of [8, 72] and step size 1 to predict the anomaly, with the results showing that 60 out of the 65 predictions are correct. Here DAMP used an incorrect window size of 24, resulting in a difference of nearly 2000 units between its predicted location and the ground truth location. There are more examples; however, due to space constraints, we invite interested readers to view the detailed experimental results/visualizations of other examples at [51].

Although the success of MADRID is evident in the previous examples, The reader may argue that MADRID's higher success rate is simply because it attempts more window lengths and thus reports more anomaly locations. Since MADRID makes multiple predictions while DAMP only makes one, this comparison might seem unfair to DAMP. Therefore, for situations and downstream algorithms that require a *single* answer, how can we summarize MADRID's multiple predictions into a single value?

We believe that there is no *single* answer to this question, here we highlight some possible approaches:

Human Inspection: In many cases, the ultimate filter of anomaly detection is human inspection. Anyone glancing at Fig. 37 would surely come to the right conclusion. The instability of locations for $m < 34$ suggests that such short lengths are inappropriate for this domain, but there *is* a significant anomaly here.

Highest Score: Because we have made the discord score commeasure, we can simply report only the location of the *highest* score. In Fig. 37 this would return the correct answer ($m = 71$ has the highest discord score of 0.52). This idea generally works well, but there is a danger of obtaining a false result for a small value of m, if we allow a very low value of *minL*.

Clustering: Here, we construct a cluster-based approach to summarize the results returned by MADRID. First, we employ DBSCAN to cluster the top discords reported by MADRID based on their two-dimensional coordinates [*Loc*, *m*] in the multi-length discord table, which can be accessed in MADRID's meta-data. Typically, when DAMP searches on excessively short lengths m, its predictions are unstable, the distribution of Loc is discrete and random. DBSCAN can address this issue by adjusting its epsilon parameter to exclude sparsely distributed points from the clusters. After clustering, to measure the importance of each cluster, we calculate the sum of the discord scores of all top discords within each cluster as the cluster's weight. We do this because the sum of the discord scores is related to both the magnitude of each discord's score within the cluster and the number of discords in the cluster. These two elements are crucial for evaluating the importance of the current cluster. Finally, we sort the clusters in

descending order based on their weights and return the position of the centroid of the highest-weighted cluster as MADRID's best prediction.

If we apply any of these three methods (recognizing that "1" is not a true algorithm) we can significantly boost the performance of DAMP, which is currently SOTA on this archive [65]. However, we will not report the actual number, as there is a chance of being accused of HARKing (hypothesizing after the results are known) [55]. Here the accusation may be justified, as we took advantage of the high certainty ground truth[7] to understand these issues and design these solutions.

3.5.3 Scalability

Here we test the scalability of our proposed algorithm. We consider two synthetic datasets of lengths 100,000 and 1,000,000. In both cases, the first 50,000 datapoints were used as training data. As shown in Fig. 38, the synthetic datasets have three anomalies embedded into them. These datasets are designed to be easy, and to have an unambiguous ground truth. Recall that our baseline is warm-start DAMP (cf. Fig. 30), which is already much faster than the application of pure DAMP. To appreciate the anytime properties of MADRID we plot the results inside a unit square as shown in Fig. 28.

[7] A recent paper offers forceful evidence that many TSAD benchmarks have high undocumented uncertainty in their ground truth [59].

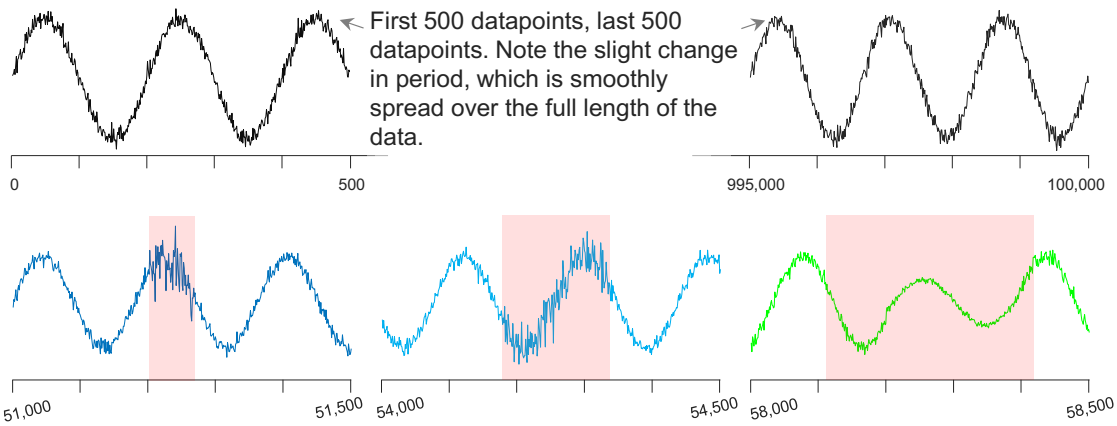

Fig. 38 *top*) The beginning and end of the test datasets. *bottom*) The three anomalies embedded into the synthetic data are visually obvious with human inspection.

We also want to test the *semantic convergence* of MADRID, to ensure that RMSE measured is a good proxy for the task-at-hand. We therefore ask the following question: at what point could we have stopped MADRID and have the *majority* of the *Loc* pointers point to one of the three anomalies locations? As shown in Fig. 39, this happens after only ~1.55% of the total computation was completed, at which point there were 58 pointers pointing to the first anomaly, 89 pointing to the second anomaly, and 46 pointing to the third anomaly, confirming MADRID's strongly anytime behavior.

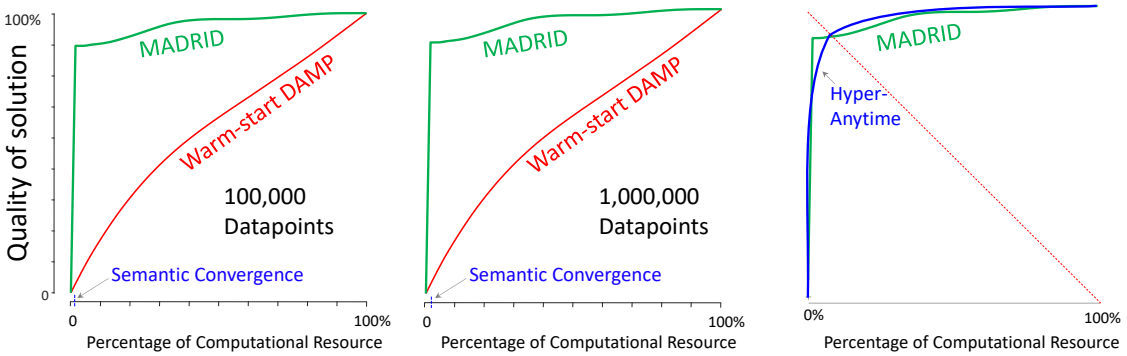

Fig. 39 Anytime convergence plots for MADRID on 100K (*left*) and 1,000K (*center*), datapoints. In both cases, *minL* = 64, *maxL* = 256 and step size *S* = 1. *right*) Overlaying MADRID's convergence plot onto the nomenclature template shown in Fig. 28 suggests that MADRID is a Hyper-Anytime Algorithm.

It is difficult to see in the above plot, but MADRID's convergence is slightly faster for larger datasets, and in both cases the algorithm is a Hyper-Anytime Algorithm.

MADRID returns a real-valued anomaly score for the most anomalous location at each length. However, how do we make the binary decision as to sound an alarm or not? This is simple. We use the training data to sample time series of a random length in the range *minL* to *maxL*, and then find their nearest neighbor (i.e., their "discord score", in spite of the fact that they are not anomalous by definition, coming from the training data). We can then use the classic "mean-plus-three-standard-deviations" technique as an anomaly threshold. As Fig. 40 shows, this correctly flags all three anomalies.

