



# MERLIN++: parameter-free discovery of time series anomalies

Takaaki Nakamura<sup>1</sup> · Ryan Mercer<sup>3</sup>  · Makoto Imamura<sup>2</sup> · Eamonn Keogh<sup>3</sup>

Received: 30 March 2021 / Accepted: 5 October 2022 / Published online: 16 January 2023

© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2023

## Abstract

The burgeoning age of IoT has reinforced the need for robust time series anomaly detection. While there are hundreds of anomaly detection methods in the literature, one definition, *time series discords*, has emerged as a competitive and popular choice for practitioners. Time series discords are subsequences of a time series that are maximally far away from their nearest neighbors. Perhaps the most attractive feature of discords is their simplicity. Unlike many of the parameter-laden methods proposed, discords require only a single parameter to be set by the user: the subsequence length. We believe that the utility of discords is reduced by sensitivity to even this single user choice. The obvious solution to this problem, computing discords of all lengths then selecting the best anomalies (under some measure), appears at first glance to be computationally untenable. However, in this work we discuss MERLIN, a recently introduced algorithm that can efficiently and exactly find discords of all lengths in massive time series archives. By exploiting computational redundancies, MERLIN is two orders of magnitude faster than comparable algorithms. Moreover, we show that by exploiting a little-known indexing technique called Orchard's algorithm, we can

---

Responsible Editor: Panagiotis Papapetrou.

---

Takaaki Nakamura and Ryan Mercer these authors should be considered joint first authors.

---

✉ Ryan Mercer  
rmerc002@ucr.edu

Takaaki Nakamura  
nakamura.takaaki@dy.mitsubishielectric.co.jp

Makoto Imamura  
imamura@tsc.u-tokai.ac.jp

Eamonn Keogh  
eamonn@cs.ucr.edu

<sup>1</sup> Mitsubishi Electric Corporation, Kamakura, Japan

<sup>2</sup> Tokai University, Tokyo, Japan

<sup>3</sup> University of California, Riverside, USA

create a new algorithm called MERLIN++, which is an order of magnitude faster than MERLIN, yet produces identical results. We demonstrate the utility of our ideas on a large and diverse set of experiments and show that MERLIN++ can discover subtle anomalies that defy existing algorithms or even careful human inspection. We further compare to five state-of-the-art rival methods, on the largest benchmark dataset for this task, and show that MERLIN++ is superior in terms of accuracy and speed.

**Keywords** Time series · Anomaly detection · Multi-scale

## 1 Introduction

Humans measure things, and with rare exceptions, things change over time, producing *time series*. Time series data is ubiquitous in industrial, medical, and scientific settings. One of the most basic time series analytical tasks is to simply spot anomalous regions. In some cases, this may be the end goal of the analytics. In other cases, it may be simply a preprocessing step for a downstream task, for example precursor discovery or data cleaning. There are at least hundreds of algorithms for finding anomalies, but which should we use?

Since their introduction, *Time Series Discords* have emerged as a competitive approach for discovering anomalies (Lin et al. 2005). For example, Kumar and colleagues conducted an extensive empirical comparison concluding that “on 19 data sets, comparing 9 different techniques (time series discords) is the best overall” (Chandola et al. 2009). We attribute much of this success to the simplicity of the definition. Time series discords are intuitively defined as the subsequences of a time series that are maximally far away from their nearest neighbors. This definition only requires a single user specified parameter, the subsequence length. With only a single parameter to fit,<sup>1</sup> it is harder to *overfit*, and overfitting seems to be a major source of false positives for this task (Chandola et. al 2009; Hundman et al. 2018).

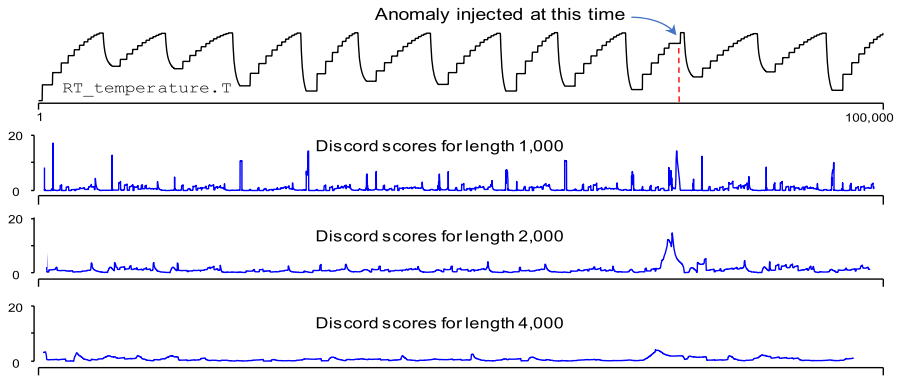
To help the reader appreciate the importance of the subsequence length in anomaly discovery, let us consider an excerpt of the Gasoil Plant Heating Loop Data Set (Filonov et. al 2016). This data set had a simulated cyber-attack introduced at the time indicated by the red dashed line shown in Fig. 1.*top*.

We computed the anomaly scores for every subsequence for three different lengths. For the shortest length of 1000, it is unsurprising that we get many spurious anomalies. This system transitions between discrete temperature states, giving it a “staircase” effect. If the subsequence length is less than the length of a step, the z-normalization “blows up” the subsequence and produces unstable results. At the longer length of 4000 the curse of dimensionality is beginning to dominate. As noted by Beyer et. al. “as dimensionality increases, the distance to the nearest data point approaches the distance to the farthest data point” in (Beyer et al. 1999).

However, consider the plot for subsequences of length 2000 shown in Fig. 1. There is a clear peak at the correct location. Moreover, it is significantly larger than the mean

---

<sup>1</sup> Note that some *algorithms* that discover discords may have other parameters, the discord representation itself requires just a single parameter.

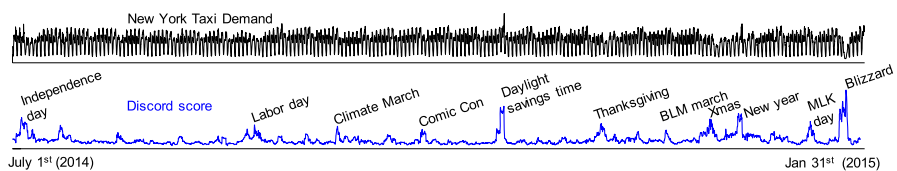


**Fig. 1** *Top* An excerpt from Filonov’s Gasoil dataset, a reading from `RT_temperature.T` (Filonov et al. 2016). *Bottom* The discord scores for three lengths, 1000, 2000 and 4000. The higher the score, the more anomalous the corresponding subsequence is

value of the scores, giving a clear visual signal that this is a true anomaly. This example shows that there is a “sweet spot” (or rather, sweet *range*) for subsequence length when performing anomaly discovery. In some cases, the analyst may have a first-principles-model or experience to suggest a good value, but recall that anomaly/novelty discovery is often exploratory by nature.

Before continuing, we will take the time to reiterate the utility of discord discovery in the vast space of anomaly detection techniques (Chandola et al. 2009; Filonov et al. 2016; Laptev and Amizadeh 2015; Vasheghani-Farahani et al. 2019; Däubener et al. 2019; Barz et al. 2017; Doan et al. 2015; Hundman et al. 2018; Ahmad et al. 2017; Keogh et al. 2005; Bu et al. 2009). In essence, we want to answer the following question: “*why make an effort to address the noted weakness of discords, rather than invent or use a different method?*” Fig. 2 shows the discord scores computed for a benchmark dataset that has been considered in over one hundred research efforts (Ahmad et al. 2017).

Note that the discords discovered have different causes. Some are predictable holidays, some are caused by ad-hoc events, like the hastily organized BLM march, and some are severe weather events. One anomaly is simply a bookkeeping error; setting



**Fig. 2** *Top* Six months of taxi demand in New York City. *Bottom* The discord scores for subsequence length of one day. Most of the discords discovered have an intuitive meaning

the clock back by one hour for daylight saving time (DST) made it appear as if the taxi demand doubled just after midnight.<sup>2</sup>

Vasheghani and colleagues also consider this dataset (Vasheghani-Farahani et al. 2019). While they find some true positives, they also find many false positives. More importantly, however, they tell us that “*the parameters for this experiment are  $w = 30$ ,  $k = 6$ ,  $q = 5$ ,  $h1 = -3.57$ , and  $h2 = -4.28$ .*” Thus, to find these anomalies, they had to set five parameters, two of them to three significant digits. Similarly, there are many research efforts on deep learning anomaly detection. One recent paper using an LSTM model also considers this taxi dataset (Zhang 2019). It *does* find Xmas, New Year, and the blizzard but fails to find Thanksgiving, the BLM march, or the (obvious even to the human eye) daylight-savings-time anomaly.

These two comparisons highlight the attractiveness of discords for practitioners. It is hard to imagine that most practitioners would be able and willing to carefully set the five parameters of the Markov Chain approach (Vasheghani-Farahani et al. 2019), or the dozen or so parameters/choices for a LSTM model (Hundman et al. 2018). Moreover, even if they did so, with so many parameters to fit on a small dataset, avoiding overfitting would be very challenging.

Because the effectiveness of discords is central to our work, we will take the time to consider just one more motivating example. A recent paper conducted a “bake-off” with eight diverse representatives of the state-of-the-art anomaly detection algorithms (as opposed to simply minor variants of a single approach) (Däubener et al. 2019). Figure 3 contrasts the results on one benchmark (Yahoo) dataset with time series discords.

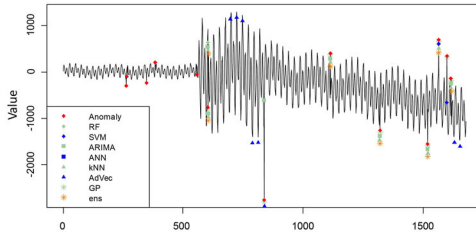
The authors of this study noted, “*None of the algorithms tested can correctly identify the first five anomalies, ... AdVec generates seven false positives...*” In contrast to these eight approaches, the discord approach performs perfectly on this task, assuming only that its one parameter is a reasonable value. The goal of this research effort is to remove the need to set even that sole parameter. We call our proposed algorithm MERLIN.<sup>3</sup> MERLIN can efficiently and exactly discover discords of every possible length and then either report all of them or just the top-K-discords under an arbitrary user defined scoring metric.

To summarize this section, we have chosen to fix flaws and optimize discord discovery because time series discords are already widely used, in astronomy (Daigavane et al. 2020), energy management (Nichiforov et al. 2020), medicine etc. There is no forceful evidence that any of the more recently proposed algorithms are generally better (Wu and Keogh 2021; Huet et al. 2022; Kim et al. 2022; Hwang et al. 2022), and there is at least anecdotal evidence that setting the expected anomaly length is a major pain point for many practitioners.

---

<sup>2</sup> Note that this DST anomaly is misidentified in the original work that introduced this dataset as the NY-Marathon anomaly (Ahmad et al. 2017). This misidentification has since been repeated in dozens of papers. We are confident that our labeling is correct. If we correctly process the data with the standard DST algorithm  $\text{count}(1-2\text{am}) = \frac{1}{2} \text{apparent count}(1-\text{am})$ , then the apparent anomaly disappears.

<sup>3</sup> This name is a play on the fact that the first paper on time series discords was titled “Approximations to Magic” (Lin et al. 2005). Merlin was the magician of the Arthurian legend. In addition, Mitsubishi Electric Corporation’s subsidiary in the USA is called MERL (Mitsubishi Electric Research Laboratory, Boston).



Note to reader. The green text is from the Däubener paper.

Fig. 2. Yahoo time series A4: 8, with 13 labeled true anomalies (red dots). None of the algorithms tested can correctly identify the first five anomalies, while the next 8 are correctly identified by gaussian processes, and 7 by four other methods. AdVec generates seven false positives.

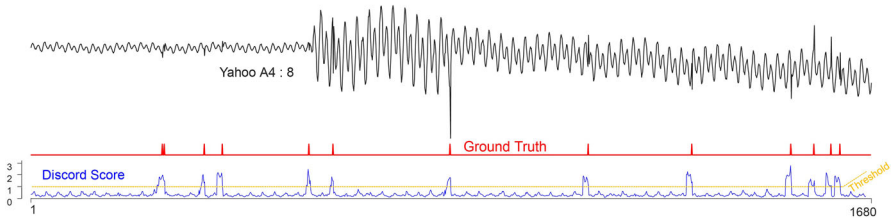


Fig. 3 *Top* A screen capture from (Däubener et al. 2019) showing the performance of eight state-of-the-art anomaly detectors on one of the Yahoo benchmarks (Laptev and Amizadeh, 2015). *Bottom* Time series discords (here, of length 8) have a perfect score on this problem, with only the mildest of assumptions

The rest of this paper is organized as follows. In Sect. 2, we introduce background material and related work. Section 3 reviews the MERLIN algorithm before introducing MERLIN++ , an extension that is an order of magnitude faster, while producing the exact same results. We offer an extensive empirical evaluation in Sect. 4, before concluding with a discussion of our findings in Sect. 5.

