Matrix Profile XXX: MADRID: A Hyper-Anytime and Parameter-Free Algorithm to Find Time Series Anomalies of all Lengths

Yue Lu
University of California, Riverside
Riverside, USA
ylu175@ucr.edu

Thirumalai Vinjamoor Akhil Srinivas
University of California, Riverside
Riverside, USA
tsrin002@ucr.edu

Takaaki Nakamura
Mitsubishi Electric Corporation
Kamakura, Japan
Nakamura.Takaaki@dy.MitsubishiElectric.co.jp

Makoto Imamura
Tokai University
Tokyo, Japan
imamura@tsc.u-tokai.ac.jp

Eamonn Keogh
University of California, Riverside
Riverside, USA
eamonn@cs.ucr.edu

***Abstract*—In recent years there has been increasing evidence that one of the simplest time series anomaly detection methods, time series *discords*, remains one of the most effective methods. However, time series discords have one notable issue; the anomalies discovered depend on the algorithm's only input parameter, the subsequence length. The obvious way to bypass this issue is to find anomalies at every possible length, however this seems to be untenably slow. In this work we introduce MADRID, an algorithm to efficiently solve the all-discords problem. We show that we can reduce the absolute time to compute all-discords, and that by using a novel computation ordering strategy, MADRID is a Hyper-Anytime Algorithm. We will formally define this term later, but this refers to an anytime algorithm that converges exceptionally fast. In practice this means that for most real-world analytical tasks, the user can interact with their data in real-time. The ability to *compute* anomalies of all lengths produces the issue of *ranking* anomalies of different lengths. We further introduce novel algorithms for this task. We demonstrate the utility of MADRID in various domains and show that it allows us to find anomalies that would otherwise escape our attention.**

Keywords—Time series, anomaly detection, anytime algorithms

I. INTRODUCTION

There is increasing evidence that, across a wide range of tasks, one of the simplest time series anomaly detection methods remains among the most effective methods. 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 [10], Matrix Profile [26], SCRIMP [28], DAMP [26], 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.

Moreover, time series discords recently have had their performance improved significantly by only considering nearest neighbors to the *left* of the subsequence being examined [26]. This makes discords invariant to the so-called *twin-freak* problem allowing them to achieve state-of-the-art performance on several benchmark datasets including the Hexagon ML/UCR Anomaly Detection datasets [9][26].

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 [4] notes: "*setting the right window size is crucial…failure to provide an optimal window size may incur monetary damage*."

To illustrate the issue, consider Figure 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 [26] in a loop and tested all discord lengths from eight hours to twenty-four hours.

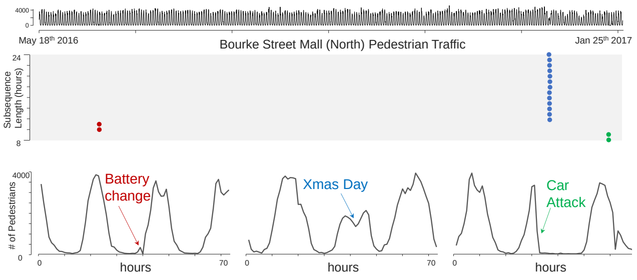


Figure 1: *top*) An eight-month long time series representing pedestrian traffic in a street in Melbourne. *center*) The location of the top-1 discord, for subsequence lengths (*m*) from eight to twenty-four hours. *bottom*) A zoom-in of the three discovered anomalies.

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 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 this work 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. As noted in Figure 2, the name MADRID is both an acronym and a tribute to the inventor of the Pan Matrix Profile, the late Frank Madrid.

Figure 2: This paper is dedicated to our dear friend, Frank Madrid, July 1986 – May 2020. Frank invented the *Pan Matrix Profile* and was working on scaling it up when he passed away. This present work is a poor shadow of what Frank would have achieved.