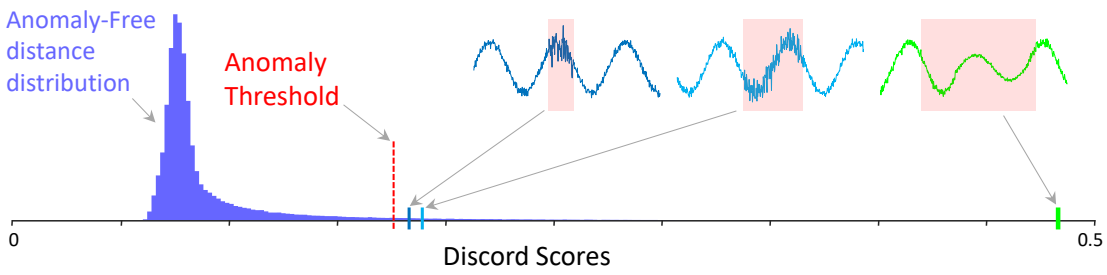

Fig. 40 The classic "mean-plus-three-standard-deviations" rule correctly flags anomalies shown in Fig. 38.

Having established MADRID performance, we measured the performance of two of the most cited TSAD algorithms, OmniAnomaly [66] and Telemanom [52] on the smaller dataset. We set their sliding window to be 160, the midrange of lengths that MADRID considers.

Using OmniAnomaly's default (binary) predictions, it reports forty positives, two of which are true anomalies. If we instead use its internal real-valued measure of the "strength" of anomalies to find the top three anomalies, we find none of them are true positives. We used the default parameters, perhaps changing these parameters could improve performance, however the training/test times of 352 minutes/10 seconds do not invite the user to interact with the algorithm. Telemanom does much better, correctly reporting all three anomalies. However, its training/test times of 111 minutes/23 minutes are also very slow for such a small dataset. In contrast MADRID takes a total time of 3.9 minutes and converges on the correct answer in just 34 seconds.

In summary, in this dataset MADRID can test 192 different subsequence lengths much faster than the SOTA deep learning algorithms can test a *single* length, and MADRID successfully finds *all* anomalies.

There are two other algorithms that we can directly compare to: MERLIN, and the PAN-Matrix Profile. To be fair, to the PAN-Matrix Profile, we note that it is solving a more general and more difficult task, as it is computing the full Matrix Profile at every length (i.e., *motifs* and discords), whereas MADRID and MERLIN are only finding discords. On the 100K dataset, MERLIN takes 3.6 hours and the PAN-Matrix Profile takes 1.8 days.

3.5.4 Case Study in Industrial Data

The core motivation for MADRID is that there may exist anomalies of very different lengths within a single dataset. Here we show this to be true on an industrial conveyor system known as HRSS [53]. The system consists of multiple high-speed conveyor belts that can move an object along both the horizontal and vertical axis (A video of the system is at [53]). The dataset is interesting because the technicians, after allowing 48 normal cycles, begin to introduce *physical* obstructions that resulted in anomalies, thus we have an unambiguous ground truth. We ran MADRID on a voltage trace from the system, with *minL* = 64 and *maxL* = 256.

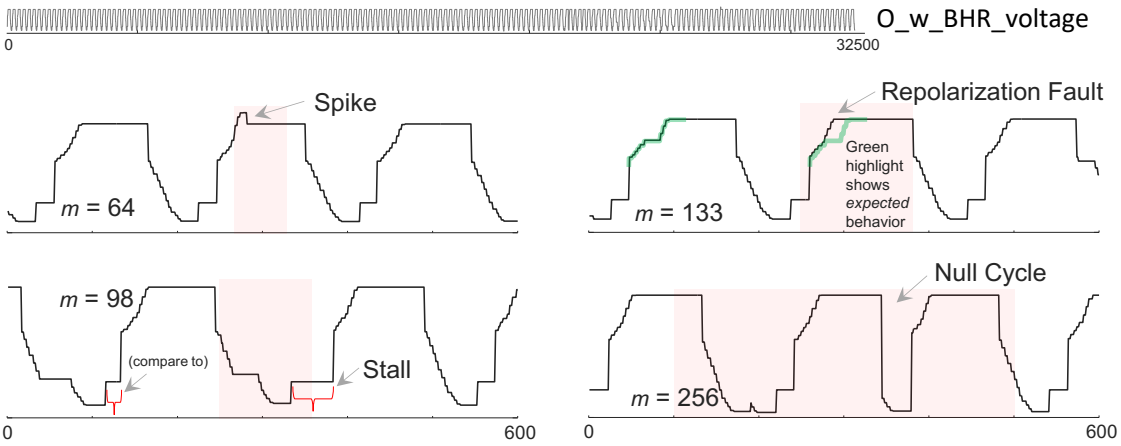

Fig. 41 *top*) A 27-minute-long O_w_BHR_voltage trace from HRSS. *bottom*) MADRID with *minL* = 64, *maxL* = 256 and step size *S* = 1, finds four distinct anomalies, all true positives.

MADRID reported four distinct different length anomalies, all true positives. If we use the commonly proposed heuristic of setting a sliding window length to about one cycle, almost all algorithms (including the Matrix Profile) will fail to find the two shorter anomalies, *Spike* and *Stall*.

Another interesting observation is that for the *Repolarization Fault*, MADRID detects the *first* occurrence of it, and notices the following cycle has the same fault. This shows that MADRID is "twin-freak" invariant, a property we inherit from our use of DAMP. In contrast, the classic Matrix Profile fails to find this anomaly. Finally, even though we searched over a large range of *m*, at the finest resolution possible, MADRID is still much faster than real-time here.

3.5.5 Case Study in PSML

To further investigate multi-scale anomalies, we explored the PSML dataset [64], which contains relative humidity, temperature, and electricity load for CASIO Zone 1 (Northern California). We used the entirety of 2018 as training data to detect anomalies occurring during 2019 and 2020. Since the PSML dataset is sampled every minute, our training and testing data have lengths of 0.5 million and 1 million, respectively. To ensure that MADRID can perform fine-grained and efficient searches on such a large-scale dataset, we set $minL$ = 720 (half a day), $maxL$ = 10,080 (one week), and step size S = 720 (half a day).

MADRID detected various multi-scale anomalies in three dimensions: relative humidity, temperature, and electricity load. Due to space constraints, we only report the search results for relative humidity here. For the rest of the results, please refer to [51].

As illustrated in Fig. 42, MADRID identified four distinct relative humidity anomalies, each corresponding to extreme weather events of varying durations. The shorter anomalies captured by MADRID are associated with three different storms - Winter Storm Kai [42], Winter Storm Nadia [45], and the Thanksgiving week storm of 2019

[50]. These storms brought about brief but intense precipitation, thereby causing a short-term increase in relative humidity. All the longer anomalies, extending beyond two days, are caused by the Kincade Fire, which ravaged Northern California for a protracted period of 15 days. As reported, due to the wildfire spread, "*humidity remained critically low*" [57].

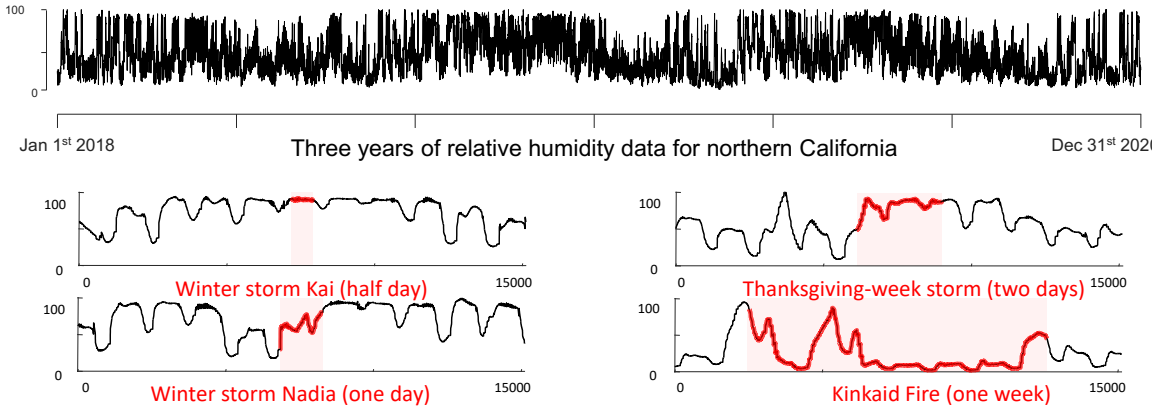

Fig. 42 *top*) Three years of relative humidity data for Northern California from PSML. *bottom*) MADRID with *minL* = 720, *maxL* = 10,080 and step size *S* = 720, finds four distinct anomalies.