## 2 Background and related work

In this section, we introduce all the necessary definitions and notations, including a review of an existing algorithm for discord discovery that we will use as a starting point for our research. We will also consider related work to put our ideas in context (Yankov et al. 2008).

### 2.1 Time series notation

We begin by defining the data type of interest *Time Series*:

**Definition 1:** A Time Series  $\mathbf{T} = t_1, t_2, \dots, t_n$  is a sequence of  $n$  real values.

Our distance measures quantify the distance between two time series based on local subsections called *subsequences*:

**Definition 2:** A subsequence  $\mathbf{T}_{i,L}$  is a contiguous set of  $L$  values starting from position  $i$  in time series  $\mathbf{T}$ ; the subsequence  $\mathbf{T}_{i,L}$  is of the form  $\mathbf{T}_{i,L} = t_i, t_{i+1}, \dots, t_{i+L-1}$  where  $(1 \leq i \leq n - L + 1)$  and  $L$  is a user-defined subsequence length with value in range of  $3 \leq L \leq |\mathbf{T}|$ .

Here we allow  $L$  to be as short as three, although that value is pathologically short for almost any domains.

Many time series analytical algorithms need to compare subsequences using some distance measure  $Dist$ ; here we use the z-normalized Euclidean distance. As pointed out by the original authors of the discord definition, we must be careful to exclude certain trivial matches from any meaningful definitions of subsequence similarity. Effectively, an exclusion zone is place around each subsequence query  $\mathbf{T}_{q,L}$  such that only subsequences  $\mathbf{T}_{p,L}$  satisfying  $|p - q| \geq L$  are compared to  $\mathbf{T}_{q,L}$ . Rephrased, nearest neighbor subsequences are non-overlapping. We can now use this definition of non-self matches to define *time series discords*:

**Definition 3: Time Series Discord:** Given a time series  $\mathbf{T}$ , the subsequence  $\mathbf{T}_{i,L}$  of length  $L$  beginning at position  $i$  is said to be the discord of  $\mathbf{T}$  if  $\mathbf{T}_{i,L}$  has the largest distance to its nearest neighbor  $\mathbf{T}_{nni,L}$ . That is,  $\forall$  pairs of nearest neighbor subsequences  $\mathbf{T}_{j,L}$  and  $\mathbf{T}_{nnj,L}$  of  $\mathbf{T}$ ,  $Dist(\mathbf{T}_{i,L}, \mathbf{T}_{nni,L}) > Dist(\mathbf{T}_{j,L}, \mathbf{T}_{nnj,L})$ .

The starting location of the discord is recorded in `index` and its distance to its nearest neighbor is recorded in `distance`. All previous efforts to find discords considered only a single length. However, we plan to consider all lengths in a given range; thus producing an array of discords indexed by the length  $L$ , `discordL = [indexL, distanceL]`.

For simplicity, we define only the top-1 discord, the generalization to top-K is trivial (Yankov et al. 2008). Having defined discords, we will next review an algorithm to discover them.

## 2.2 A review of the SOTA discord discovery algorithm

Our proposed algorithm makes repeated use of the discord discovery algorithm introduced in (Yankov et al. 2008). The algorithm was unnamed in that work, so for clarity we will call it DRAG, which is both a truncated version of the inventor's name and a backronym that stands for Discord Range-Aware Gathering.

Why generalize DRAG? As shown in the introduction (and forcefully shown in Sect. 4.10) time series discords are still state-of-the-art for time series anomaly detection. However, time series discords do require the user to set one unintuitive parameter, the subsequence length  $L$ . Our work is motivated by the observation that the most obvious way to bypass that difficulty is to simply find discords at *all* possible lengths. Then the choice of the time series discords to consider can be left to a post-hoc algorithm that has access to all the information it needs. For example, if an algorithm (or human) noted that for all possible lengths, the anomaly was in a single location, it can just report that single location. The Mars Rover example in Fig. 15 is such an example. A downstream algorithm could report "Anomaly is at location 2400". In contrast, a downstream algorithm examining the NY-Taxi data shown in Fig. 16 could report "In the range of 5 to 10 h, there is an anomaly on November 2<sup>nd</sup>; in the range 11 h to 96 h, there is an anomaly beginning November 27<sup>th</sup>". To the

best of our knowledge, no existing anomaly detection algorithm has this multi-scale ability.

Recall our question, *why generalize DRAG?* DRAG is still state-of-the-art, in terms of speed, for discovering discords. However, its speed (but not its accuracy), critically relies on the setting of a difficult-to-set parameter. For any user-given length, the algorithm requires the input parameter  $r$ . This value should ideally be set such that it is just a little less than the *discord distance*; that is, the distance between the discord and its nearest neighbor. Of course, that distance is unknown at this point, so the user must provide an estimate. If this estimate is accurate, just a little less than the eventually discovered true discord value, then DRAG has a time and space complexity of just  $O(nL)$ . If the estimate is much too small, the algorithm will give the correct result but have a time and space complexity of  $O(n^2)$ . In either case, we call any invocation of DRAG that used an  $r$  value less than the eventually returned discord distance a *success*.

In contrast, if the estimate for  $r$  is too large, the algorithm will return null, a situation we denote as a *failure*. Of course, the situation can be remedied but requires the user to reduce the  $r$  value and try again. This sensitivity to  $r$  parameter was largely glossed over in the original paper (Yankov et al. 2008), but as we will show in Sect. 3 it is a significant limitation of DRAG. However, as we will later explain, we have solved this issue for MERLIN.

We refer the reader to (Yankov et al. 2008) for a detailed explanation of the DRAG algorithm, but for completeness, we will give an overview. The DRAG algorithm is a two-phase algorithm, with each phase being a pass across the time series.

- **Phase I** As shown in Table 1 the algorithm initializes a set  $C$ , of candidate discords by placing the first subsequence in  $C$ . The algorithm then “slides” along the time series examining each subsequence. If the subsequence currently under consideration is greater than  $r$  for any item in the set, then it may be the discord, so it is added to the set. However, if any items in the set  $C$  are less than  $r$  from the subsequence under consideration, we know that they could not be discords; thus, they are admissibly pruned from the set. At the end of Phase I, the set  $C$  is guaranteed to contain the true discord, possibly with some additional false positives.

Note that the algorithm can end in *failure* (line 14). Or, we can regard this situation as successfully finding no discord greater than the threshold of  $r$ . If the user wants to find the discord regardless of its eventual distance, she must run the algorithm again with a smaller value for  $r$ . We will have more to say about this issue in Sect. 3.1.

After Phase I has built a set of candidate discords, we are now ready to run Phase II to refine them.

- **Phase II** As shown in Table 2, we again slide along the time series, this time refining the candidates to remove the false positives. We simply consider each subsequence’s distance to every member of our set, doing a *best-so-far* search for each candidate’s nearest neighbor. The algorithm returns a sorted list of all discords with a distance greater than  $r$  (there is guaranteed to be at least one). The largest such score is our top-1 discord.

Given this review of the algorithm, it is easy to see why its performance depends so critically on the user’s choice of  $r$ . A pessimistically small value for  $r$  will mean

**Table 1** Phase I, candidate selection algorithm

<b>Procedure: CandidateSelection(<math>T, L, r</math>)</b>	
<b>Input:</b> Time series $T$ , Subsequence length $L$ , Range of discords $r$	
<b>Output:</b> Candidate set of discords $C$	
1	$C = \{\}$ // Start with empty set
2	<b>for</b> $i = 1$ to $ T  - L + 1$ // Scan all subsequences
3	$iscandidate = \text{true}$
4	<b>for</b> $j$ in $C$
5	<b>if</b> $i$ and $j$ are <b>not</b> trivial matches
6	<b>if</b> $\text{dist}(T_{i,L}, T_{j,L}) < r$
7	$C = C \setminus j$
8	$iscandidate = \text{false}$ // We can prune this
9	<b>if</b> $iscandidate$
10	$C = C \cup \{i\}$ // Add to candidate set
11	<b>if not isemptyset()</b>
12	<b>return</b> $C$ // Implicitly return success
13	<b>else</b>
14	<b>return failure</b> // Explicitly return failure

**Table 2** Phase II, discords refinement algorithm

<b>Procedure: DiscordRefinement(<math>C, T, L, r</math>)</b>	
<b>Input:</b> Discord candidate set $C$ , Time series $T$ , Subsequence length $L$ , Range of discords $r$	
<b>Output:</b> Set of discords (index, distance) $D$	
1	$D = \{\}$ // Start with empty set
2	<b>for</b> $i = 1$ to $ T  - L + 1$ // Scan all subsequences
3	$isdiscord = \text{true}$
4	<b>for</b> $j$ in $C$ // Scan all candidates
5	<b>if</b> $i$ and $j$ are <b>not</b> trivial matches
6	$d = \text{dist}(T_{i,L}, T_{j,L})$
7	<b>if</b> $d < r$
8	$C = C \setminus j$
9	$isdiscord = \text{false}$ // Eliminate candidate
10	<b>if</b> $isdiscord$
11	$D = D \cup \{(j, L, d)\}$ // Add to the set of true
12	<b>return</b> $D$ // Return discord

that in Phase I *most* subsequences will be added to the candidate set, exploding the time and space complexity to the  $O(n^2)$  case. However, if  $r$  is chosen well, the size of this set remains very small relative to  $n$ . For example, in (Yankov et al. 2008) they show that even with a million subsequences, for a good value of  $r$ , the size of  $C$  does not exceed fifty candidates, making the algorithm effectively  $O(nL)$ .

## 2.3 Related work

In the previous section we claimed DRAG is the state-of-the-art in *discord discovery* (we are not (yet) claiming it is state-of-the-art in *anomaly detection*). The reader may be surprised to find that we did not list the more recent Matrix Profile (MP) algorithms



as state-of-the-art (Yeh et al. 2016). The MP algorithms (SCRIMP etc.) surely are state-of-the-art for *motif discovery*, and as a side-effect of motif discovery, they happen to also compute discords. However, the MP algorithms are all  $O(n^2)$ . It is impressive that their time complexity is independent of  $L$ , as almost all algorithms in this space scale poorly with  $L$ , the classic curse of dimensionality. Nevertheless, for our purposes these algorithms compute much more information than is needed and are thus much slower than what we can achieve for the limited task-at-hand.

There are also algorithms that discover discords by discretizing the time series, typically using SAX, and hashing the symbolic words that correspond to subsequences (Lin et al. 2005; Keogh et al. 2005). The basic idea being that a lack of collision for a word is evidence that the word might be unique hence corresponding to a discord. After the candidates have been identified this way, an algorithm similar to Phase II in Table 2 can be used to refine them. These algorithms *can* be competitive with DRAG but only if three parameters for SAX are very carefully set (Keogh et al. 2005). Moreover, such algorithms based on discretizing the space are always approximate relative to the original data.

The more general area of anomaly detection is increasingly difficult to review. In particular there has been a recent explosion of papers on deep learning for anomaly detection (Filonov et al. 2016; Vasheghani-Farahani et al. 2019; Zhang 2019; Däubener et al. 2019; Hundman et al. 2018; Ahmad et al. 2017; Bu et al. 2009). This is a diverse group of research efforts; the one thing that they have in common from our point of view is that they all require many critical parameters to be set. For example, (Vasheghani-Farahani et al. 2019) explicitly lists five parameters (and perhaps has a few more in the background), the LSTM network in (Hundman et al. 2018) requires eight parameters. Clearly deep learning has had an enormous impact in image processing, NLP etc. However, as we hinted at in Figs. 2, 3, and 15 as we will later empirically show, it is not obvious that deep learning outperforms simpler and more direct shape-based methods for anomaly detection.

A recent work surveyed the literature and concluded “*The state-of-the-art solutions for subsequence anomaly detection (are) discords*” (Boniol et al. 2021). While acclaiming the basic distance-based approach of discords, this work then goes on to suggest that discords have two weaknesses: “(i) *the number of anomalies present in a dataset is usually more than one and is not known in advance; and (ii) often times anomalous subsequences repeat themselves (approximately the same) in the same dataset.*”

However, in Figs. 16, 17, 18, 19, 20, and 22, we show that MERLIN *is* capable of finding *multiple* anomalies in a single dataset. Moreover, recall that Fig. 3.*bottom* offers strong evidence that even if we confine our attention to single-length discords, the top-K discords can discover K different anomalies.

As to the second point raised by (Boniol et al. 2021), this problem has been noted before, and called the “twin freaks” problem (Wei et al. 2006). To be clear, the issue is that a *single* occurrence of a strange shape would be a high scoring discord, but if it happened again, the two occurrences would be mutual nearest neighbors and, thus, have a low discord score.