We then demonstrate that we can cast the search as an *anytime algorithm* [24]. Previous work has shown that some discord discovery algorithms, most noticeably the Matrix Profile, are amenable to being cast as anytime algorithms [28]. 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 paper is organized as follows. In Section II we review our motivation and state our assumptions. Section III introduces all needed definitions and notation, allowing us to introduce MADRID in Section IV, which in turn allows us to offer a discussion of the implications of all-lengths anomaly search in Section V. We empirically evaluate our work in Section VI. We omit a separate related works section, instead we discuss such material inline and in context.

II. Motivation and Assumptions

There has been a recent explosion of research efforts in Time Series Anomaly Detection (TSAD) [4][15][19][23][26]. 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 [26]. 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 Figure 1. Moreover, this has been empirically observed by several independent research groups [4][7][15][21], and we demonstrate it in Figure 3 and elsewhere in this paper.
- 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*" [4].
- 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 [7] 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 Figure 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.

A. 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 Figure 3 we illustrate both cases by using the Matrix Profile to find anomalies for consecutive values of m.

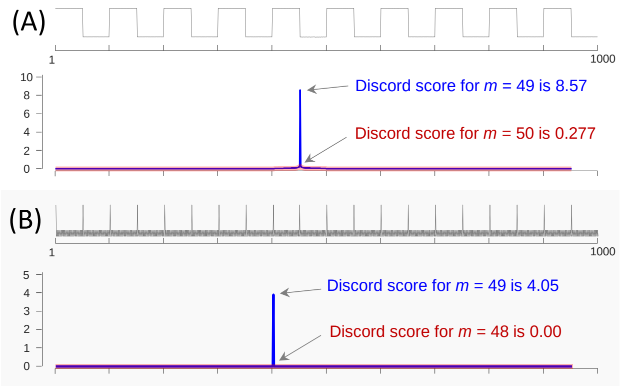


Figure 3: Two examples to show that anomaly detection can be hypersensitive to the sliding window length. In (A) we can correctly find a subtle anomaly when $m = 49$ but increasing m by just one causes the discord score to plunge. In (B) we can correctly find a subtle anomaly when $m = 49$ but decreasing 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 Figure 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.

B. *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 [26], thus processing perhaps a thousand subsequence lengths appears to be an insurmountable task for realistic deployments. In this work 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.

III. Definitions and Notations

We begin by introducing the necessary definitions and notation.

Definition 1: A *time series* T is defined as a sequence of real numbers t_i, represented as $T = [t_1, t_2, \ldots, t_n]$, where n denotes the length of T.

As we hinted at above, TSAD does not consider global properties of a time series, instead we focus on local *subsequences* within the time series.

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

By taking any subsequence $T_{i,m}$ as a query, computing its distance from all subsequences within the time series T and saving the distances in an array sequentially, we can obtain a *distance profile*.

Definition 3: *Distance profile* D_i of time series T is an ordered array of Euclidean distances between the query subsequence $T_{i,m}$ and all subsequences in time series T. Formally, $D_i = d_{i,1}, d_{i,2}, \ldots, d_{i,n-m+1}$, with $d_{i,j}$ $(1 \leq i,j \leq n-m+1)$ representing the Euclidean distance between $T_{i,m}$ and $T_{j,m}$.

In distance profile D_i for query $T_{i,m}$, the i^{th} position signifies the distance between the query and itself, resulting in a value of 0. The values preceding and following position i are close to zero, as the corresponding subsequences overlap with the query. We disregard these matches of the query and itself [28], concentrating instead on *non-self match*.

Definition 4: *Non-Self Match*: In a time series T with a subsequence $T_{p,m}$ of length m beginning 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 $|p - q| \geq m$.

Using the concept of non-self match, we can now define *time series discords*.

Definition 5: *Time series discord*: In time series T, the subsequence $T_{d,m}$ of length m at position d is considered a discord of T if the distance between $T_{d,m}$ and its nearest non-self match is largest. In other words, $\forall$ subsequences $T_{c,m}$ of T, non-self matching set M_D of $T_{d,m}$, and non-self matching set M_C of $T_{c,m}$, $min(d_{d,M_D}) > min(d_{c,M_C})$.

One way to represent time series discords is to compute the Matrix Profile (MP) [28].