MADRID required 10.6 hours to conduct 14 searches of varying lengths on 1 million data points and report the four meaningful anomalies of different lengths. We made every effort to make Telemanom and OmniAnomaly work on this dataset, however the size of the training data completely defeated them. We documented these efforts at [51].

4. FIRE makes any Time Series Anomaly Detection Algorithm Faster, More Accurate and more Practical

Time series anomaly detection has become an increasingly important task with our dramatically increasing ability to sense the world. The time series anomaly detection (TSAD) task is typically framed as follows: Given some training data, build a model to predict the occurrence of anomalous subsequences.

In this chapter, we argue that any practical definition of TSAD must be generalized to: Given some training data, predict when a user would claim a subsequence is anomalous. This is a simple but critical distinction, but as we will show, without understanding and modeling the user's knowledge and requirements, most TSAD algorithms are doomed to be plagued with false positives. Moreover, false positive fatigue is widely understood to be the greatest barrier to adoption for TSAD algorithms. After demonstrating this claim, we introduce FIRE, an intuitive framework to allow us to represent the user's knowledge and requirements and communicate them to the anomaly detection algorithm. As we will show, FIRE is algorithm and domain agnostic, and it can make anomaly detection faster, more accurate and more useful.

4.1 Motivation and Observations

We begin by stating the fundamental observations that inform our work. The reader may find these observations obvious or even bordering on tautological. However, we are not aware of their concrete statement in the literature.

Fundamental Observation 1: For virtually any domain, and for virtually any ADA, there will be patterns observed that are not anomalies, but they will trigger the ADA. We call such patterns OMITS (Obviously MeanIngless Time Series).

To illustrate, consider the time series shown in Fig. 43.

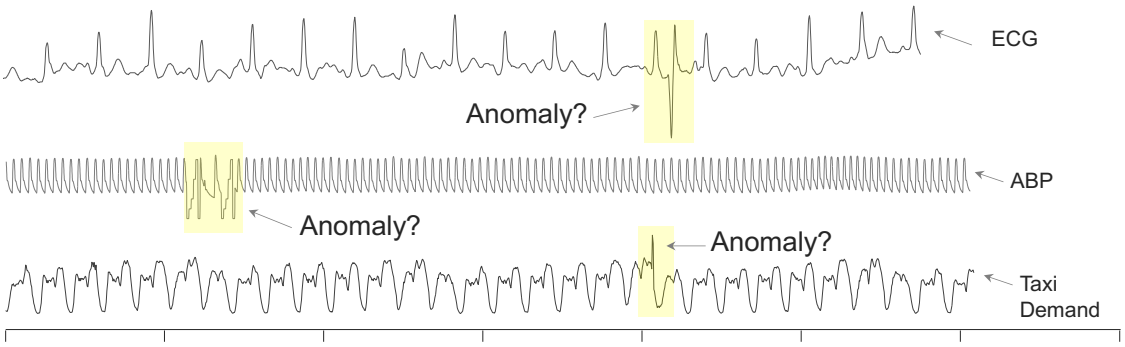

Fig. 43 Three independent time series that apparently have strong anomalies in the yellow highlighted regions. As we will show in this chapter, these should not be considered anomalies.

The reader may be surprised by these examples, the highlighted regions *appear* to be visually obvious anomalies. Indeed, the Taxi Demand example shown in Fig. 43.*bottom* has been flagged as an anomaly by several hundred papers. However, the Taxi "anomaly" is caused by a simple formatting quirk of dealing with Daylight Saving Time. Given that this spike could be predicted years in advance, it makes no sense to call it anomalous. By analogy, for millennia comets were literal anomalies in the sky, but the 1758 appearance of Halley's Comet was *not* an anomaly, it had been predicted decades early.

This introspection leads us to a new observation.

The NY-Taxi example shown in Fig. 43.*bottom* has an interesting history. In the original dataset there is a labeled anomaly for Marathon, which did indeed take place on Sunday, November 2nd 2014. Of the five labeled anomalies in this dataset, this is the most visually obvious anomaly, and over one hundred papers report being able to discover it. However,

the large visually obvious spike does not happen at during the late morning/early afternoon when the athletes were running in the marathon, it happens exactly at 2:pm, when the clocks move back for Daylight Saving Time. It is obvious that a naïve processing script *summed* the taxi demand for the two hours then mapped that value to one hour, rather than *averaging* them, creating an apparent spike in demand. Wu and Keogh [35] noted that many datasets suffer from similar mislabeling.

Fundamental Observation 2: Most definitions of 'anomaly' make reference only to predictions made based on properties of a data set. However, any *practical* definition of 'anomaly' should instead predict how a given *user* would label the data. Moreover, the definition must allow for the fact that the user may have access to arbitrary out-of-band data and domain knowledge.

We can use Fig. 43.*middle* to illustrate this. If we look at the highlighted shape, every ADA we are aware of will strongly signal that region as containing an anomaly. However, there is an important domain-specific fact that experienced users in this domain know. This data comes from a sensor that is glued to a patient's skin. If the sensor has a weak contact with the skin, it can send erratic signals, but critically, those signals have very low cardinality. Thus, a medical doctor will realize that these strange signals are *not* biological anomalies to be investigated, just low-quality data to be ignored.

Finally, we note that even within an single domain, the definition of anomaly can be idiosyncratic.

Fundamental Observation 3: Time series anomalies can be subjective. We can *only* meaningfully define anomalies with reference to purpose of monitoring.

To illustrate consider Fig. 43.*top* . Without context the highlighted pattern appears to be an anomaly in someone's electrocardiogram. However, as Fig Y shows, this dropout is the result of the body-mounted sensor experiencing a dramatic acceleration. Is this pattern an anomaly? To a cardiologist, the answer is *no*, it is just corrupted data. However, to the bedside vascular technologist who is in charge of collecting the data, the answer is *yes*. They are charged with producing high-quality data, and some anomalous event, perhaps a nurse tripping over the sensor cable, has caused an anomaly that that the technician must address, perhaps by tying up the loose cable.

This realization that TSAD is an inherently subjective task initially appears to be bad news. By analogy, we are used to *objective* classification problems like {cat | dog}. There really is an unambiguous definition of cat (i.e., any animal descended from *Pseudaelurus* [68]) and dog. The problem may still be challenging, perhaps the animals are seen from unusual angles or partly obfuscated by furniture. However, no one doubts that there is a ground truth in principle.

In contrast, suppose that instead we are asked to classify {cats-ann-likes | cats-ann-does-not-like }. This subjective problem initially may see like a much harder task. However, suppose we are allowed to interview Ann, and she informs us that "*I like cats that are fat and I like cats that are a single color*". This now becomes a much easier problem, given only that there is some way to communicate Ann's preferences to our algorithm. Note that the problem is now much easier, but still not trivial, we need to be able to learn the animal's mass and color.

To summarize, we claim that anomaly detection needs to be reframed as the problem of predicting when a given user would claim a subsequence is anomalous.

Given the above motivation, the obvious question is how we should represent and communicate this user information to the ADA? We propose a data structure called FIRE. FIRE is a vector that is parallel to the time series being examined, its it used to communicate between the user and the ADA. As we will show, FIRE can make anomaly detection algorithms faster and more accurate.

4.1.1 Anomalies can be subjective

The fact that anomalies can be subjective has significant implications for evaluation of TSAD algorithms. The *Inter-rater reliability* (IIR) of the ground truth labels provided for a benchmark dataset creates limits on the claims that can be made. For example, if Cohen's kappa is say, 0.7 on a dataset, it would be ludicrous for someone to claim that algorithm **A** is better than algorithm **B** because the former had an F1 score that was 5% higher (See [35] for a formal discussion of IIR of datasets and the claims that can be made on such datasets).