However, recall the famous quote from Anna Karenina, “All happy families are alike; each unhappy family is unhappy in its own way”. A time series version of this

might be: all normal behaviors are alike, each anomaly is anomalous in its own way. For example, there might be only one or very few ways to have a normal bipedal gait of walking. However, there are essentially an infinite number of ways to stumble, slip, topple, trip, tumble, flounder, lurch, reel, stagger, sway, teeter or fall. As such, we claim that repeated shape conversed anomalies are rare. It is telling that a paper that wanted to introduce an anomaly detection method that was invariant to “twin freaks” had to resort to copying and pasting data to contrive the situation (Bu et al. 2009), but was unable to find a single real example. In any case, this issue seems to be essentially moot, as it can be solved by changing the first nearest neighbor (Definition 4) to the  $k$ th nearest neighbor. However, given that in practice this rarely seems to be an issue, in this work we use only the simple first nearest neighbor.

Finally, recently there has been a spate of papers that suggest that because of flaws in both benchmark *datasets* (Wu and Keogh 2021), and in evaluation *metrics*, much of the literature on time series anomaly detection is unreliable (Wu and Keogh 2021; Huet et al. 2022; Kim et al. 2022; Hwang et al. 2022). We will not wade too deep into this debate, other than to note that we have taken great care to ensure that our datasets are not trivial, and our evaluation metric cannot be “gamed”.

## 2.4 Why distance based anomaly detection?

An extraordinary number of approaches have been applied to the problem of anomaly detection in time series, including: Isolation Forests, One-Class Support Vector Machines, Convolutional Neural Networks, Residual Neural Networks, Long Short Term Memory networks, Gated Recurrent Units, Autoencoders, Multi-Layer Perceptrons, ARIMA models, Markov models, Minimal Description Length, Bayesian techniques, Rule-Based Systems etc. Indeed, it is difficult to think of a single machine learning or signal processing tool that has not been advocated as at least part of a time series anomaly detection solution. Given the plethora of possible approaches, why do we so strongly advocate a distance-based approach?

Part of the answer is simply that distance-based methods offer highly competitive performance, as we shall demonstrate in Sect. 4. Another reason is the dearth of parameters that need to be set, as few as one, or for MERLIN, *none*. However, there is another important and practical reason. In the last twenty years, distance-based methods have been highly competitive for time series *classification*. Because of this, the community knows a lot about time series distance measures, and this knowledge can be directly exploited here. For example:

- Suppose that we have years of experience with pedestrian traffic anomaly detection with a data source that happens to be sampled twice an hour (see Figs. 18 and 19). Further suppose that we have managed to learn a threshold  $T$  for sounding an alert, any discord score that is greater than 15.2 is a significant anomaly that warrants attention. Now imagine that we learn that in the new year an upgraded sensor will produce the data at a four times finer sampling rate of eight times an hour. We know from published results that we can find the new threshold as  $T_{\text{new}} = 15.2 \times \sqrt{4}$  (Linardi et al. 2020). For all the other methods mentioned above, it is not clear how we should adjust a threshold or if that is even possible.

- Suppose once more that we are tasked with monitoring pedestrian traffic anomaly detection. This time the traffic engineer tells us “*It only makes sense to compare midnight to midnight, and anything that happens between 3 and 5am is twice as important as anything that happens at any other time*”. We can trivially support this domain information with distance-based measures. Indeed, if using the MASS to compute the distance we only have to change two lines of code (Mueen 2015). As before, it is not clear how we “tell” most other approaches the relative importance of various time periods.

To summarize, in the last two decades the community has gathered a vast store of knowledge about time series distance measures. We understand how to deal with time series data that has wandering baselines, missing values, uncertain values, non-constant noise levels, uniform scaling, etc. by either adjusting the distance measure or by preprocessing the data before calling the distance measure (often, these are logically equivalent). In contrast, for most other approaches it is not clear how we can exploit our understanding of the domain. For this reason alone, distance-based measures are very attractive to practitioners.

### 3 The MERLIN++ algorithm

We begin by illustrating some novel observations about the sensitivity of DRAG to the  $r$  parameter.

#### 3.1 Exploitable observations about DRAG

Consider the small synthetic dataset shown in Fig. 4: it is simply a slightly noisy sine wave with an obvious “anomaly” embedded in it starting at location 1,000.

What would be an appropriate value of  $r$  here given that we wish to discover discords of length 512? Even with significant experience with the DRAG algorithm, it is not immediately obvious to us. To gain some intuition, in Fig. 5, we considered every possible value of  $r$  from 1 to 40 in increments of 0.25, measuring both how long DRAG takes and whether it ended in *success* or *failure*.

After the fact, we know that the true discord value is 10.27. The reader will appreciate that this value, or rather, this value minus a tiny epsilon, is the optimal setting of  $r$  (Yankov et al. 2008).

Suppose that we had guessed  $r = 10.25$ , then DRAG would have taken 1.48 s to find the discord. However, had we guessed a value that was just 2.5% less, DRAG would have taken 9.7 times longer. Had we guessed  $r = 1.0$  (a perfectly reasonable value on visually similar data), DRAG would have taken 98.9 times longer.

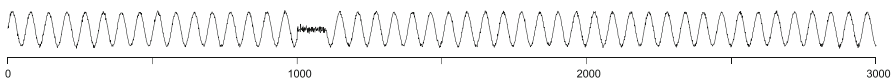
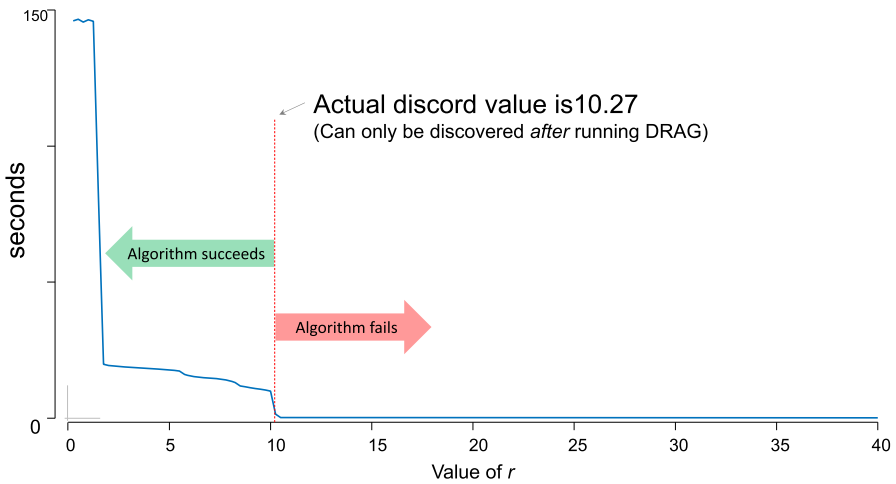


Fig. 4 A slightly noisy sine wave with an anomaly embedded at location 1,000



**Fig. 5** The time taken for DRAG given values for  $r$  that range from 1.0 to 40.0. For any value greater than 10.27 the algorithm reports *failure* and must be restarted with a lower value

In the other direction, had we guessed any greater than 1% more, DRAG would have failed. The time it takes to complete a *failed* run is about 1/6 the time of our successful run when  $r = 10.25$  was set to the 10.25 guess. So, while failure is cheaper, it is not free. This eliminates certain obvious algorithms to find a good value for  $r$ . For example, we could have tried every integer from 40 downwards until *success*, but that would have cost 29 *time-for-failures* plus one *time-for-success* with  $r = 10$ , which is about 39.2 s or about 26 times worse than our “lucky” guess of  $r = 10.25$ .

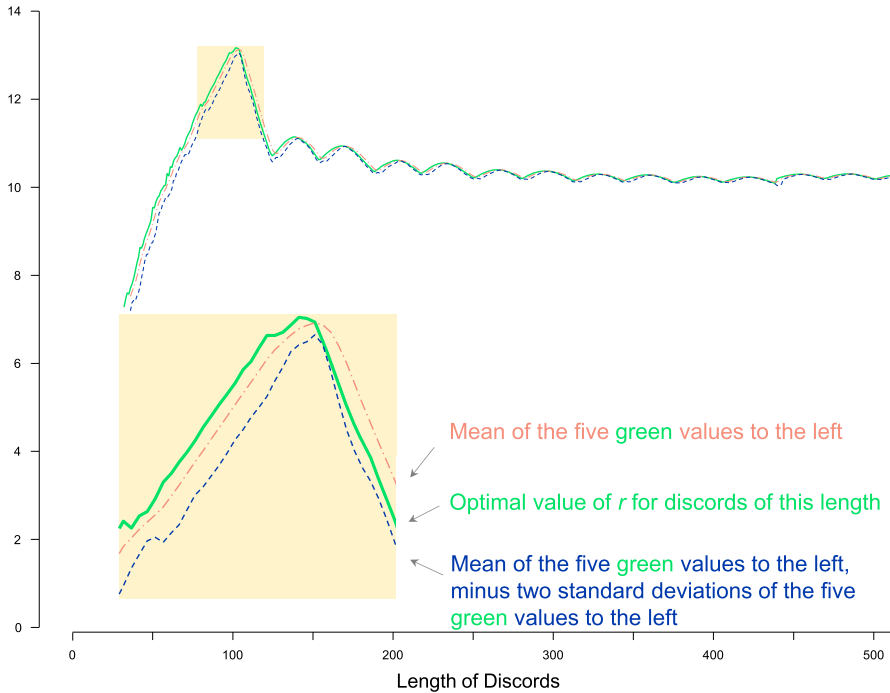
Note that a failure lets us know that our guess for  $r$  was too high, but otherwise does not appear to contain exploitable information as to a better value for  $r$ .

One might imagine that there is some simple heuristic for setting  $r$ . If there is, it has eluded us (and, to the best of our knowledge, the rest of the community that uses this algorithm (Chandola et. al 2009)). Even on datasets that are superficially similar to each other, say two examples of ten minutes of healthy teenage female electrocardiograms, the best value for  $r$  can differ by at least two orders of magnitude.

In summary, choosing a good value for  $r$  is critical for DRAG to be efficient, but it is a very difficult parameter to set. However, for our task-at-hand, there is a ray of hope. The best value for  $r$ , for discords of length  $L$ , is likely to be very similar to the best value for  $r$ , for discords of length  $L - 1$ . To see this, we measured the correlation between the optimal  $r$  for discords with lengths differing by one, for all  $L$  from 16 to 512 for the example shown in Fig. 4. The correlation was 0.998.

It is important to ward off a possible misunderstanding, suggested by this very high correlation; these differences are typically very small, but they are not necessarily all positive. Because we are working with z-normalized Euclidean distance, when we make the discord length longer, the discord score can increase, decrease or stay the same. The blue line shown in Fig. 6 illustrates this fact.

As Fig. 6 makes clear, the obvious idea of using the last  $\text{discord}_L$  distance to set the value for  $r$  when attempting to discover  $\text{discord}_{L+1}$  is a bad idea. In this



**Fig. 6** (Blue line) The discord score, which is also the optimal setting for  $r$ , for the dataset shown in Fig. 4. The inset shows a zoom-in of the region from 64 to 100. Here we can more clearly see the blue line is accompanied by a red line, which attempts to predict it, using only the five previous values (Color figure online)

example, it would result in 45.4% of the runs ending in *failure*. Thus, we want the value of  $r$  to be a “little less” than discord distance. The meaning of “little less” here depends on the data and on the lengths currently considered, so we propose to learn it by looking at the variance of the last few (say five) discord values.

Thus, we have an informal algorithm to set the value of  $r$ .

Compute the discords working from the minimum to the maximum length. At each stage, compute the mean  $\mu$ , and standard deviation  $\sigma$ , of the last five discord distances, and for the next invocation of DRAG, use  $r = \mu - 2\sigma$ . If DRAG reports *failure*, repeatedly subtract another  $\sigma$  from the current value of  $r$  until it reports *success*.

Using this simple prediction algorithm on the dataset shown in Fig. 4, we would have zero failures. Moreover, on average, the value predicted would be 99.03% of the optimal value for  $r$ .