Definition 6: A *Matrix Profile P* of a time series T is a vector that records 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 $(1 \leq i \leq n - m + 1)$ represents the distance profile of query $T_{i,m}$ in time series T.

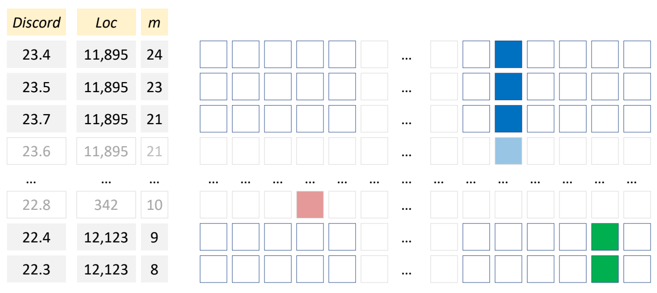
It is obvious that the *maximum* value of the MP is the time series discord. Note that the MP contains additional information, for example about *motifs*, that do not interest us for the task of TSAD. As explained below, a special version of the MP called the *left Matrix Profile* (*left*-MP), can be computed by only examining the past, i.e. *before* the current time series subsequence. This variant was introduced in [26], but reviewed here for completeness.

Definition 7: A *left Matrix Profile*, denoted as P^L, for a time series T is a vector that maintains the z-normalized Euclidean distance between each subsequence and the nearest non-self match preceding it. Formally, for 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 stores only the distance between the query subsequence and all subsequences occurring prior to the query, then we have $P^L = [min(D_1^L), min(D_2^L), \ldots, min(D_{n-m+1}^L)]$.

There are two reasons why the *left*-MP is superior to the classic MP. First, it is defined for the *online* case, whereas the classic MP is only defined for the batch case, i.e., the post-mortem examination of data. Secondly, the classic MP will fail to find an anomaly if there are two or more occurrences of it in the data, the so-called *twin-freak* problem [26]. In remainder of this paper, unless otherwise stated, the term *discord* refers to the highest value on the left Matrix Profile P^L, i.e., *left*-discord.

Definition 8: 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 $M = [P_{minL}^L;\ P_{minL+S}^L,\ P_{minL+2S}^L;\ \ldots;\ P_{maxL}^L\ ]$.

In Figure 4 we illustrate a multi-length discord table M.



Discord	Loc	m
23.4	11,895	24
23.5	11,895	23
23.7	11,895	21
23.6	11,895	21
...	...	...
22.8	342	10
22.4	12,123	9
22.3	12,123	8

Figure 4: An illustration of a multi-length discord table M. To ground it in a familiar setting, it is loosely based on Figure 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].

A. *A Brief Review of DAMP*

MADRID exploits some ideas from the recently introduced DAMP algorithm [26], 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 [5], 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 [24].

B. *Measuring Anytime Algorithm Efficiency*

In the title of this work, 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 [24]. 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 Figure 5, 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 Figure 5. As we will show in Section IV.*A*, we will propose a variety of algorithms to solve the task-at-hand, culminating with MADRID, which is a *Hyper*-Anytime Algorithm.

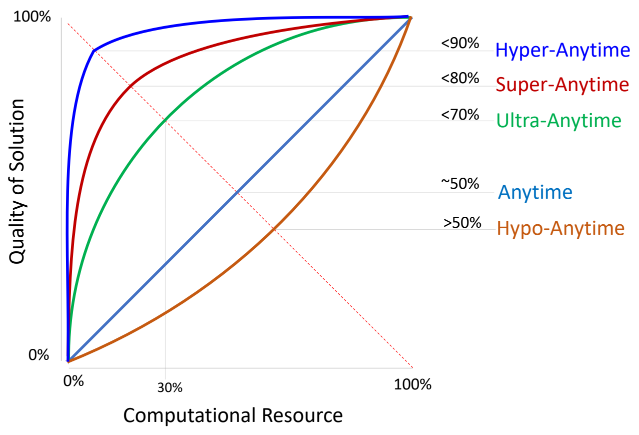


Figure 5: An illustration of the performance of five hypothetical anytime algorithms. The point where they intersect the red dashed line allows us to rank them with the proposed hierarchy.