Given this, it may be surprising to note that virtually no TSAD papers make any effort to estimate the IIR of the benchmarks they consider. This would not be an issue if we had good reason to think that the IIR would be very high, but is that the case? Consider the NY-Taxi example mentioned in the previous section. There are supposedly five anomalies `Marathon`, `Thanksgiving`, `Christmas`, `New Year's Day` and `Blizzard`. However, as Wu and Keogh noted, a strong case could be made that there *at least* five more, including `Independence Day`, `Labor Day`, `MLK Day`, `Grand`

Jury, BLM march, Comic Con, Labor Day and Climate March. By visual inspection these unlabeled events seem to be as novel as the five official anomalies. Yet many papers have reported four significant digits when reporting their performance on this dataset.

There is a further layer of subjectivity and uncertainty here. We could argue that Thanksgiving, Christmas, New Year's day etc. are not anomalies, as they can be predicted decades in advance. We would further argue that from the list above, only Grand Jury and BLM march, are true anomalies in the sense that they could not have been predicted at least a day in advance (Note that we had several days of warning for Blizzard).

Note that it *is* possible that a holiday could be an anomaly. The Matrix Profile reports an anomaly on October 13, Columbus Day. Most US states do not celebrate Columbus Day as an official state holiday, and the current authors, who live in California, are completely oblivious to its passing each year. However, New York has a large Italian American population, and they are perhaps the only group to recognize this holiday. No New Yorker would be unaware that traffic would be different on this day, due to parades. This further reinforces the main claim of this chapter, we can only meaningfully define anomalies with reference to the user's knowledge.

4.2 Definitions and Discussion

To keep our discussion generic, we will use *ADA* to refer to any Anomaly Detection Algorithm. In later sections we concretely demonstrate our ideas with particular algorithms. In general, all ADAs work by "sliding" a window across the time series and

giving each subsequence an anomaly score. In some cases, ADA may immediately decide that the anomaly score is high enough to warrant sounding an alarm. In other cases, the ADA may abrogate this responsibility, and simply maintain a list of the top-K anomalies encountered thus far. In this instantiation the assumption is that at any point it may be queried (by an algorithm or person) to report the top-K tentative anomalies it has encountered thus far. If needed, some external test can be performed to decide if top-K *tentative* anomalies warrant labeling as true anomalies.

Definition 10: An OMIT is a subsequence of a time series that will cause an ADA to report and anomaly, but which the user will not consider to be an anomaly.

The reader may respond to this this definition, "*Can't you just make sure that these patterns are in the training data, thus allow the ADA to learn the OMIT pattern?*" However:

- Some OMITS may be due to out-of-band data that the ADA will not have access to. For example, as shown in Fig V, most algorithms will trigger an alarm when they encounter a region that includes the changing of clocks due to daylight saving time (DST). In order to learn this concept, an ADA would need to reason not just about the shapes of the time series but also about the day and the month. However, no ADA algorithms we are aware of even represent or consider such information.
- It may be the case that the OMITS are not learnable by the ADA. For example, consider the low cardinality example shown in Figure X. We are not aware of any ADA algorithm that is sensitive to the intrinsic cardinality of the data and could therefore learn to ignore low cardinality subsequences.

- Even where ‘2’ above is not true, and the ADA could in principle learn the OMIT, asking the ADA to *learn* the OMITS means that the algorithm may do a worse job at learning the important regularities. All ADA models have a finite capacity to learn. We should not tax this limited resource by asking them to learn something that could be trivially hardcoded.
- Even where ‘2’ and ‘3’ above are not true, in at least some cases, it would simply be much easier and faster to simply hardcode the rules. As we shall see, we can typically achieve invariance to OMITS with a single line of intuitive code.

In our discussion below, note that ADA is potentially both a *consumer* of information from FIRE, and a *producer* of information that is written to FIRE, for potential post-mortem analysis.

- FIRE is a categorical vector that accompanies a time series. FIRE contains various codes that annotate and explain the time series data.
- Some of the values in FIRE may be assigned before ADA begins, and some may be assigned by ADA as it processes the data. The values of FIRE that are preassigned are designed to change how the ADA processes the data.
- Some of the values in the FIRE may be assigned by out-of-band information.
- Some of the value’s FIRE may be assigned by the ADA as it processes the data.

4.3 Some of Values in FIRE may be Assigned by Out-of-band Knowledge

It may sometimes be useful to tell ADA to ignore regions of the time series. If ADA sees such a code, it should do nothing, except move on to the next datapoint. The instruction

to ignore could have been placed in FIRE ahead of time, or in real-time. Let us consider some examples of each.

Established Exclusion: Suppose we have an anomaly detector that monitors the production of a batches with a short cycle, say five minutes. While skilled operators can generally operate the machines with metronome-like regularity, there may be unusual telemetry that happens around midnight, 8:00am and 4:00pm, when there is a shift change. The newly arriving worker may change settings slightly to reflect her personal preferences, or abort the last operators unfinished cycle in favor of a calibration run. This example is motivated by a real-world situation in which a plant manager disabled a monitoring system because it gave so many false positives at shift change times. FIRE allows us to simply ignore these using something like: `If MOD(now.minutes,480) < 5, then` $FIRE_i$ `= 'ignore'`. Clearly this is an example of a previously establish exclusion code that can be place in FIRE before ADA is run.

Consider this additional example. The highly studied NAB NY-Taxi dataset has an apparent anomaly with an interesting cause. On one day the demand for taxis appears to suddenly spike, but the real cause is that because of the clock change for daylight saving time, were two hours of taxi demand are mapped to one hour (many papers misattribute this anomaly to the NY Marathon). Again, it would be easy to instruct ADA to ignore such anomalies in perpetuity by placing an '`ignore`' code in FIRE, for a few hours surrounding the clock change.

Real-time Exclusion: Many medical sensors that are attached to the patient skin with sticky pads that are vulnerable to motion artifacts. In most cases these artifacts have no

medical meaning, and do not warrant recording as an anomaly. For example, a motion artifact can be caused by a patient scratching the skin near the electrode or even by sneezing. As recently noted, "*motion artefact are the most difficult type of noise to eliminate because their spectrum usually overlaps with the very important spectral components of the ECG signal, making it difficult for traditional signal processing techniques to separate.*" [68]. Because of the ubiquity of motion artifacts, the majority of anomalies detected in such data streams are uninteresting motion "anomalies", and this may mask the handful of potentially interesting medical anomalies. Fortunately, there is a simple solution. Many medical sensors now also include an accelerometer. If we observe a significant change in acceleration, we can simply write an "ignore" code to the next five seconds of FIRE. We may desire to be more specific and use a code that explains why the region of time was ignored, something like: `If acceleration > 1.0ms`^{2} `then` $\texttt{FIRE}_{\texttt{[i:i+5seconds]}}$ `='ignore-due-to-motion'.`

4.4 Case Studies

Here we show some case studies, beginning with an example of *intrinsic* annotation of FIRE.

Recall the APB-data example in shown in Fig. 43.*middle*. We noted that the bizarre shape was *not* an anomaly, but merely a disconnection artifact. In Fig. 44 we show how FIRE can be programed to communicate to the ADA to ignore such regions.