This idea leaves just one thing unspecified. How do we set  $r$  for the first five discord lengths? We do have an *upper* bound as to the largest possible discord distance for time series of length  $L$ , it is simply the largest possible distance between any pair of z-normalized subsequences of length  $L$ , which is  $2\sqrt{L}$  (Marcus and Minc et al. 1992). So, for the first length of discord we attempt to discover, we can set  $r = 2\sqrt{L}$  and keep halving it until we get a *success*. In general,  $2\sqrt{L}$  is a very weak bound and

**Table 3** The MERLIN Algorithm

<b>Procedure:</b> MERLIN ( $T, MinL, MaxL$ )		
<b>Input:</b> Time series $T$ , Minimum subsequence length $MinL$ , Maximum subsequence length $MaxL$		
<b>Output:</b> Set of discords (index, length, distance) $D$		
1	$r = 2 \times \text{sqrt}(MinL)$	// Set $r$ to its largest possible value
2	$distance_{MinL} = -\text{inf}$	// Allow entry into loop
3	<b>while</b> $distance_{MinL} < 0$	// Find first discord
4	$[index_{MinL}, distance_{MinL}] = \text{DRAG}(T, MinL, r)$	
5	$r = r \times \frac{1}{2}$	// if loop repeats, make $r$ smaller
6	<b>for</b> $L = MinL + 1$ to $MinL + 4$	// Find next 4 discords
7	$distance_L = -\text{inf}$	// Allow entry into loop
8	$r = 0.99 \times distance_{L-1}$	
9	<b>while</b> $distance_L < 0$	// Make $r$ a little smaller till success
10	$[index_L, distance_L] = \text{DRAG}(T, L, r)$	
11	$r = r \times 0.99$	// if loop repeats, decrease $r$ in 1% steps
12	<b>for</b> $L = MinL + 5$ to $MaxL$	// Find all remaining discords
13	$M = \text{mean}(distance_{L-1} \text{ to } L-5)$	// Use local info about..
14	$S = \text{STD}(distance_{L-1} \text{ to } L-5)$	// ..the mean and STD..
15	$r = M - (2 \times S)$	// ..to predict good value for $r$
16	$[index_L, distance_L] = \text{DRAG}(T, L, r)$	
17	<b>while</b> $distance_L < 0$	// looks like our $r$ was too high..
18	$[index_L, distance_L] = \text{DRAG}(T, L, r)$	// ..so let us reduce..
19	$r = r - S$	// until success

likely to produce many failures. So, we do not want to do this for the next four items. Here instead, we can use the previous discord distance, minus an epsilon, say 1%. In the very unlikely event that this was too conservative and resulted in a failure, we can keep subtracting an additional 1% until we get a *success*.

Table 3 formalizes this algorithm.

The algorithm has an apparently arbitrary choice. Why work from the minimum to the maximum length rather than the other way around? Recall that it is only for the first invocation of DRAG that we are completely uncertain about a good value for  $r$ , and we may have multiple failure runs and/or invoke DRAG with too small of a value for  $r$ , making it run slow. It is much faster to do this single unoptimized run on the shorter subsequence lengths.

The memory complexity for MERLIN is the same as that of DRAG, as MERLIN is essentially comprised of repeated runs of DRAG. The only information shared between the iterations of MERLIN is a single real number,  $r$ . Recall that a very poor choice of  $r$ , that is, a pessimistically small value, would mean that in Phase I many subsequences will be added to the candidate set, resulting in a space complexity of  $O(n^2)$ . However, if  $r$  is chosen well, the candidate set remains small relative to  $n$ . For example, in (Yankov et al. 2008) they show that even with a time series of length one million, and subsequences of length 128, provided they have a good value for  $r$ , the size of the candidate set does not exceed fifty candidates, or 0.64% memory overhead.

Of course, that *assumes* a good value for  $r$ . This was difficult to achieve for DRAG, but trivial for MERLIN, in all iterations but the first. However, in just the first iteration, we creep up on the value of  $r$  by decreasing a value that is guaranteed to be too large. This may cause more *failure* runs (line 14 of Table 1) than an inspired guess, but we

are only doing this once, and we do it when the cost of failure is relatively cheap, as the subsequence length is the shortest value that we need to consider.

For all the experiments conducted in this paper, the memory overhead was never greater than 3%, thus we do not show experiments discussing memory demands.

### 3.2 Is there some other way to set $r$ ?

Our work is motivated by the claim that there is no heuristic to set a good value for  $r$  for DRAG in the general case, i.e., in the *single* run of the DRAG algorithm case. It is difficult to prove a negative, but below we will discuss why we think it is very unlikely that any such heuristic could exist.

To see this, let us imagine that such a heuristic  $H$  is claimed. Suppose we test  $H$  on a long time series  $\mathbf{T}$ , and it predicts a value for  $r$ . For concreteness, let us say that  $\mathbf{T}$  was a much longer version of the data in Fig. 4, a slightly noisy sine wave with a significant anomaly embedded. Recall that this optimal value of  $r$  must be a “little less” than the true discord distance. We can ask ourselves, how many datapoints in  $\mathbf{T}$  would we need to change in order to radically change the true discord distance, and thus invalidate the prediction made by  $H$ ? The answer is only  $L$  datapoints. We could:

- Replace the embedded anomaly with a typical sine wave. This would mean that the true discord distance would *decrease* to close to zero.
- Replace the embedded anomaly with an even more drastic anomaly, in this case a random vector (not a random walk). This would mean that the true discord distance would dramatically *increase*.

This adversarial argument tells us that any robust heuristic  $H$  must look at *all* the data, otherwise it could give a radically optimistic or radically pessimistic value for  $r$ . This observation excludes any possibility of a heuristic based on sampling any subset of the data; any useful heuristic *must* examine all the data.

However, if the heuristic is examining *all* the data, then it is at least  $O(n)$ , but DRAG itself is only  $O(nL)$  in the best case. There is very little room for an efficient and effective heuristic. Moreover, it seems unlikely that any heuristic algorithm that merely “slides” along the data could glean enough information to be useful. This is a task that requires some understanding of the relative distances between subsequences. Any heuristic would surely have to compute some significant number of such pairwise comparisons.

### 3.3 Introducing MERLIN++

The previous section largely reviewed the recently introduced MERLIN algorithm (Nakamura et. al. 2020). Here we introduce a novel optimization that can dramatically accelerate MERLIN. For clarity, we will refer to the new algorithm as MERLIN++ .

#### 3.3.1 The opportunity to accelerate MERLIN

Recall that in Phase II of DRAG (Table 2) we have a small set of time series subsequences, the candidate discords  $C$ . The size of set  $C$  depends on how good the  $r$

value was. However, using MERLIN, which provides good estimates for  $r$  (except possibly in the first iteration), the size of  $C$  tends to be between 40 and 80, even for the largest datasets we consider. Because this is a small number, relative to the full  $|T|-L + 1$  possible candidates that logically *could* initially have been the discord, MERLIN is much faster than STOMP or SCRIMP which must consider all  $|T|-L + 1$  possible candidates (albeit in a very efficient manner).

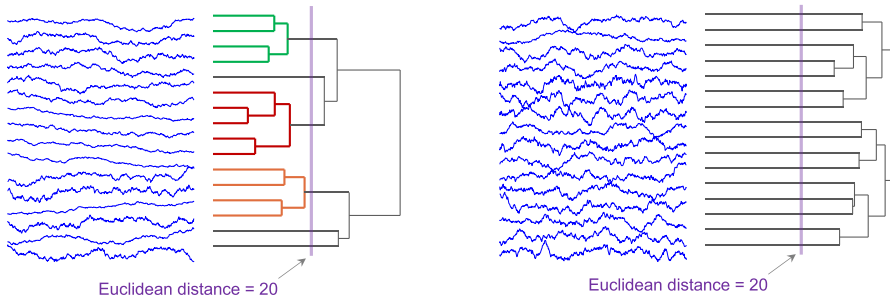
However, the discords refinement algorithm outlined in Table 2 is typically responsible for at least 90% of the overall time required by MERLIN. This algorithm basically consists of repeated nearest neighbor searches of the candidates set. If we could index  $C$  to allow faster search, we could potentially speed up MERLIN. However, indexing time series is a notoriously difficult problem, especially for longer  $L$ . While there are many techniques that address this problem (see (Ding et al. 2008) and the references therein) they all rely on an assumption that is not true in our setting. Classic time series indexing approaches assume that the data to be indexed can be implicitly or explicitly clustered into many Minimum Bounding Rectangles (MBRs) or similar (typically after using dimensionality reduction with DFT, DWT, PAA, SAX etc. see (Keogh et al. 2001)). Most of the speed-up achieved is by admissibly pruning some of the MBRs from consideration (Keogh et al. 2001). However, the set of candidates  $C$ , produced by the candidate selection algorithm (Table 1) and searched in discords refinement algorithm (Table 2) strongly violates that assumption. This set is not clusterable. In order to be clusterable, some candidates must be similar to other candidates. However, here that is not true by definition. If two subsequences were mutually similar, closer than  $r$  apart, then the conditional in line 5 of Table 2 would have ensured that only one of them was included in  $C$ . Thus, we can be sure that the candidate set consists of time series that are all mutually at least  $r$  apart. In a sense, the candidate generation algorithm is very similar to a diversity maximization algorithm.

To illustrate this, we did the following. We created a random walk time series of length 100,000, from which we plan to find discords of length 512. We then ran DRAG with the optimal value of  $r$  (To find the optimal value of  $r$ , we ran DRAG twice. With the first run we discovered the true discord score, which was 21.39054. From that we can subtract an epsilon to get the optimal value for  $r$ , i.e.,  $r = 21.39053$ ).

In Fig. 7.right we show the single linkage clustering of sixteen items in the candidate set in the second run of DRAG. Sixteen is about as small as a candidate set could ever be. Recall that we were able to make it so small by only using the optimal value  $r$ , which we discovered in this case by “cheating”.

In Fig. 7.left we show the clustering of sixteen random subsequences from the same dataset. Even though these are randomly extracted subsequences, they still exhibit some clusterability that could be exploited by various hierarchical indexing techniques. The contrast with Fig. 7.right is stark. The set  $C$  is essentially the optimally difficult dataset for hierarchical indexing techniques (Ding et al. 2008; Keogh et al. 2001). The index building algorithm will either place all sixteen objects into their own individual MBR or will place all sixteen objects into a single MBR. In either case, zero speed-up will be achieved, and the overhead of dealing with the index will ensure that the search is actually slower than simple sequential search.





**Fig. 7** (Left) Sixteen random subsequences of length 512 extracted from a random walk of length 100,000. In spite of being randomly extracted, the data does have some clusterable structure, illustrated by the colored subtrees, that can be exploited by indexing algorithms. (Right) The sixteen elements contained in set  $C$  on the same dataset. Note that there are no natural clusters in this dataset. This is true by the definition of the algorithm that created this set  $C$ . The annotation at 20 is an arbitrary marker to allow the reader to have a commensurate point in both clusterings

As a result, we have an issue. We have a dataset that could greatly benefit from faster search but is pathologically hard to index using standard indexing techniques (Ding et al. 2008; Keogh et al. 2001).

### 3.3.2 Accelerating MERLIN

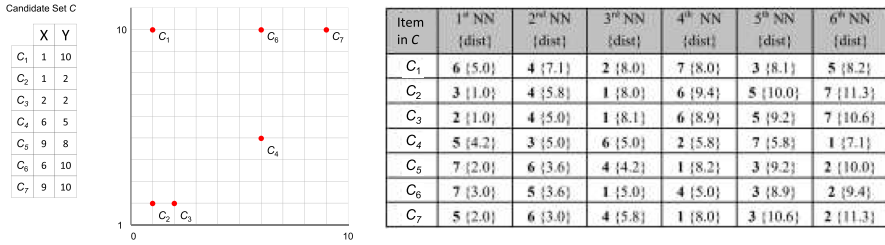
In spite of our pessimistic assessment in the previous section, there is a little-known<sup>4</sup> indexing technique called Orchard's algorithm that is perfectly suited to the task-at-hand (Orchard 1991; Wang et al. 1999). While Orchard's algorithm is currently used in some specialized domains such as vector quantization, it has not received much attention in the data mining community because of a serious flaw; it requires quadratic space and time. This would be the death knell for most applications; however, the size of set  $C$  is almost always less than 100 (given a reasonable value for  $r$ ), thus space and time are not an issue.

The algorithm is discussed in detail in (Orchard 1991), however for completeness we will give a brief review. We will use the toy example shown in Fig. 8 for exposition.

As Fig. 8.right shows, the Orchard's index is very simple. It is just a table containing sorted lists of nearest neighbors for each item.

The basic intuition behind Orchard's algorithm is to prune non-nearest neighbors based on the triangular inequality. Suppose we have our dataset  $C = \{C_1, C_2, \dots, C_7\}$ , and we want to find the nearest neighbor of a query  $q$ . Further suppose  $C_i$  is the nearest neighbor found in  $C$  thus far. For any unseen element  $C_j$ , we know it cannot be the nearest neighbor if:  $dist(C_j, q) \geq dist(C_i, q)$ . Given that we are dealing with a metric space (recall we are using the Euclidean distance), the triangular inequality tells us that:  $dist(C_i, C_j) \leq dist(C_i, q) + dist(C_j, q)$ .

<sup>4</sup> Orchard's algorithm is obscure enough that when googling for "Orchard's algorithm", almost half the results are about algorithms for literal fruit processing.



**Fig. 8** (Left) A dataset containing seven two-dimensional datapoints. We can gain a better intuition for the dataset by plotting it (middle) with a scatterplot. (Right) The Orchard’s index is simply a table where the first cell of each row represents an object in our dataset, and the remaining six cells in the row contains the six remaining objects, sorted by their distance to the object in the first cell

Combining these two facts, we can derive the fact that if  $dist(C_i, C_j) \geq 2 \times dist(C_i, q)$  is satisfied, then  $C_j$  can be admissibly pruned, since it could not be the nearest neighbor to  $q$ .