IV. MADRID

We are now in a position to introduce MADRID. The reader will recall the *completed* multi-length discord table *M* shown in Figure 4. In Figure 6 we show what this data structure looks like when it is initialized.

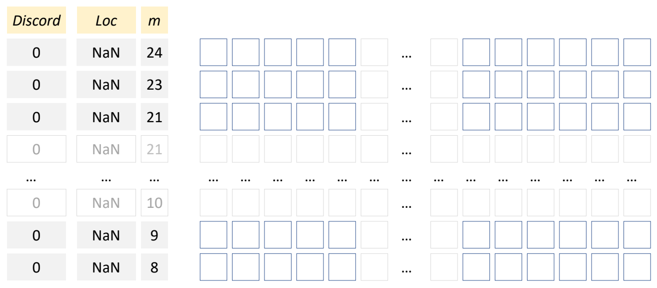


Figure 6: An initialized multi-length discord table *M*. Compare to Figure 4. Here the user chose *minL* = 8, *maxL* = 24 and *S* = 1.

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.

A. Setting a Strong Baseline for Efficiency

As shown in Figure 7, 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 Figure 7.*top* it takes 15.5 hours to finish.

The MASS Brute Force was introduced as a baseline in [26]. It also searches across all historical data, however it avails of MASS [5], which is a highly optimized subsequence search algorithm. As Figure 7 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 III.*A*, significantly reducing the search space and thereby greatly accelerating the process. As Figure 7, 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. Figure 7 shows that it gives a further 1.5 times speed-up.

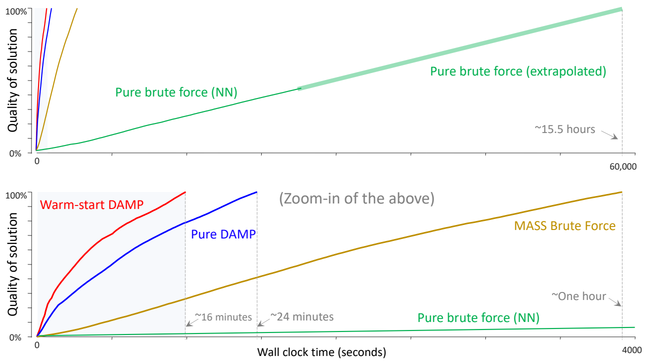


Figure 7: *top*) The performance of four algorithms tasked with computing *M* for a time series of length 8,192, with *minL* = 128, *maxL* = 768, and step size *S* = 1. As pure brute force visually dominates, in (*bottom*) we show a zoom-in on the other three approaches.

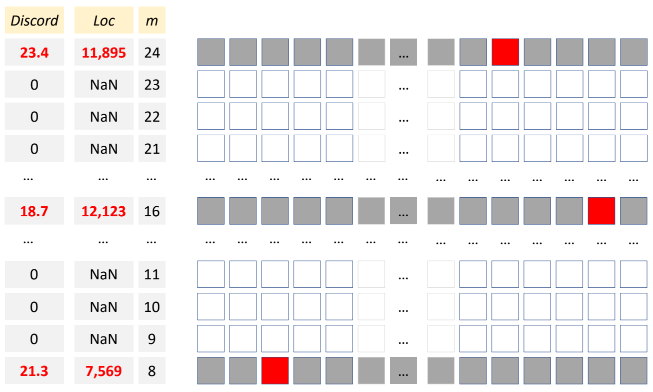
For the rest of this paper, we will only show results corresponding to the gray shaded area in Figure 7, having established it as our strong baseline. In addition, in future figures in order to factor out the influence of the hardware being used, we will plot not the absolute time, but the percentage 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 [24] 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.

Figure 8 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 *Loc*s. 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 Figure 8, 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.