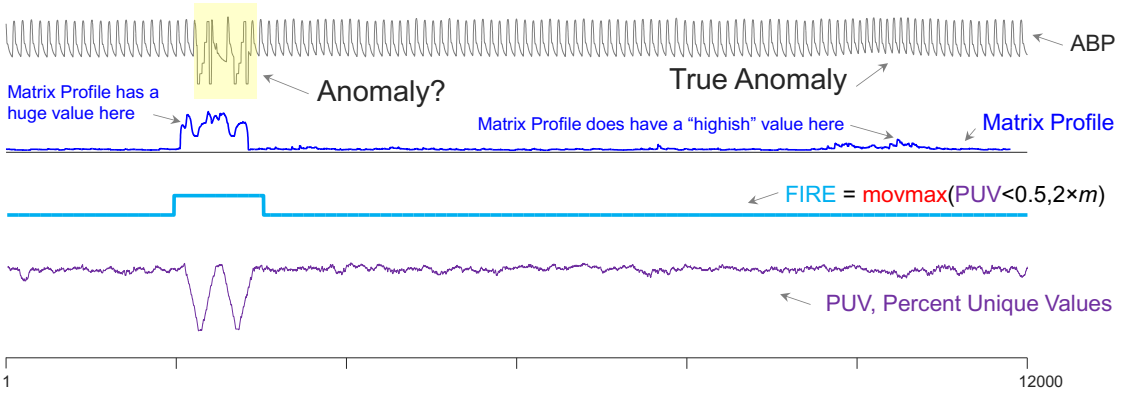

Fig. 44 *top* to *bottom*) The ABP time series shown in Fig. 43. The Matrix Profile ADA strongly peaks at the highlighted disconnection artifact. We can use FIRE to suppress the MP's selection of this region as the top anomaly. FIRE is programed to ignore regions around low cardinality subsequences.

Note that in this case suppressing the discovery of the disconnection artifact allows us to find a subtle *true* anomaly, the transient rise at about 10,000. This are often called "sighing" by the attending physician, and depending on the medical context, may be worth investigating.

In our next example, we consider an example of *extrinsic* annotation of FIRE. In Fig. 45 we revisit the ECG-data example shown in Fig. 43.*top*.

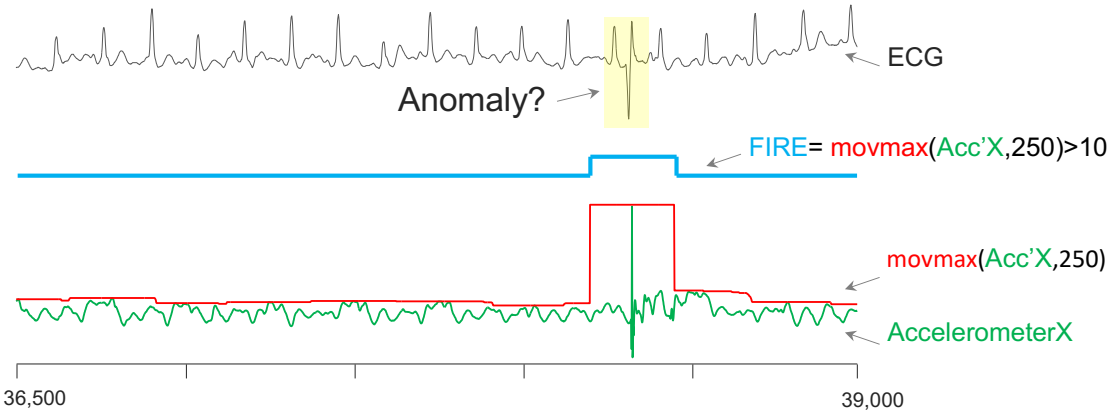

Fig. 45 *top* to *bottom*) The ECG time series shown in Fig. 43. The apparent anomaly here is simply caused by the sensor receiving a 'bump'. Because we have an accelerometer on the ECG sensor, we can suppress any apparent anomalies that happen during sudden movements of the sensor.

We investigated the PSML dataset [71], which collects hourly load data from 2018 to 2020 for 66 representative load zones in the U.S. electricity market. In our initial survey, we noted two OMITS in ERCOT Zone 4, which covers Northern Texas.

- Dropouts.
- Linear regions interpolation artifacts.

Fig. 46 shows two examples of each.

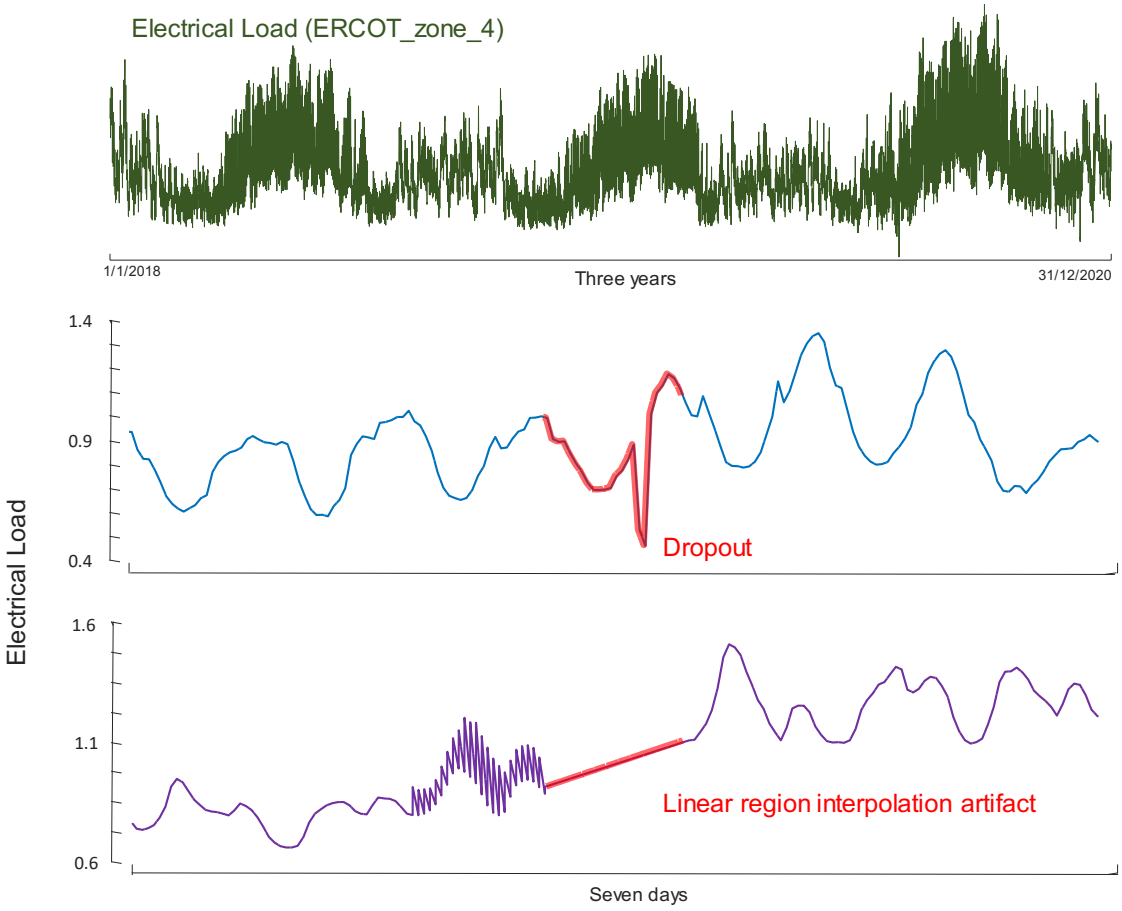

Fig. 46 *top* to *bottom*) Three years of electrical load data for Northern Texas from [71]. The first OMIT occurs on May 12, 2020 and is a dropout. The second OMIT occurs on December 12, 2020 and is a linear region caused by linear interpolation.

Thus, before investigating other regions, we created the FIRE vector in Table 10.

Table 10: Pseudo code snippet for creating FIRE vectors to suppress OMITS in PSML dataset

```
// Generate FIRE vector to suppress dropouts
FIRE = movmax((abs(diff(T,2))>0.02),2*m)
// Generate FIRE vector to suppress linear regions
For i = 1 to length(T) - m + 1 // Scan all subsequences
      linear_region_ratio = max(diff(find(vertcat(1,…
      diff(round(diff(zscore(T_{i:i+m-1})),3)),1))))/m
      If linear_section_ratio > 0.4
            FIRE_i = 1
```

To see what difference this makes, we searched CAISO Zone 2, which covers Northern California, both with and without FIRE.

For an m = 36-hour search, eight out of the top ten reported anomalies are false positives caused by interpolation artifacts (3) or dropouts (5). However, if we rerun with FIRE, then ten out of the top ten anomalies appear to be true positives. For example, consider Fig. 47.