Orchard’s algorithm is simply a way to efficiently exploit this observation, using the information stored in the table shown in Fig. 8.right. The algorithm begins by randomly picking one of the rows as its “guess” as to the nearest neighbor of  $q$ . Then the algorithm begins to move across the row inspecting the remaining items until one of three things happen:

- The end of the row is reached. This case would mean that no speedup was achieved. It is logically possible, but essentially never happens on real data.
- The next item on the list has a value that is more than twice the current *best-so-far*. The algorithm can terminate. The current *best-so-far* is the true nearest neighbor.
- The third possibility is that the item in the list is closer to the query than the current *best-so-far*. In that case the algorithm simply jumps to the head of the row associated with this new nearest neighbor to the query and continues from there.

If the last possibility occurs, we could visit an object more than once, however we can cache the results of any calculations to avoid redundant calculations.

Nearest neighbor search with Orchard’s algorithm in the best case is  $O(1)$  and in the worst case  $O(|C|)$ , however empirically the average time is  $O(\log(|C|))$  (Wang et al. 1999).

Our use of Orchard’s algorithm offers a unique opportunity for an optimization. Recall that the search begins with randomly picking one of the rows as a “guess” as the nearest neighbor. It is known that the speed of the algorithm is somewhat sensitive to this guess. Naturally, the best guess would be the true nearest neighbor. To test this sensitivity, we twice ran a million random queries on the dataset shown in Fig. 7.right, indexed with Orchard’s. In the first run, we used a truly random guess. In the second run, we let the algorithm “cheat” by always making the *best* possible guess (which we had recorded during the first run). We discovered that using the “magic” guess made the total time about 2.4 times faster.

Surprisingly, our unique problem setting lets us actually make this optimal guess more than 95% of the time. Normally any index might be expected to answer multiple *independent* queries. But our queries come from a linear scan of all the subsequences

of  $\mathbf{T}$ , they are not independent. In general, any two consecutive subsequences  $\mathbf{T}_{i,L}$  and  $\mathbf{T}_{i+1,L}$  are very similar, and very likely to have the same nearest neighbor in  $C$ . Thus, we can set a policy for  $\mathbf{T}_{i+1,L}$ : use the (*just discovered*) nearest neighbor of  $\mathbf{T}_{i,L}$ . We call this augmented version of Orchard's algorithm Orchard++, and the modified version of MERLIN that exploits it, MERLIN++. It is worth restating that the output of MERLIN and MERLIN++ are identical, only their speeds differ.

### 3.4 Defeating MERLIN++

There are two circumstances where MERLIN++ can dramatically fail. Fortunately, there are trivial fixes.

If there is a constant region longer than  $MinL$ , then our attempt to z-normalize before computing the Euclidean distance will divide by zero. However, it is trivial to monitor for and report or ignore such regions. Depending on user choice, such regions may warrant flagging as an anomaly or not. For example, in hospital settings the data is replete with constant regions, due to disconnection artifacts during bed transfers etc. In contrast, a constant region in an insertable cardiac monitor (pacemaker) is almost certainly battery failure or heart-failure, in either case warranting an alarm.

As noted in our discussion of related work, another way MERLIN++ could fail is if the anomaly happens twice, and essentially looks the same both times. This has been called the "twin freak" problem. This can be solved by changing the first nearest neighbor (Definition 4) to the  $k$ th nearest neighbor. However, empirically this rarely seems to be an issue. For example, a paper that wanted to show an anomaly detection method that was invariant to "twin freaks" had to resort to copying and pasting data to contrive the situation (Bu et al. 2009), but they were unable to find a *natural* example. Note that Electrocardiograms (ECG) are often used to test anomaly detection algorithms (Boniol et al. 2021), and it is possible that an arrhythmia can be both intermittent *and* conserved in shape when it occurs. Thus, this is at least one dataset where twin freaks may be an issue. However, if we are monitoring an ECG in real time, the *first* occurrence of an arrhythmia will be unique, and therefore flagged. If we are examining an archival ECG, we can simply emulate real time monitoring by only allowing each candidate subsequence to be compared with subsequences to its left (i.e., observed earlier in time).

### 3.5 Handling the near constant region problem

There is a subtle and unappreciated issue that will cause MERLIN++ and other discord-based algorithms to produce false positives on many datasets. The issue is that many datasets have near constant, but not perfectly constant regions. Before explaining why this is an issue, let us explain why near constant regions are so prevalent:

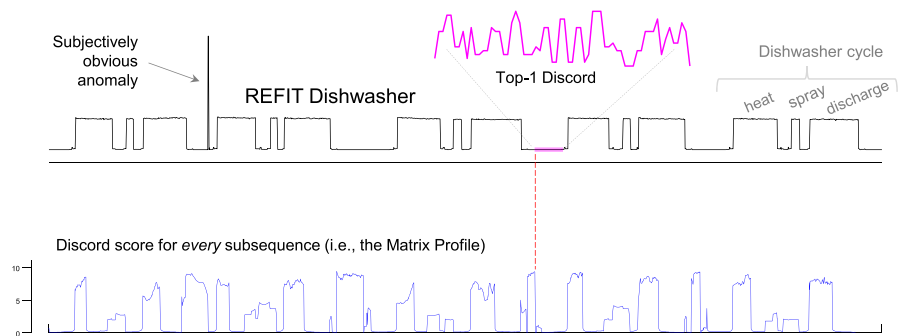
- Electrical power demand time series typically have many near constant regions. Why not *perfectly* constant regions? Virtually all modern appliances consume a trickle of power in standby mode. When viewed on a plot that includes a heavy demand from a heating element, these regions may appear to be perfectly constant. However, if we zoom in, as in the Top-1 discord shown in pink in Fig. 9, we can

see that there are tiny fluctuations. These regions are easiest to see on individually monitored appliances, as in our example in Fig. 9. But even on a whole house meter, we typically see long near constant regions when the occupants are asleep.

- Many long-term (hours or longer) medical recordings have near constant regions. In most cases they reflect a disconnection artifact, may be inadvertent or a deliberate disconnection to facilitate patient transfer or an intervention. Some such devices will report a perfectly constant signal when disconnected. However, because such devices often need to be exquisitely sensitive, most will always report tiny fluctuations, due to Johnson–Nyquist noise.
- In recent work, Abdoli et al. (2020) report on a project to record the motions of chickens, 24 h a day, for the entire bird’s life. While most of this data is highly dynamic, with the birds pecking, preening, and dustbathing throughout the day, all birds had moments of extreme stillness lasting at least several seconds. Naturally, these are not perfectly constant, as the sensor is sensitive enough to pick up vibrations from the bird’s heart.

Why are such near constant regions so problematic? Because, as hinted at in Fig. 9 they are essentially guaranteed to be the top-discord.

The reason nearly constant regions have such a high discord score is because we are working in  $z$ -normalized space. As the zoomed-in region shown in pink in Fig. 9<sup>top</sup> demonstrates, after  $z$ -normalization, such regions “explode” to become something strongly resembling a random vector. It is well known that in high-dimensional spaces, random vectors have two properties. They are almost the same distance to any other vector (including other random vectors), and that distance is a large fraction of the maximum distance (the diameter) of the space (Beyer et al. 1999; Thirey et al. 2015). The obvious implication of this is shown in Fig. 9<sup>bottom</sup> where all the subsequences corresponding to nearly constant regions are about the same distance, approximately nine, from their nearest neighbor.



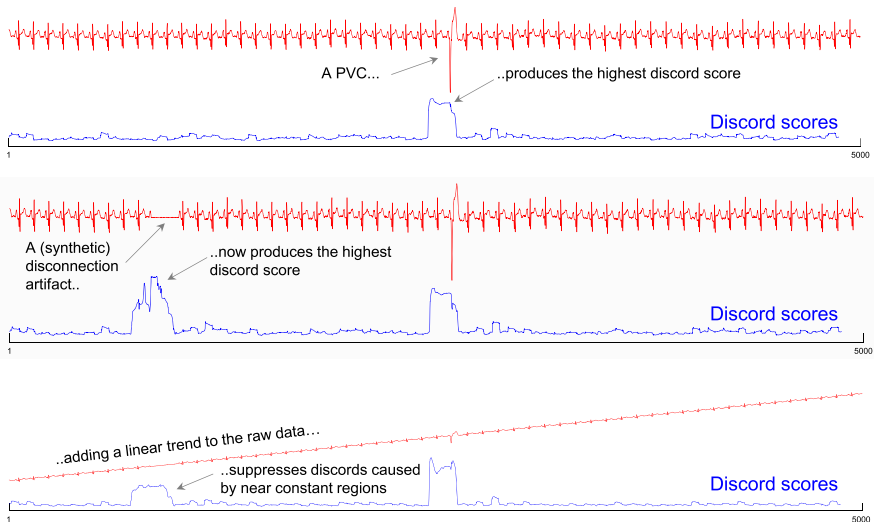
**Fig. 9** *Top* A snippet of electrical demand from a house in England (Murray et al. 2015). The data consists mostly of five cycles of a dishwasher, with one demand surge that most people would agree is the only obvious anomaly. *Bottom* A plot of the discord score for every subsequence of  $L = 80$  indicates that many regions have an approximately equal discord score of about nine, and that the obvious anomaly is not scored the highest by discords

Before we introduce our fix for this, it is worth dismissing the idea of *not* performing z-normalization on each subsequence. The need for z-normalization in time series has been made forcefully in (Thanawin et al. 2013) and elsewhere. In brief, without z-normalization the mean value of a subsequence dominates the distance function, and the actual shape becomes almost irrelevant. This is known to produce pathologically poor results in clustering, classification, and motif discovery (Thanawin et al. 2013), and it also devastates anomaly detection.

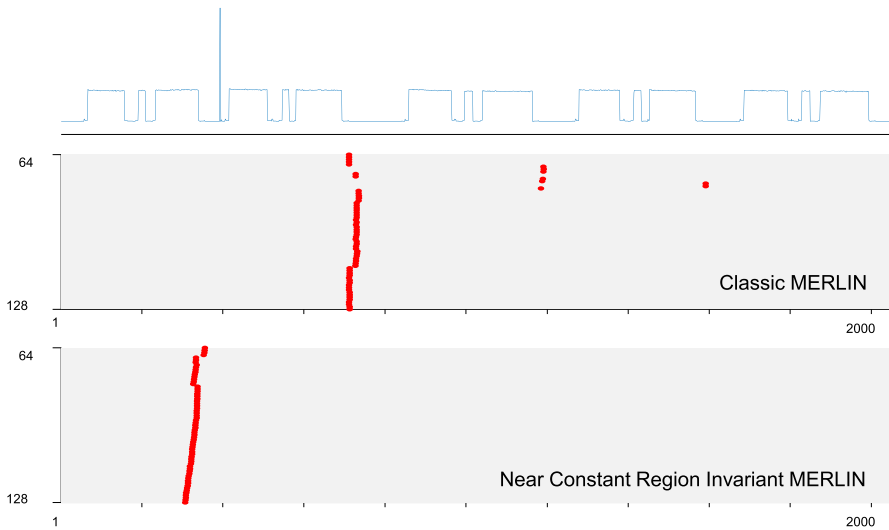
It is important to note that we cannot simply detect and *remove* near constant subsequences. Figure 9 makes that obvious, for example a dishwasher cycle with an extra spray cycle would be an anomaly, but still be comprised of several near constant regions. With domain knowledge, near constant regions in time series with steady baselines may be easily ignored by simply ignoring subsequences with maximum amplitudes below some  $\varepsilon$ . Though, a time series may not be well understood or contain a wandering baseline. Additionally, ignoring discord scores for subsequences with maximum amplitudes fluctuating around  $\varepsilon$  will result in a jittery discord score. Rather than entirely ignore near constant regions, we suppress them in proportion to their amplitude difference relative to the rest of the time series.

Our solution to this problem is simple, we can run MERLIN++ not on the original data, but on a transformation of the data. To be clear, this transformation is an internal representation only, and is never shown to the user. Our somewhat unintuitive solution is to suppress near constant regions by adding a large amplitude trend to the time series.

Consider Fig. 10.*top* which shows the electrocardiogram (ECG) of a 51-year-old



**Fig. 10** *Top* An ECG time series with an obvious anomaly of a PVC, which is easily discovered by discords. (*middle*) If we add a synthetic disconnection artifact it will produce the highest discord score. *Bottom* Adding a linear trend to the data *before* discord discovery is performed suppresses high discord scores produced by near constant artifacts



**Fig. 11** (Top) The result of running classic MERLIN on the electrical demand dataset. (Bottom) The result of running classic Near Constant Region Invariant MERLIN++ on the electrical demand dataset

male, with a Premature Ventricular Contraction (PVC) as an obvious anomaly at about the half-way point. In the top panel we show the discord score at every location. Clearly discords have no difficulties here. In Fig. 10.middle we add a near constant region to emulate a disconnection artifact. The accompanying discord scores shows that this region now results in the highest discord scores. Finally, in Fig. 10.bottom we show the discord scores computed on the data after we have added a linear trend. This has suppressed the anomaly caused by the near constant region and the PVC is once again the top discord.