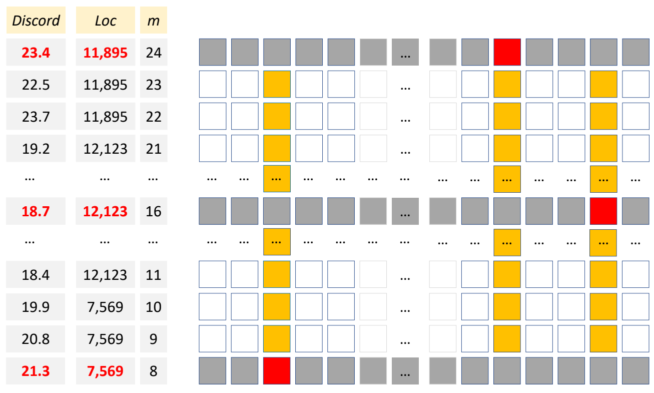


Discord	*Loc*	*m*
23.4	11,895	24
0	NaN	23
0	NaN	22
0	NaN	21
...	...	...
18.7	12,123	16
...	...	...
0	NaN	11
0	NaN	10
0	NaN	9
21.3	7,569	8

Figure 8: MADRID's multi-length discord table *M* after the first step of initialization. As a follow-up to Figure 6, 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 Figure 9, 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 Figure 9 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.



Discord	*Loc*	*m*
23.4	11,895	24
22.5	11,895	23
23.7	11,895	22
19.2	12,123	21
…	…	…
18.7	12,123	16
…	…	…
18.4	12,123	11
19.9	7,569	10
20.8	7,569	9
21.3	7,569	8

Figure 9: MADRID's multi-length discord table *M* after the second step of initialization. As a follow-up to Figure 8, here *minL* = 8, *maxL* = 24 and *S* = 1.

B. *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 Figure 10 shows, the correct normalization is to divide by the square root of the subsequence length.

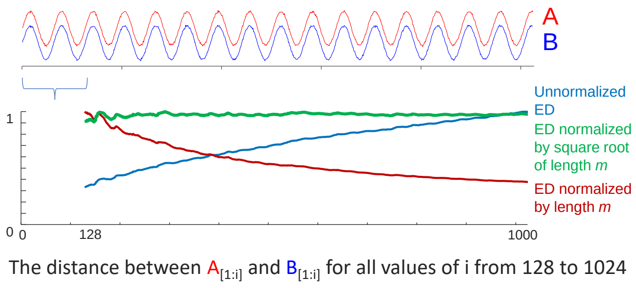


Figure 10: *top*) Two slightly noisy sine waves. *bottom*) We measured the distance between the prefixes of these sine waves of every length from 128 to 1024. The unnormalized distance grows, the length normalized distance plunges, but the proposed "divide by the square root of *m*" normalized distance remains almost constant.

For brevity, we relegate the derivation of this normalization factor to [12].

V. Discussion

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

A. *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 [23] notes "*specifying the hyperparameters of anomaly detection is particularly difficult because it requires an in-depth understanding of the data and the algorithm*", [22] bemoans "*a full parameter grid search is clearly infeasible*" and [4] 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[1].

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 Figure 11.*top*, the normal data simply consist of *two* small bumps, followed

[1] 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.

by a large bump. The anomaly we created is similar, but there are *three* small bumps. This anomaly is visually obvious.

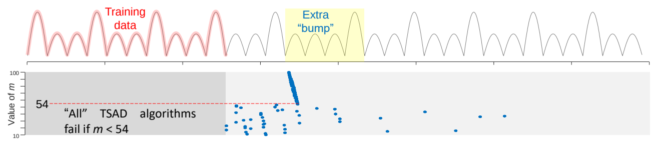


Figure 11: *top*) A trivially simple anomaly detection problem is unsolvable for any algorithm that considers a sliding window length m < 54. *bottom*) However, MADRID with $minL$ = 10, $maxL$ = 100 and S = 1 easily solves this.

As shown in Figure 11.*bottom*, $\text{MADRID}_{[10,100,1]}$ 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 [12] 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, Figure 12 invites us to imagine we forced the algorithms to use a single-sized bounding box for all images.

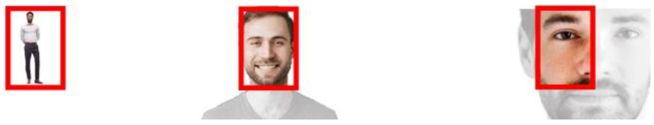


Figure 12: The futility of the "one-size-fits-all" unspoken assumption of TSAD is obvious if we consider its analogue in image processing. Any face processing algorithm would clearly find the images at the *left* or *right* extremely challenging.