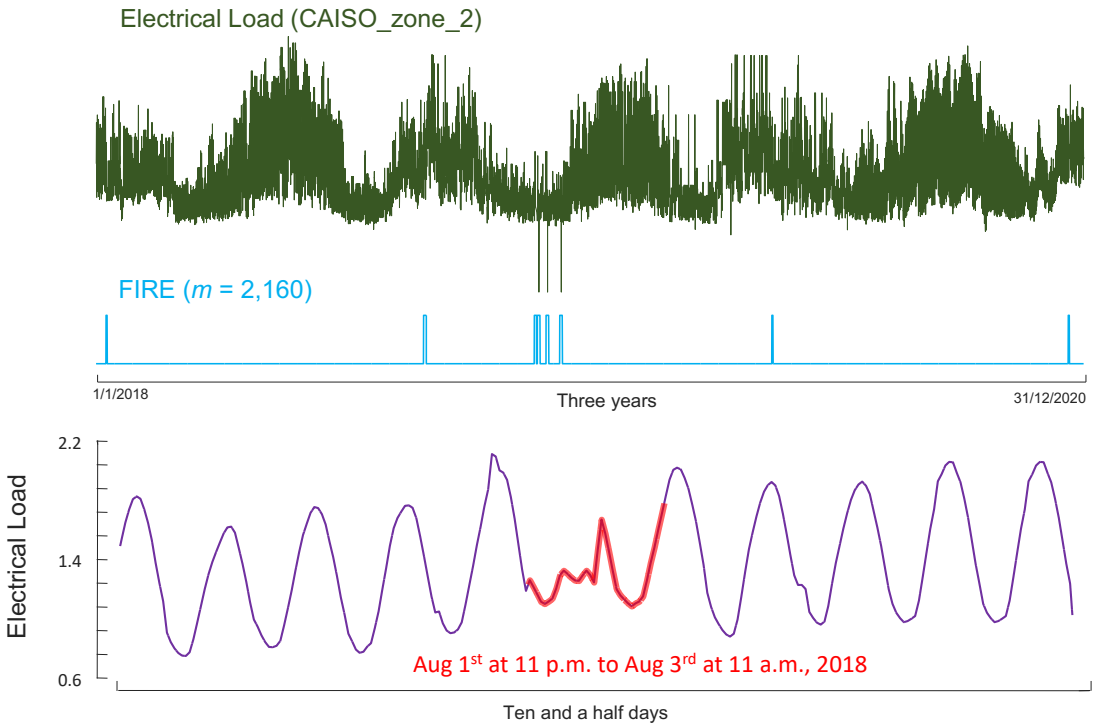

Fig. 47 *top* to *bottom*) Three years of electrical load data for Northern California from [71]. The FIRE vector generated using Table 10's algorithm. One of the top-10 discords reported by FIREDAMP in this dataset. Here the window size is set to 2,160 (one and a half days in minutes).

Based on the date, this anomaly appears to be caused by the Donnell Fire [69]. This was a wildfire that started on August 1, 2018, due to an unattended illegal campfire, near Donnell Reservoir, burning around California State Route 108 in Tuolumne County, California. More than 100 buildings were destroyed, and 9 civilians were injured. The fire cut brought down powerlines, reducing what otherwise would have been a high-demand day for air conditioning.

We also considered an m = 48-hour search. This time six out of the top ten reported anomalies are false positives caused by interpolation artifacts (2) or dropouts (4). Once again, using FIRE, ten out of the top ten anomalies appear to be true positives. For example, consider Fig. 48.

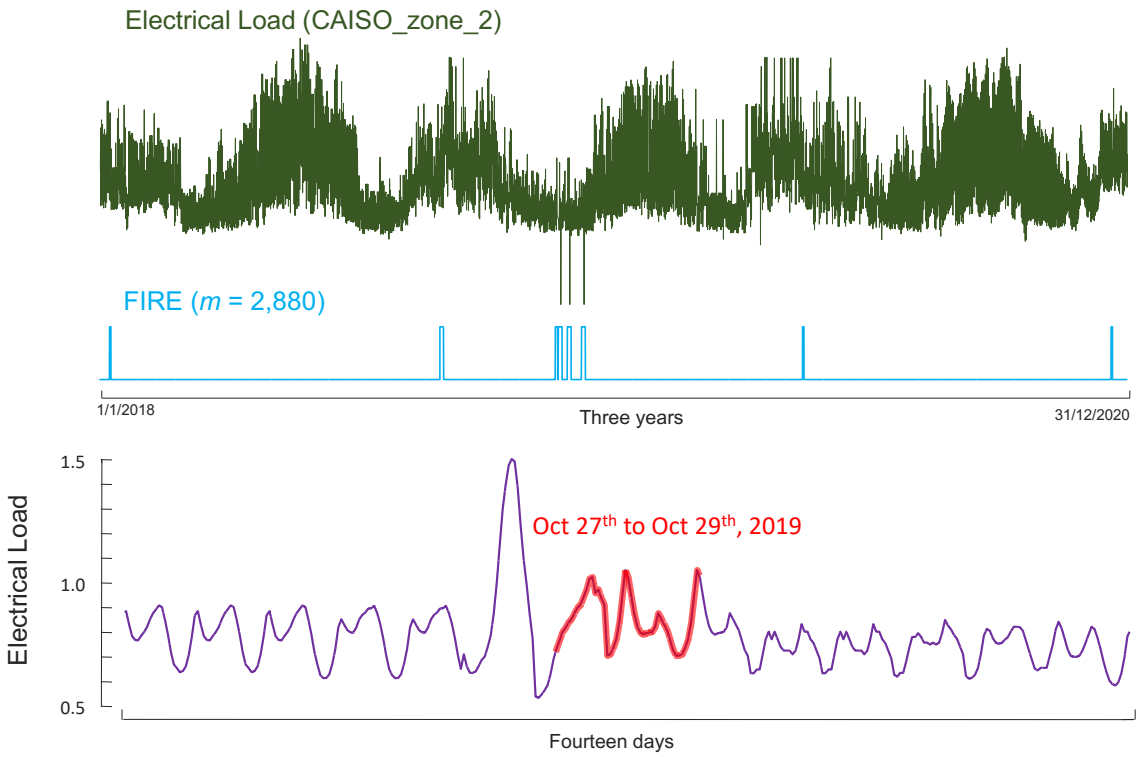

Fig. 48 *top* to *bottom*) Three years of electrical load data for Northern California from [71]. The FIRE vector generated using Table 10's algorithm. One of the top-10 discords reported by FIREDAMP in this dataset. Here the window size is set to 2,160 (one and a half days in minutes).

Based on the date and location, this anomaly appears to be caused by a preemptive move by Pacific Gas and Electric (PG&E). The previous year, The Camp Fire was the deadliest and most destructive wildfire in California's history, and the most expensive natural disaster in the world in 2018 in terms of insured losses. At least eight-five people lost their lives. The cause of the fire was traced to a PG&E power transmission line that was downed by high winds. Given that background, it is unsurprising that when a year later on October 26th 2019, when a severe windstorm was expected in Northern California preemptively cut power to almost one million people, explaining "*Public Safety Power Shutoff (PSPS) events in order to mitigate catastrophic wildfire risk presented by*

significant offshore wind events combined with low humidity levels and critically dry fuels" [70].

That appears to explain the highlighted anomaly, but not that on the day before the anomaly there is a *peak* in demand. Here we can only speculate. The Donnell Fire anomaly in Fig. 47 was an unexpected event. In contrast, the people affected by this event were given 24 to 48 hours' notice. This may reflect a change in behavior of individuals expecting a power cut, doing laundry or charging their automobiles while they could.

5. Conclusion

In this thesis, we introduced three innovative algorithms to address the challenges of speed and accuracy in TSAD tasks. First, we presented DAMP, a novel algorithm that computes left-discords on fast-arriving data streams, significantly improving the scalability of the algorithm up to trillions of data points. Next, we proposed MADRID, a Hyper-Anytime algorithm designed to tackle the issue of dependency on subsequence length for detecting anomalies and swiftly identify multi-scale anomalies within time series. Lastly, we established FIRE, a user-centric framework for encapsulating users' requirements and communicating to the algorithms, thereby making anomaly detection algorithms faster, more accurate, and more applicable.