The entire time series is first z-normalized so that the standard deviation is close to one. With this normalized time series, a step of one is added to each time point. A time series  $\mathbf{T}$  will have a value of  $i$  added to the time index  $\mathbf{T}_i$ . As shown in Fig. 10.bottom this is visually unintuitive as the time series becomes dominated by the linear trend. A vector with lower magnitude noise relative to the whole time series will be transformed into a trend with low amplitude noise once z-normalized, which will be among candidates with lower Euclidean distance.

In Fig. 11 we show that this technique solves the problem noted on the electrical demand dataset introduced in Fig. 9.

This feature has been bundled into the public release of MERLIN++ (Nakamura 2021).

## 4 Empirical evaluation

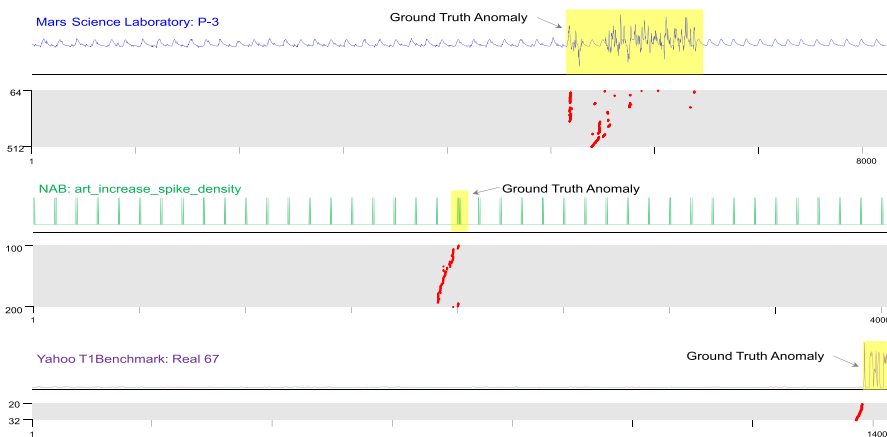
We begin by stating our experimental philosophy. We have designed all experiments such that they are easily reproducible. To this end, we have built a webpage that

contains all datasets, code and random number seeds used in this work, together with spreadsheets which contain the raw numbers (Nakamura 2021). This philosophy extends to all the examples in the previous section.

#### 4.1 Metrics of success and the unsuitability of benchmarks

There are now a handful of benchmark datasets in the literature. We have already considered (a subset) of them in Figs. 1, 2, and 3 and we will consider more below. However, we believe that the reader should be somewhat skeptical of research efforts that report *only* summary statistics on these datasets. There are at least two reasons for such skepticism.

- Consider the NYC Taxi example which is part of the NAB benchmark (Ahmad et al. 2017). This dataset is labeled as having five anomalies, but as Fig. 2 shows, this dataset has at least twice that number of anomalies. For example, the benchmark does not list the daylight-saving time anomaly, which is arguably the most visually jarring anomaly in the dataset. Any algorithm that does find this anomaly will be penalized as having produced a false positive. In (Nakamura 2021) we show more examples of mislabeled benchmark data.
- A large fraction of the benchmark datasets contain anomalies that are so obvious that they are trivial to detect. For example, consider Fig. 12, which shows examples from the Mars Science Laboratory (Hundman et al. 2018), NAB (Ahmad et al. 2017) and Yahoo benchmarks. It is hard to imagine any reasonable algorithm failing to find such anomalies. Even if the benchmark data *also* includes some challenging anomalies, counting success on these trivial problems can artificially inflate metrics of success such as ROI curves, giving the illusion of progress. See Appendix A for more information and examples.



**Fig. 12** Examples from the three main anomaly benchmark datasets that we regard as too simple to be informative for algorithm comparison. *Top* From the NASA benchmark (Hundman et al. 2018). *(Center)* From the NAB benchmark (Ahmad et al. 2017). *Bottom* From the Yahoo Benchmark

For the reasons above, we think that a direct *visual* summary of the output of a proposed anomaly detection algorithm on diverse datasets can offer the reader the most forceful summary of the algorithm's strengths and weaknesses (although we must be careful to avoid attempting “*proof-by-anecdote*”). For that reason, we have chosen to show twenty diverse examples below.

It is important to note that our discussion of some issues with the benchmark datasets should *in no way* be interpreted as criticism of those who have made these datasets available. These groups have spent tremendous time and effort to make a resource available to the entire community and should rightly be commended. It is simply that we must be aware of the limitations of metrics reported on them without visual context.

As such, we have endeavored to have many such examples in this work. In particular, before performing conventional experiments to compare MERLIN++ to the state-of-the-art, we begin with some case studies that give the reader an appreciation of the kind of subtle anomalies that MERLIN++ can discover.

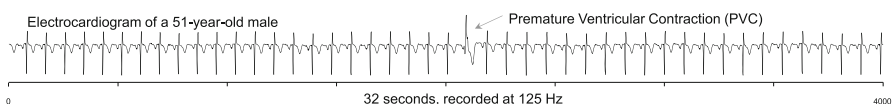
## 4.2 Discovery of ultra subtle anomalies

Virtually all anomaly detection benchmarks in the literature contain anomalies that also yield to casual visual inspection. Of course, this does not mean that algorithms that can detect such anomalies are of no utility. Human inspection, especially at scale, is expensive. Nevertheless, it is interesting to ask if we can detect *very* subtle anomalies, that would defy human inspection. However, this seems to beg the question, how can we know if a time series contains such ultra-subtle anomalies?

We propose the following experiment to allow us to obtain ground-truth subtle anomalies. Consider Fig. 13, which shows the electrocardiogram (ECG) of a 51-year-old male, with an obvious anomaly at about the half-way point. The anomaly is so obvious that surely *any* algorithm could discover it.

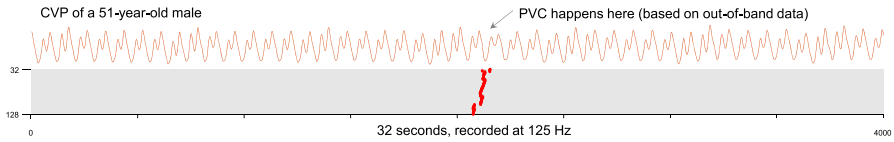
However, suppose we consider *only* the Central Venous Pressure (CVP) data, which was recorded in parallel. The ECG is an electrical signal, whereas the CVP is a mechanical signal, the blood pressure in the *venae cavae*. Moreover, because the CVP reflects the amount of blood returning to the heart, the elasticity of the blood vessels tends to dampen out any irregularities in the heartbeat. As Fig. 14 shows, the Premature Ventricular Contraction (PVC) anomaly is not visually apparent in the CVP, yet MERLIN++ clearly indicates at the correct location.

Note that our inability to see the anomaly in Fig. 14 should not be attributed to the small size of the figure (the reader is invited to see a larger reproduction here (Nakamura 2021)) or our lack of medical experience. Dr. Greg Mason, an experienced clinician with decades of experience, could not detect this anomaly.

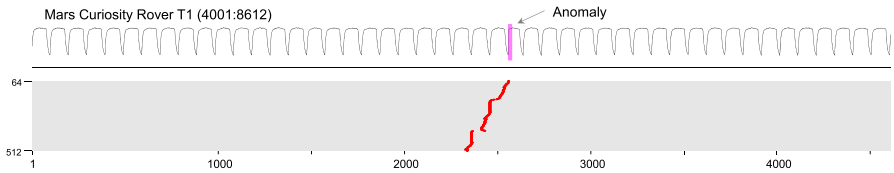


**Fig. 13** An ECG signal with an obvious anomaly (a PVC)





**Fig. 14** A CVP signal recorded in parallel with the ECG shown in Fig. 13 does not show visual evidence of an obvious anomaly caused by the PVC, yet MERLIN++ clearly indicates its presence



**Fig. 15** A signal from the Mars Curiosity rover was annotated as having an anomaly from 2550 to 2585 (pink bar) (Hundman et al. 2018). While the cause of the anomaly is unclear, MERLIN++ has no difficulty finding it (Color figure online)

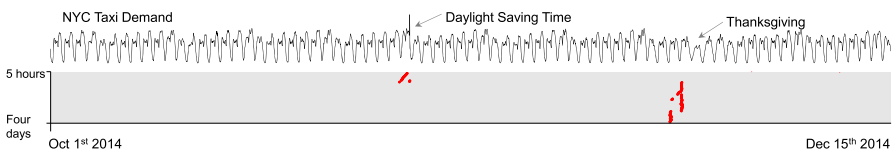
To see that this was not pure luck, let us consider a dataset from a totally different domain with a similarly subtle anomaly. In Fig. 15 we show a snippet of data from the Mars Science Laboratory (MSL) rover, Curiosity (Hundman et al. 2018).

In the paper that introduced this dataset, the authors introduced a LSTM network that also found this anomaly (Hundman et al. 2018). However, to do so, they required training data and the careful setting of *eight* parameters. In contrast, MERLIN++ finds this subtle anomaly with no training data and only the weakest of hints as to the anomaly lengths (*MinL* and *MaxL*) to consider.

### 4.3 Anomalies at different scales

In this section we anecdotally demonstrate the utility of being able to discover multiscale anomalies. We simply wish to show that anomalies that differ by at least an order of magnitude can exist even in quotidian datasets. We begin by revisiting the NYC Taxi demand dataset shown in Fig. 2. In Fig. 16 we show a subset of the data with just the top-1 motif of every length from 5 h to four days.

While the daylight-saving anomaly directly affects only two hours, the shape of these two hours is only usual in the context of the few hours that surround them. Similarly, while Thanksgiving is somewhat unusual in its lower passenger volume and lack



**Fig. 16** A subset of the Taxi demand dataset shown in Fig. 2, shown with all discords the range of 5 to 96 h

of a rush hour peak of people leaving the city after work, a somewhat similar pattern to this also happens on the weekends. However, in the context of being surrounded by normal days, Thanksgiving is unusual. The discords of up to four days long discovered by MERLIN++ in Fig. 16 reflect this.

We also considered a similar but much longer dataset of passenger volume at the Taipei Xinjian District Office metro station. We searched from ten hours to ten days. Over this enormous range of scales, only seven distinct anomalies are discovered, Fig. 17.bottom shows four of them. Note that some of the anomalies have natural causes (weather events), and some are cultural artifacts such as Chinese New Year.

It is easy to find other datasets that reflect daily patterns of activity. As hinted at in Fig. 18.top, the city of Melbourne has released almost a decade's worth of pedestrian traffic volume from various sites in the city (Pedestrian Counting System 2013).

While there is good spatial coverage, the temporal resolution is very low at just one datapoint per hour. Because of this, like most of the many research groups that explored this data resource, we originally only searched for anomalies of length days or weeks (Doan et al. 2015). However, as Fig. 18, hints at, using MERLIN++ to free ourselves

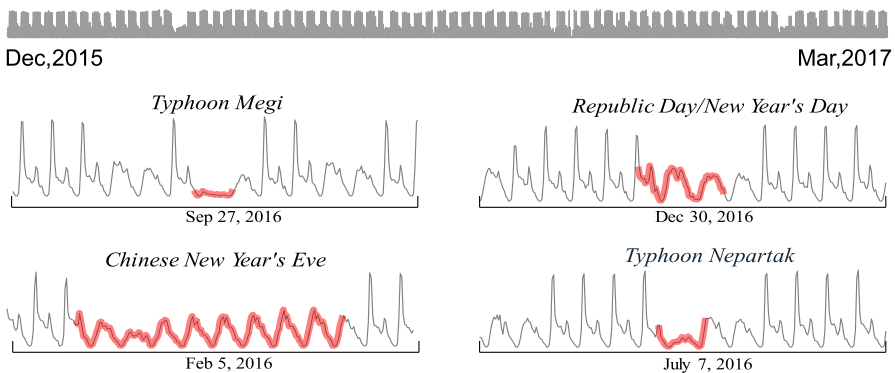


Fig. 17 Top Passenger volume at a Taipei metro station. Four of the anomalies discovered are shown in context

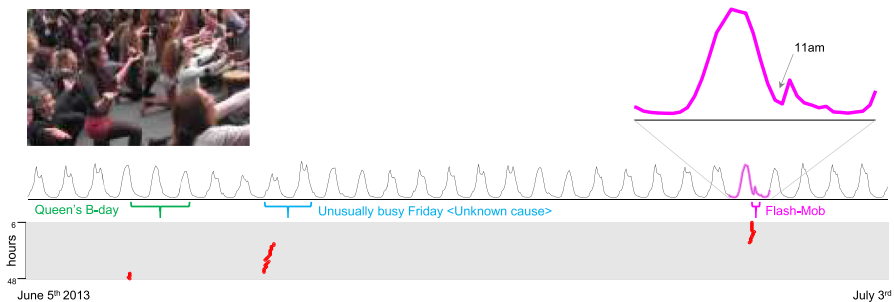
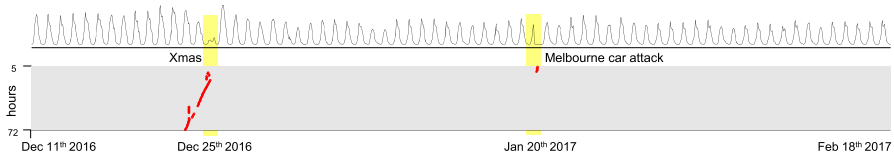