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.

VI. Experimental Evaluation

To ensure experiments are easily reproducible, we have built a website [12] that contains all the data/code used in this work. All experiments were conducted on an Intel® Core i7-9700CPU 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.

A. *Revisiting Melbourne Dataset*

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

In [26] 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 (Figure 13) are:

- **15-hours**: *Police say widespread* ***flash flooding*** *is beginning to affect central areas around Melbourne* [17]
- **19-hours**: *Thousands of AFL fans have created a carnival-like atmosphere as they...* ***final parade*** [1]
- **39-hours:** *...public holiday in* ***Remembrance Day*** [2]

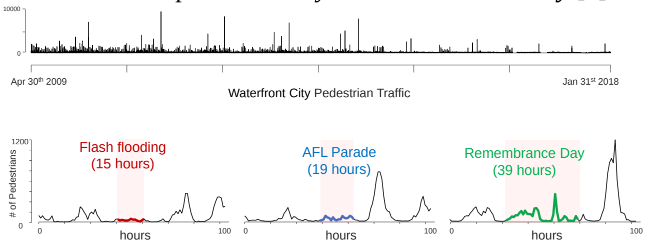


Figure 13: *top*) About eight years of pedestrian traffic. *bottom*). MADRID with $minL$ = 4, $maxL$ = 48 and step size S = 1, finds eight distinct anomalies, the top-3 are shown.

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 [13]. 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 [8] 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.

B. *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 [19][26], 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. Figure 14 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.

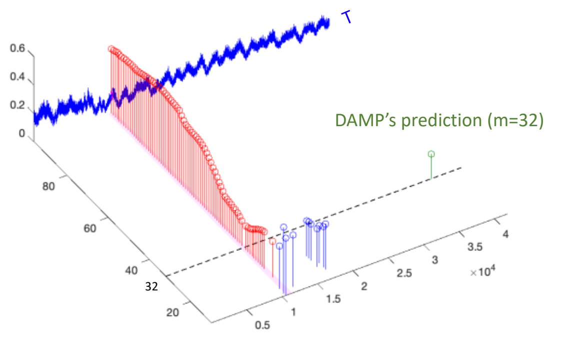


Figure 14: The 3-dimensional multi-length discordance table of dataset UCR-202, where *minL* = 10, *maxL* = 96, and step size *S* = 1.

From Figure 14, 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 [12].

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:

1. **Human Inspection**: In many cases, the ultimate filter of anomaly detection is human inspection. Anyone glancing at Figure 14 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.
2. **Highest Score**: Because we have made the discord score commeasure, we can simply report only the location of the *highest* score. In Figure 14 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*.
3. **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 [26]. However, we will not report the actual number, as there is a chance of being accused of HARKing (hypothesizing after the results are known) [16]. Here the accusation may be justified, as we took advantage of the high certainty ground truth[2] to understand these issues and design these solutions.

C. 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 Figure 15, 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. Figure 7), 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 Figure 5.

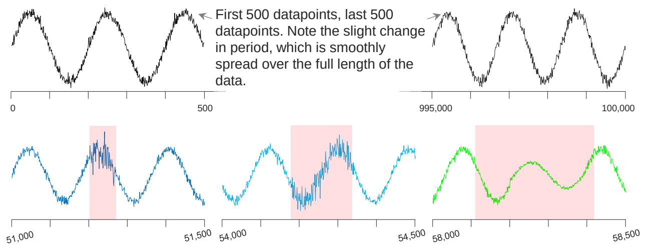


Figure 15: *top*) The beginning and end of the test datasets. *bottom*) The three anomalies embedded into the synthetic data are visually obvious with human inspection.

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 Figure 16, 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.