In Chapter 2, we created the left-discord anomaly detection framework, generalizing classic time series discords that previously only handled the batch case, to the online and *effectively online* case, and solving the twin-freak problem in the process. Further, we have introduced DAMP, a fast and scalable algorithm to discover such discords. Experimental results have demonstrated that our proposed left-discords outperform the current SOTA methods, including the most cited deep learning methods in terms of accuracy. Moreover, we have further demonstrated that DAMP is orders of magnitude faster and more scalable than any method in the literature. In future work, we plan to address the limitations of DAMP. For example, DAMP uses the Z-normalized Euclidean distance, but you cannot Z-normalize constant regions of the time series (as you get a divide-by-zero error). Another type of anomaly that DAMP cannot detect is a sudden *decrease* in the noise level of a time series, as smooth time series tend to have a relatively

low distance from all other time series. As of now, we can catch these two special cases with ad-hoc rules, but a more principled and elegant solution is desirable.

In Chapter 3, we have shown that the results of TSAD algorithms depend more strongly on the length of the subsequences considered than the community seems to appreciate. We further show that we can completely bypass the issue by testing all lengths. The expressiveness of all-lengths search does not come at the expense of tractability. Therefore, we propose MADRID, which reduces the absolute time to compute all-discords by employing a novel computation ordering strategy and allows real-time interaction with data. We have shown that MADRID can test hundreds of values for m, at least an order of magnitude faster than deep learning models can test a single length, and thus produce more accurate results. In the future, we intend to enhance the user experience of MADRID. One direction involves further adapting MADRID to become a contract algorithm, which affords users the flexibility to plan their computational resources based on their availability of time, memory, and other factors.

In Chapter 4, we propose a novel paradigm for anomaly detection wherein algorithms leverage FIRE, a vector for annotating and interpreting time series data, to gain insights into user knowledge and requirements in order to make more accurate decisions. We used numerous real-world examples to demonstrate that FIRE improves the accuracy and efficiency of anomaly detection algorithms. In the future, we will expand FIRE to include more categorical information, providing enriched context for anomaly detection tasks. For instance, categorical FIRE vectors can be used in integration learning by assigning weights to multiple anomaly detection algorithms. Moreover, another possibility is to

employ FIRE to dynamically adjust algorithmic thresholds based on domain or out-of-band information. An example would be recalibrating anomaly detection thresholds in a factory setting when new processes or equipment alter the standard machine behavior.

We believe that the throughput and scalability of DAMP will allow the community to address datasets and applications that are currently out of reach, opening up numerous new possibilities for research. In this thesis, MADRID and FIRE serve as excellent examples of extensions to DAMP, their superior performance attributable to utilizing DAMP as a foundation. We believe that there are further potentials to be unearthed in the future. Finally, to encourage the community to adopt and expand upon our work, we have made all code and data available at [10][51].

6. Bibliography

[1] Aubet F-X, Zügner D, Gasthaus J (2021) Monte Carlo EM for Deep Time Series Anomaly Detection. arXiv:211214436 [cs, stat]

[2] Audibert J, Marti S, Guyard F, Zuluaga MA (2021) From Univariate to Multivariate Time Series Anomaly Detection with Non-Local Information. In: Lemaire V, Malinowski S, Bagnall A, et al. (eds) Advanced Analytics and Learning on Temporal Data. Springer International Publishing, Cham, pp 186–194

[3] Batista GEAPA, Keogh EJ, Tataw OM, de Souza VMA (2014) CID: an efficient complexity-invariant distance for time series. Data Min Knowl Disc 28:634–669. https://doi.org/10.1007/s10618-013-0312-3

[4] Boniol P, Linardi M, Roncallo F, et al (2021) Unsupervised and scalable subsequence anomaly detection in large data series. The VLDB Journal 30:909–931. https://doi.org/10.1007/s00778-021-00655-8

[5] Boniol P, Paparrizos J, Palpanas T, Franklin MJ (2021) SAND: streaming subsequence anomaly detection. Proceedings of the VLDB Endowment 14:1717–1729

[6] Bu Y, Chen L, Fu AW-C, Liu D (2009) Efficient anomaly monitoring over moving object trajectory streams. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09. ACM Press, Paris, France, p 159

[7] Case Western Reserve University Bearing Data Center. [Online]. Available: https://csegroups.case.edu/ bearingdatacenter/home. Accessed: Nov. 15, 2021.

[8] CNC Crashes. Video. (15 Feb 2018). from https://youtu.be/t2tBtZCa7j4?t=205. Retrieved December 20, 2021.

[9] Daigavane A, Wagstaff KL, Doran G, et al (2022) Unsupervised detection of Saturn magnetic field boundary crossings from plasma spectrometer data. Computers & Geosciences 161:105040

[10] DAMP (2022) https://sites.google.com/view/discord-aware-matrix-profile

[11] Dau HA, Bagnall A, Kamgar K, et al (2019) The UCR time series archive. IEEE/CAA J Autom Sinica 6:1293–1305. https://doi.org/10.1109/JAS.2019.1911747

[12] Doshi K, Abudalou S, Yilmaz Y (2022) TiSAT: Time Series Anomaly Transformer. arXiv:220305167 [cs, eess, stat]

[13] Higham NJ (2002) Accuracy and Stability of Numerical Algorithms (2 ed). ISBN: 978-0-89871-521-7

[14] Hundman K, Constantinou V, Laporte C, et al (2018) Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, London United Kingdom, pp 387–395

[15] Imani S, Madrid F, Ding W, et al (2020) Introducing time series snippets: a new primitive for summarizing long time series. Data Min Knowl Disc 34:1713–1743. https://doi.org/10.1007/s10618-020-00702-y

[16] Keogh E (2021) Irrational Exuberance Why we should not believe 95% of papers on Time Series Anomaly Detection. 7th SIGKDD Workshop on Mining and Learning from Time Series at SIGKDD 2021. Workshop Keynote https://www.youtube.com/watch?v=Vg1p3DouX8w&t=324s

[17] Khansa HE, Gervet C and Brouillet A (2012) Prominent Discord Discovery with Matrix Profile : Application to Climate Data Insight. 10th International Conference of Advanced Computer Science & Information Technology (ACSIT 2022) May 21~22, 2022, Zurich, Switzerland

[18] Kirti R, Karadi R (2012) Cardiac tamponade: atypical presentations after cardiac surgery. Acute Medicine 11:93–96

[19] Mueen A, Zhu Y, Yeh M, et al (2017) The fastest similarity search algorithm for time series subsequences under euclidean distance. url: www cs unm edu/~ mueen/FastestSimilaritySearch html. Accessed 24 Janurary, 2022

[20] Murray D, Liao J, Stankovic L, et al A data management platform for personalised real-time energy feedback.

[21] Nakamura T, Imamura M, Mercer R, Keogh E (2020) MERLIN: Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives. In: 2020 IEEE International Conference on Data Mining (ICDM). IEEE, Sorrento, Italy, pp 1190–1195

[22] National Weather Service. January 24, 2019 Heavy Rain and Flooding. from https://www.weather.gov/aly/24Jan19HeavyRainFlood. Retrieved May 1 2022.

[23] Neupane D, Seok J (2020) Bearing Fault Detection and Diagnosis Using Case Western Reserve University Dataset With Deep Learning Approaches: A Review. IEEE Access 8:93155–93178. https://doi.org/10.1109/ACCESS.2020.2990528

[24] Nilsson F (2022) Joint Human-Machine Exploration of Industrial Time Series Using the Matrix Profile. Halmstad University, School of Information Technology, Halmstad Embedded and Intelligent Systems Research (EIS), CAISR - Center for Applied Intelligent Systems Research.

[25] Palpanas T. Personal communication June 4th 2022.