Fig. 18 Bottom A month of pedestrian traffic volume on Bourke Street Mall in Melbourne. Top the shortest anomaly discovered is semantically meaningful, it corresponds to a flash-mob dance performance (video at (McRae 2013)) that restricted traffic for about ten minutes



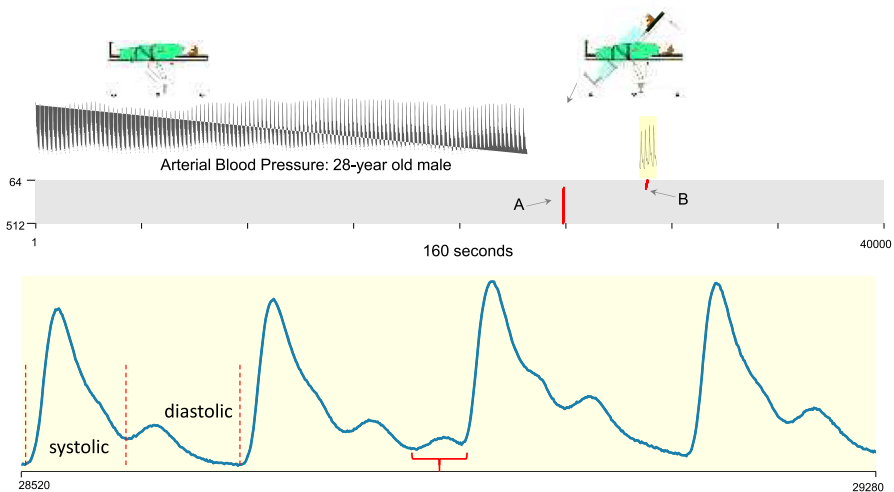
**Fig. 19** Two months of pedestrian traffic volume on Bourke Street Mall in Melbourne. The anomaly for Xmas is to be expected, but what caused the short anomaly on Jan-20-2017?

from assumptions about possible anomaly duration allowed us to find unexpectedly short anomalies.

Given this ability to find motifs at all scales, we begin to find unexpected anomalies everywhere. Three years after the flash-mob happened, we discovered another short and subtle anomaly on the same street, as indicated in Fig. 19. With a little investigation we realized it corresponded to a car attack in which an individual deliberately drove at pedestrians, killing six and injuring twenty-seven (Wikipedia 2022).

The ability to find anomalies without specifying the length in advance can occasionally produce surprises. We tested MERLIN++ on a dataset that we are very familiar with, having considered it for other tasks (Imamura, et al. 2020). The dataset, shown in Fig. 20 is the Arterial Blood Pressure (ABP) of a healthy twenty-eight-year-old man undergoing a tilt table test (Heldt et. al 2003). Because we know exactly *when* the table was tilted, we decided to use this as a test to see how well MERLIN++ works in the face of wandering baseline and periodicity drift.

We tested  $L$  in the range 64 to 512. As Fig. 20.*bottom* shows, the tilt event does indeed show up as event **A**. However, in the range of 64 to 145, a different anomaly, event **B** is evident. A zoom-in shows that event **B** is unusual in that it has a second



**Fig. 20** *Top* The ABP of a healthy twenty-eight-year-old man undergoing a tilt table test, annotated by the two events found by MERLIN++ . *Bottom* Zoom-in of the region corresponding to Event B

“bump” in the diastolic region. The first bump, the dicrotic notch, is the only increase normally expected in this phase. Dr. Greg Mason, a Clinical Professor of Medicine at UCLA was kind enough to explain this finding: “*Baroreceptors are sensors in the heart that sense pressure changes by responding to change in the tension of the arterial wall. When a person has a sudden drop in blood pressure, for example standing (or being tilted) up, the decreased blood pressure is sensed by baroreceptors as a decrease in tension therefore will decrease in the firing of impulses. This can take a few seconds, what Event B shows is the baroreceptor response suddenly “kicking in” to decreasing parasympathetic (vagal) outflow.*”

To summarize this section, the ability to find motifs without specifying the scale is useful not only because it removes a parameter, but because we have less opportunity to project our assumptions on the task-at-hand, we can often find completely unexpected anomalies and novel behavior in the data.

#### 4.4 Scalability

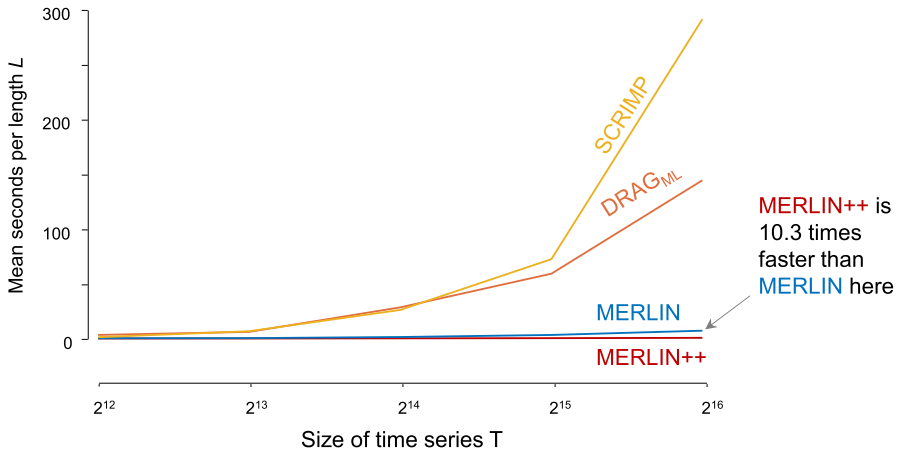
To demonstrate the scalability of our algorithm, we compare it to the two most cited exact discord discovery algorithms SCRIMP (Yeh et al. 2016) and DRAG (Yankov et al. 2008). There are at least a dozen approximate discord discovery algorithms, but it would greatly confuse the issue to have to compare on time, *and* precision/recall.

In a sense, this comparison is a little unfair to SCRIMP, which discovers not only the discords, but also motifs. Nevertheless, it is a very scalable algorithm because it is implemented in a way that makes it constant in the length of the subsequences. We used the latest version of the code available from the author’s website (Yeh et al. 2016), disabling the GUI interface, which required significant time overhead.

We also wish to test the effectiveness of our method to set the value of  $r$  for MERLIN or MERLIN++ by sharing information across different values of  $L$ . To do this, we implemented the method for setting  $r$  suggested in (Yankov et al. 2008), which we independently rerun for every value of  $L$ . This algorithm is denoted as DRAG-multilength, or DRAG<sub>ML</sub>. Note that DRAG<sub>ML</sub> differs from MERLIN/MERLIN++ only in how  $r$  is set. For MERLIN/MERLIN++ , (apart from the first iteration) the (slightly adjusted) value of  $r$  from iteration  $i-1$  is used to set the  $r$  value for iteration  $i$ . In other words, any observed speedup in this experiment tells us how useful this information sharing between iteration was.

The time needed for SCRIMP is independent of the value of  $L$ . However, the time needed for the two other algorithms depends on the data. The best case would be a dataset like the one shown in Fig. 12.top, a mostly repetitive time series with a dramatic discord that is very far from its nearest neighbor. To avoid such bias, we will use the worst-case dataset for MERLIN/MERLIN++ , *random walk*. For such data, the top-1 discord is only slightly further away from its nearest neighbor than any randomly chosen subsequence, meaning that the candidate set built in Table 1 grows relatively large even if given a good value for  $r$ .

Note that SCRIMP’s performance is independent of the structure of the data, but because the other algorithm’s performance does (weakly) depend on it, we averaged over ten runs. Figure 21 shows the results of datasets with lengths ranging for  $2^{12}$  to



**Fig. 21** The scalability of MERLIN, MERLIN++, DRAG<sub>ML</sub> and SCRIMP in the face of increasingly large datasets

$2^{16}$ .

For short time series, all algorithms perform similarly, but as the time series grows longer, SCRIMP's quadratic complexity begins to show. While MERLIN's *first* run (for  $L = \text{Min}L$ ) is no faster than DRAG, its subsequent runs are greatly accelerated by the predicted value of  $r$ , and the amortized cost is about twenty-one times faster by the time we consider time series of length  $2^{16}$ . To put these numbers in context,  $2^{16}$  datapoints is about 18 min of data recorded at 60 Hz. Suppose we suspected that there were anomalies of length 1 s in our data, but we wanted to bracket our search with every value for 30–90 datapoints. This would take MERLIN just 7.1 min, faster than “real-time”, and take MERLIN++ just 42.3 s.

#### 4.5 Can MERLIN/MERLIN++ be further accelerated

For a given dataset, with a given *MinL* and *MaxL*, the time taken by MERLIN or MERLIN++ depends only on the initial estimate for  $r$ . However, the design of these algorithms means that we do not critically depend on obtaining a high-quality initial estimate, as these only affect the first iteration when the subsequences are all *MinL*, a relatively short length. To see how insensitive MERLIN is to this step, we conducted the following experiment. We consider the Dutch Factory dataset, which is of length 35,040, reflecting a year of data sampled every fifteen minutes (Keogh et al. 2005). We ran MERLIN to discover discords from length 96 (one day) to 192 (two days). This took 34.3 s. We ran the algorithm again, but this time allowing it to “cheat”. From our first run, we know that the true discord distance for *MinL* is 10.89740. Thus, in the second run, we primed MERLIN with the optimal value for  $r$ , an epsilon under the true discord value. Concretely, we replaced line 1 in Table 3 with ‘ $r = 10.897399$ ’. After doing this, the time taken only improved to 34.1 s.

We can generalize this experiment by allowing MERLIN to cheat at *every* iteration. In particular, we replaced the computed value of  $r$  in line 16 of Table 3 with the true discord value for that length, minus an epsilon. Even with this extraordinary advantage, the time requirement only improved to 31.3 s. Thus, it appears that our scheme for choosing  $r$  is close to optimal. We do not bother to show the discords discovered for this simple problem, in every case the top discord spans Xmas day, when the factory was closed, the subjectively obvious correct answer.

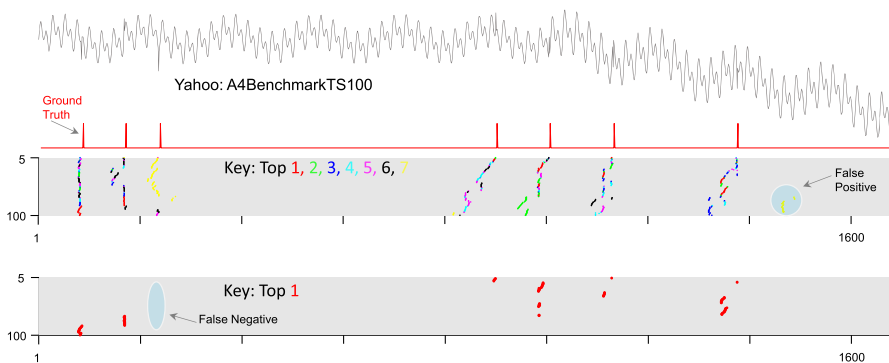
To summarize, there may be ways to speed up MERLIN by further speeding up the DRAG subroutine. The triangular inequality techniques used to generalize MERLIN to MERLIN++ , is one such example, and there may be other ideas that have not occurred to the current authors. However, there does not appear to be any room for improvement when it comes to the strategy for selecting  $r$ .

#### 4.6 First look at the yahoo! webscope benchmarks

In recent years, the Yahoo Webscope anomaly datasets have emerged as the de-facto benchmark for anomaly and changepoint detection. This diverse archive consists of 367 time series of various lengths in four different classes A1/A2/A3/A4 with class counts 67/100/100/100. While class A1 has real data from computational services, classes A2, A3, and A4 contain synthetic anomaly data with increasing complexity. We previously showed examples from this benchmark in Figs. 3 and 12.bottom.

Before presenting summary statistics on the entire archive, we will take the time to consider one example in detail. Because most of these datasets have multiple anomalies, this is an ideal opportunity to show the output of the top-K discords. In Fig. 22.top we show an example with seven anomalies.

We know that examples in this subset have *point* anomalies, so a smaller value of MaxL would be appropriate. However, we “stress test” our algorithm by considering unreasonably long discords up to length 100. Figure 22.center shows that had we considered only 5 to 64, we would have obtained perfect results. It is only when we



**Fig. 22** *Top* An example of a synthetic datasets from the Yahoo archive with seven anomalies, which are marked by the red binary vector. *center*) The result of running MERLIN++ to discover the top-7 anomalies. *Bottom* The result of running MERLIN++ to discover just the top-1 anomalies

consider MaxL for an unrealistic value of greater than 65, that we obtain a single false positive, and then only for the 7th discord. Another way to consider how effective MERLIN++ is here is to see how many of the seven anomalies we can detect if we only consider the single top-1 discord. As Fig. 22.bottom shows we would still detect six out of seven true positives and have no false negatives.