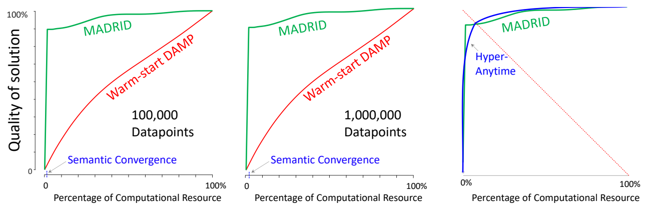


Figure 16: Anytime convergence plots for MADRID on 100K (*left*) and 1,000K (*center*), datapoints. In both cases, *minL* = 64, *maxL* = 256 and step size *S* = 1. *right*) Overlaying MADRID's convergence plot onto the nomenclature template shown in Figure 5 suggests that MADRID is a *Hyper-Anytime* 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 Figure 17 shows, this correctly flags all three anomalies.

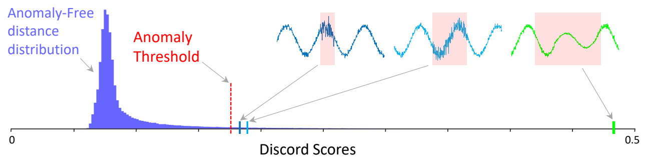


Figure 17: The classic "mean-plus-three-standard-deviations" rule correctly flags anomalies shown in Figure 15.

Having established MADRID performance, we measured the performance of two of the most cited TSAD algorithms, OmniAnomaly [27] and Telemanom [13] 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.

D. 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 [14]. 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 [14]). 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.

[2] A recent paper offers forceful evidence that many TSAD benchmarks have high undocumented uncertainty in their ground truth [20].

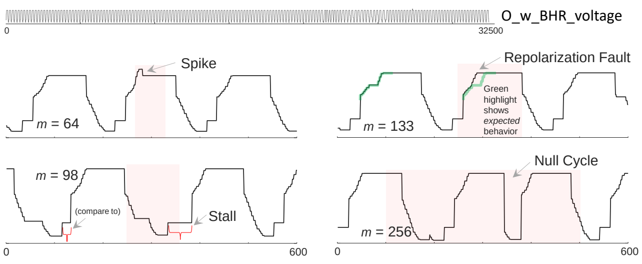


Figure 18: *top*) A 27-minute-long O_w_BHR_voltage trace from HRSS. *bottom*) MADRID with *minL* = 64, *maxL* = 256 and step size *S* = 1, finds four distinct anomalies, all true positives.

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.

E. Case Study in PSML

To further investigate multi-scale anomalies, we explored the PSML dataset [25], 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 [12].

As illustrated in Figure 19, 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 [3], Winter Storm Nadia [6], and the Thanksgiving week storm of 2019 [11]. 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*" [18].

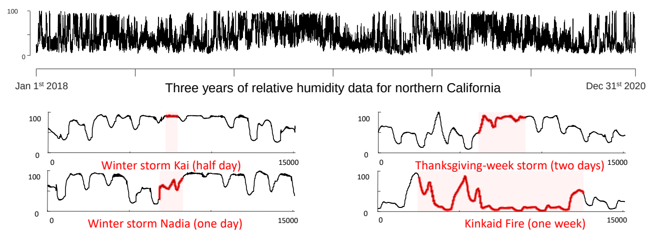


Figure 19: *top*) Three years of relative humidity data for Northern California from PSML. *bottom*) MADRID with *minL* = 720, *maxL* = 10,080 and step size *S* = 720, finds four distinct anomalies.

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 [12].

VII. Conclusions

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. 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.

References

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

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

[3] "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.

[4] 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.

[5] A. 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.

[6] A. 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

[7] B. Barz, et al. "Maximally divergent intervals for extreme weather event detection," in *OCEANS 2017-Aberdeen*, IEEE, 2017, pp. 1–9.

[8] C. 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.

[9] E. 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

[10] E. 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.

[11] 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.

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

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

[14] 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

[15] M. Zymbler and Y. Kraeva, "High-performance Time Series Anomaly Discovery on Graphics Processors," *arXiv:2304.01660*, 2023.

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

[17] 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.

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

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

[20] 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.

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

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

Dear Reader. This is an unofical expanded version of the paper