[26] Paparrizos J, Kang Y, Boniol P, et al (2022) TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection. Proceedings of the VLDB Endowment (PVLDB) Journal

[27] Park D, Hoshi Y, Kemp CC (2018) A Multimodal Anomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder. IEEE Robot Autom Lett 3:1544–1551. https://doi.org/10.1109/LRA.2018.2801475

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

[29] Silive.com. Wild storm pelts Staten Island with giant hail -- 'threat of tornado has passed' from https://www.silive.com/news/2019/05/nws-issues-tornado-warning-for-staten-island.html. Retrieved May 1 2022.

[30] Su Y, Zhao Y, Niu C, et al (2019) Robust anomaly detection for multivariate time series through stochastic recurrent neural network. pp 2828–2837

[31] Thill M, Konen W, Bäck T (2020) Time series encodings with temporal convolutional networks. Springer, pp 161–173

[32] Truong HT, Ta BP, Le QA, et al (2022) Light-weight federated learning-based anomaly detection for time-series data in industrial control systems. Computers in Industry 140:103692. https://doi.org/10.1016/j.compind.2022.103692

[33] Wastewater News. Valentine's Day Storm Slams California, Pushing Water Agencies to the Edge. from www.news.cornell.edu/Chronicle/00/5.18.00/wireless_class.html. Retrieved Dec 1 2021.

[34] Wikipedia. Leap year problem. from https://en.wikipedia.org/wiki/Leap_year_problem. Retrieved December 1, 2021.

[35] Wu R, Keogh E (2021) Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. IEEE Trans Knowl Data Eng 1–1. https://doi.org/10.1109/TKDE.2021.3112126

[36] Yeh C-CM, Zheng Y, Wang J, et al (2021) Error-bounded Approximate Time Series Joins using Compact Dictionary Representations of Time Series. CoRR abs/2112.12965 (2021)

[37] Yeh C-CM, Zhu Y, Dau HA, et al (2019) Online amnestic dtw to allow real-time golden batch monitoring. pp 2604–2612

[38] Zheng X, Xu N, Trinh L, et al (2021) PSML: A Multi-scale Time-series Dataset for Machine Learning in Decarbonized Energy Grids. arXiv preprint arXiv:211006324

[39] Zhu Y, Yeh C-CM, Zimmerman Z, et al (2018) Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. IEEE, pp 837–846

[40] "2011 AFL Parade," The Age, 30 Sep. 2010. [Online]. Available: www.theage.com.au/sport/afl/2011-afl-parade-20110930-1l0un.html.

[41] "Remembrance Day," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Remembrance_Day.

[42] “Winter Storm Kai Spreads Snow” Weather.com, Feb. 04, 2019. [Online]. Available: weather.com/safety/winter/news/2019-02-01-winter-storm-kai-snow-wind-sierra-plains-midwest-new-england.

[43] A. Ermshaus, P. Schäfer, and U. Leser, “Window Size Selection in Unsupervised Time Series Analytics: A Review and Benchmark,” in AALTD, 2023, pp. 83–101.

[44] Mueen et al., “The fastest similarity search algorithm for time series subsequences under euclidean distance,” url: www. cs. unm. edu/~ mueen/FastestSimilaritySearch. html (accessed 24 May, 2016), 2017.

[45] Shulman, "Snow falling in some parts of Redding," Redding, 2019. [Online]. Available: www.redding.com/story/news/local/2019/02/12/snow-falling-some-parts-redding/2854564002

[46] Barz, et al. “Maximally divergent intervals for extreme weather event detection,” in OCEANS 2017-Aberdeen, IEEE, 2017, pp. 1–9.

[47] Turkay, E. Kaya, S. Balcisoy, and H. Hauser, “Designing progressive and interactive analytics processes for high-dimensional data analysis,” IEEE TVCG vol. 23, no. 1, pp. 131–140, 2016.

[48] Keogh, D. R. Taposh, U. Naik, and A. Agrawal, “Multi-dataset time-series anomaly detection competition,” presented at the 2021 ACM SIGKDD. www.cs.ucr.edu/~eamonn/time_series_data_2018/UCRArchive_2018.zip

[49] Keogh, J. Lin, and A. Fu, “Hot sax: Efficiently finding the most unusual time series subsequence,” in Fifth IEEE International Conference on Data Mining (ICDM’05), Ieee, 2005, p. 8 pp.

[50] H. Fry and R.-G. Lin, "Storm slams into Northern California with heavy snow and rain, record low pressure," LA Times, Nov. 26, 2019. [Online]. Available: www.latimes.com/california/story/2019-11-26/storm-northern-california-heavy-snow-rain.

[51] https://sites.google.com/view/madrid-icdm-23

[52] Hundman, K., et al. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. (2018), SIGKDD, 387-95.

[53] inIT-OWL, "High Storage System Data for Energy Optimization," Kaggle, 2017. [Online]. Available: www.kaggle.com/datasets/inIT-OWL/high-storage-system-data-for-energy-optimization

[54] M. Zymbler and Y. Kraeva, “High-performance Time Series Anomaly Discovery on Graphics Processors,” arXiv:2304.01660, 2023.

[55] N. L. Kerr, “HARKing: Hypothesizing after the results are known,” Personality and social psychology review, vol. 2- 3, pp. 196–217, 1998.

[56] N. Parkin and Staff, "Soaked Victoria warned of flash flooding," ABC News, 12 Jan. 2011. [Online]. Available: www.abc.net.au/news/2011-01-12/soaked-victoria-warned-of-flash-flooding/1903164.

[57] National Weather Service, “Historic Fire Weather Conditions during October 2019,” U.S. Department of Commerce,2023. [Online]. Available: www.weather.gov/mtr/FireWeatherOctober2019.

[58] OneShotSTL: One-Shot Seasonal-Trend Decomposition For Online Time Series Anomaly Detection And Forecasting. To appear in VLDB.

[59] R. Wu and E. Keogh, “Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress,” IEEE Transactions on Knowledge and Data Engineering, 2021.

[60] S. Imani and E. Keogh, Multi-window-finder: Domain agnostic window size for time series data. MileTS, 2021.

[61] S. Schmidl, P. Wenig, and T. Papenbrock, “Anomaly detection in time series: a comprehensive evaluation,” Proceedings of the VLDB Endowment, vol. 15, no. 9, pp. 1779–1797, 2022.

[62] S. Schmidl, P. Wenig, and T. Papenbrock, “HYPEX: Hyperparameter Optimization in Time Series Anomaly Detection,” BTW 2023, 2023.

[63] S. Zilberstein & S. Russell, “Approximate reasoning using anytime algorithms” Imprecise and approximate computation, pp. 43–62, 1995.

[64] X. Zheng et al., “A multi-scale time-series dataset with benchmark for machine learning in decarbonized energy grids,” Scientific Data, vol. 9, no. 1, p. 359, 2022.

[65] Y. Lu, et al, “DAMP: accurate time series anomaly detection on trillions of datapoints and ultra-fast arriving data streams,” Data Mining and Knowledge Discovery, pp. 1–43, 2023.

[66] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, “Robust anomaly detection for multivariate time series through stochastic recurrent neural network,” in ACM SIGKDD 2019, pp. 2828–2837.

[67] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh, “Matrix Profile XI: SCRIMP++: time series motif discovery at interactive speeds” ICDM, 2018, pp. 837–846.

[68] X. An and G. K. Stylios, “Comparison of Motion Artefact Reduction Methods and the Implementation of Adaptive Motion Artefact Reduction in Wearable Electrocardiogram Monitoring,” Sensors, vol. 20, no. 5, p. 1468, Mar. 2020, doi: 10.3390/s20051468.

[69] https://en.wikipedia.org/wiki/Donnell_Fire.

[70] M. E. Allen, “PG&E Public Safety Power Shutoff (PSPS) Report to the CPUC October 26 & 29, 2019 De-Energization Event”, 2019.

[71] X. Zheng et al., “A Multi-scale Time-series Dataset with Benchmark for Machine Learning in Decarbonized Energy Grids.” arXiv, May 22, 2022. Accessed: Sep. 27, 2022. [Online]. Available: http://arxiv.org/abs/2110.06324.