#### 4.7 Large scale results on the yahoo! webscope benchmarks

In this section we evaluate the entire collection of Yahoo S5 datasets (Laptev and Amizadeh, 2015). We do so with some reluctance, even though, as we shall see, we achieve state-of-the-art results on this dataset. In a recent paper, a subset of the current authors have forcefully argued that this dataset has multiple flaws that render any claims made using it somewhat suspect (Wu and Keogh 2021). These flaws, hinted at in Fig. 3 and in Appendix A, include the fact that almost all of the 367 datasets can be solved with very simple methods that can be implemented in a single line of code. Of course, the fact that a *human* intelligence can visually examine each dataset, and in a few seconds suggest a single line of code (something like “YahooA1R1 > 0.45” or “diff(YahooA1R32= = 0)”) does not necessarily mean that it will be trivial for *algorithmic* intelligence. However, it would be disingenuous of us to bask in our strong showing here.

To evaluate all Yahoo 367 datasets (Laptev and Amizadeh 2015), we need to define some criteria for correct anomaly detection. Below we explain our reasoning behind our choice for metric of success.

Note that a complete anomaly detection system must have two parts, (I) A *prediction* of the most likely location(s) to contain anomalie(s), and (II) an *evaluation mechanism* (often simply thresholding) to determine if those locations warrant being flagged as anomalies. In this work, we have mostly avoided a discussion of the second part, as it is moot unless we can robustly point to candidate anomalies. Also note that in many real-world applications, the second part is not needed. For example, an analyst might query: “Show me the top-five most unusual events in the oil plant in 2018”. Likewise, thresholds can often be learned with simple human-in-the-loop algorithms. In brief, the user can simply examine a sorted list of all candidate anomalies. The discord distance of the first one she rejects as “not an anomaly” can be used as the threshold for future datasets from the same domain. Thus, we argue that the first task is the most critical and most worthy of evaluation.

Some of the Yahoo datasets have an issue that confounds evaluation. In the example shown in Fig. 22, the anomalies are all well-spaced apart. However, in the example shown in Fig. 3 the anomalies are just two datapoints apart. It is hard to imagine critiquing an algorithm that called these two events a single anomaly. More generally, we must also consider the precision of the algorithm’s prediction of location. If an anomaly is located at say location 600, we should surely reward an algorithm that predicts 599 or 602. Thus, for simplicity, we reward any prediction that is no further off than  $\pm 1\%$  of  $|T|$  from the stated location. This does not significantly increase the default rate while allowing us to bypass the issues above.

Given these considerations, we proposed the following metric of successes, which we believe to be fair, transparent and reproducible. Each algorithm is tasked with locating the one location it thinks most likely to be anomalous (We removed the handful of examples that have no claimed anomaly). If that location is within  $\pm 1\%$  of  $|T|$  from a ground truth anomaly, we count that prediction as a success.

We compare to the LSTM method introduced in (Hundman et al. 2018), which is one of the most highly cited deep learning for anomaly detection papers in recent years. We used the authors own implementation, carefully tuning it as advised in (Hundman et al. 2018). We allow the LSTM to “cheat” by training on a subset of the test data, this gives LSTM an advantage, but as we will see, one we readily overcome.

For MERLIN++ , we set  $MinL = 3$  (this is the minimum possible value) and the  $MaxL = 20$  and recorded the median location of the 18 predictions as the algorithm’s single prediction. This is a sub-optimal policy for us if there are two or more anomalies of around that length but makes the evaluation simple.

Under this metric MERLIN++ had a recall of 80.0% and the LSTM had a recall of 58.3%. While this result is strongly in our favor, because of the data quality issues discussed above, we do not weigh it as heavily as the visual evidence presented in the many visual examples shown in this work.

#### 4.8 Results on the NASA benchmarks

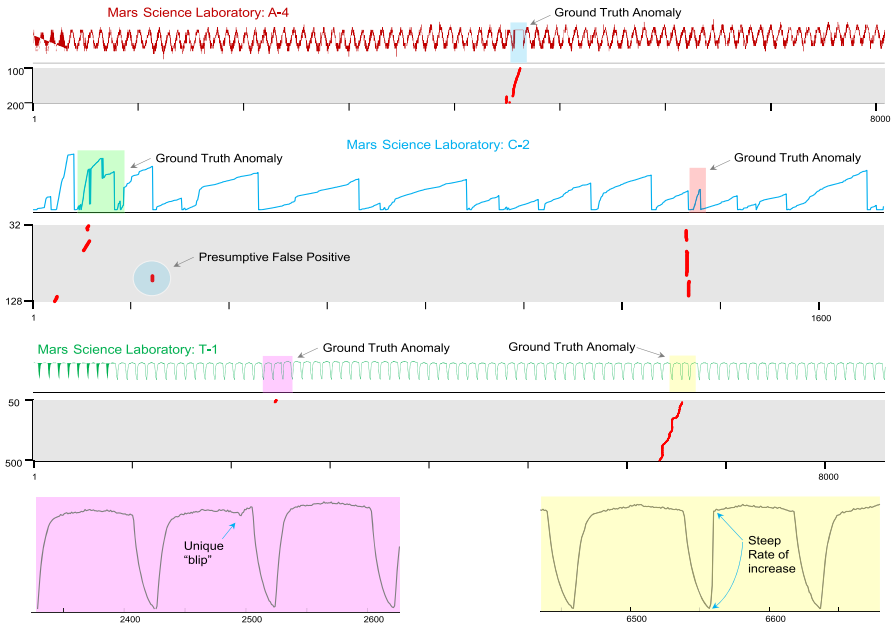
The NASA dataset (Hundman et al. 2018) has garnered significant attention in recent years, but as Fig. 12.top hints at, some of the tasks are trivial. In fact, that understates the case. Many of the anomalies consist of changes of variance/range by up to three orders of magnitude (examples A1, B1, D12, E7, P4, T3, etc.), and can trivially be detected by algorithms dating back to the 1950s (Page 1957) (see Appendix A for a concrete example of this).

In addition, for some of the examples, the labeled anomaly region comprises up to half the data (examples A7, D2, M1, M2 etc.), meaning that a random choice would have a better than even chance of being a true positive. To bypass this issue, we scanned all the datasets for examples that were not obviously solvable by the human eye in under five seconds. Excluding near redundant examples, only three datasets passed that test, the results of running MERLIN on them is shown in Fig. 23. Apart from a small region of a presumptive false positive in Fig. 23.center, we achieve perfect results. (We say “*presumptive*” because this dataset also has a handful of labeling omission errors, and we point them out at (Nakamura 2021)). Note that the bottom examples both had two anomalies, which we found with just the *single* top-1 discord of various lengths.

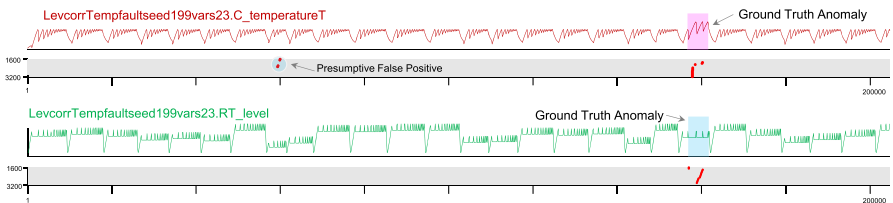
#### 4.9 Results on the gasoil benchmarks

Like the NASA dataset, we regard the Gasoil benchmark (Filonov et. al 2016) as being too easy to be interesting. Note that we are only making this claim with regard to *anomaly detection*, it may be useful for causality detection etc. In Fig. 24 we show the results of running MERLIN++ on two of the more challenging examples.





**Fig. 23** The results of running MERLIN++ on three diverse and most difficult examples from the NASA benchmark (Hundman et al. 2018). *Top* The single anomaly in A-4 is easily discovered. *center* The two anomalies in C-2 are discovered, but there may be a short region where we report a false positive. *Bottom* The two correctly detected anomalies are so subtle that we show annotated zoom-ins to explain them



**Fig. 24** The results of running MERLIN++ on two diverse difficult examples from the Gasoil benchmark (Hundman et al. 2018). *Top* The single anomaly in TempT is easily discovered, but there may be a small region where we report a false positive. *Bottom* The single anomaly in RT\_level is easily discovered

#### 4.10 Comparing MERLIN++ to state-of-the-art deep learning algorithms

As (Wu and Keogh 2021) have noted, most benchmarks in this space are too trivial to be interesting, and in any case suffer from mislabeling and other issues. Instead, here we consider the KDD Cup 2021 dataset consisting of 250 univariate time series. This archive was designed to be diverse, and to have a spectrum of difficulties ranging from easy to essentially impossible. Moreover, there is a detailed provenance for each of the datasets, giving us confidence that the claimed ground truth is at least highly plausible.

Moreover, the datasets include a vast range of domains, including cardiology, industry, medicine, zoology, weather, human behavior, etc.

The scoring metric suggested for this challenge is very direct. There are 250 datasets, each one is divided into a training section and a test section. There is a *single* anomaly in the test section, and the algorithms are tasked with discovering the location of that anomaly. If an algorithm's predicted location is within  $\pm 100$  datapoints of the true location, it is judged as *correct*, otherwise it is judged as *incorrect*. The 100-datapoint "wobble-room" in the scoring function is designed to be fair to the wide spectrum of datasets that might be applied to this task. Some algorithms have scoring functions that may peak at the beginning, the end, or the middle of an anomaly. Some anomaly evaluation functions would heavily penalize algorithms for their slight early or late bias. But if the authors of the proposed method *knew* this bias was going to be important when being evaluated, they could have easily adjusted this.

Given the above, we think that this archive is currently the best and fairest time series anomaly detection archive in existence. Table 4 shows the results of running several of the most highly cited anomaly detection algorithms on this dataset.

For the USAD, LSTM-VAE and AE methods, we started by using the original authors' suggested parameters settings, and spent a further hour on each, adjusting them in an attempt to improve their performance. When tuning them, we allowed ourselves to look at the test data, which may give us slightly optimistic results.

Note that Telemanom is marked with an asterisk. When running the experiment with Telemanom, we got "*Out of memory error*" on the longer examples. The code is opaque and there does not seem to be any simple way to fix this issue, so we did the following. We sorted all 250 datasets from shortest to longest and kept evaluating increasingly longer datasets until the out-of-memory failure. Telemanom failed at the 63rd smallest dataset. On the first 62 datasets it correctly found the anomaly for 29, giving it an accuracy of 0.468. Because the larger datasets seem to be much harder (they clearly have a much higher default rate), these results should be seen as optimistic for Telemanom. The time of 3.4 h reflects just the 62 smallest datasets, extrapolating the computation time to all the data is difficult, but it would take on the order of several days to finish all the datasets if the memory issue could be resolved.

For MERLIN and MERLIN++ , we made a slight modification to the code so that it only compared the subsequences in the test section to the subsequences in the train

**Table 4** Accuracy and time for various anomaly detection algorithms

Method	Accuracy	Train and test time
USAD (Audibert et al. 2021)	0.276	8.05 h
LSTM-VAE (Park et al. 2018)	0.198	23.6 h
AE (Audibert et al. 2021)	0.236	6.11 h
Telemanom (Hundman et al. 2018)	(0.468)*	(3.4 h)*
Matrix Profile SCRIMP (Yeh et al. 2016)	0.416	24.5 min
MERLIN (Nakamura 2021)	0.424	1.83 h
MERLIN++	0.424	14.5 min

section. We used the simple code shown in Appendix B to estimate the period  $P$  for each dataset, and set  $MinL = 0.5 \times P$  and set  $MaxL = 2.0 \times P$ . It is possible that there could be different “votes” for the location of the anomaly from different values of  $L$ . We simply took the majority prediction as the algorithm’s sole prediction.

These results offer evidence supporting our claim that Deep Learning methods do seem to underperform for the task-at-hand. In retrospect, perhaps this is not surprising. Deep Learning methods generally benefit from domains that have lots of labeled data. However, for most time series anomaly detection tasks, we typically do not have access to any positive examples.

These results do however beg the question as to why so many in the community *assume* Deep Learning works well for this task. Part of the answer may come from issues in flawed evaluation. Recently, three independent papers noted that the thoughtless use of precision and recall for the evaluation of time series anomaly detection can greatly inflate an algorithm’s apparent ability (Huet et al. 2022; Kim et al. 2022; Hwang et al. 2022). One paper (Kim et al. 2022) notes that with common evaluation schemes “*even a random anomaly score can easily turn into a state-of-the-art time series anomaly detection method*”, and another paper has a title that asks, “*Do you know existing accuracy metrics overrate time-series anomaly detections?*” (Hwang et al. 2022).

## 5 Conclusions and discussion

We have shown that *time series discords*, a simple, decades-old anomaly detection definition is surprisingly viable in many domains. In particular, it is *at least* competitive with the more complex deep learning methods, which require both significant amounts of training data and a plethora of parameters to be tuned.

Ahmed and Mahmood created an influential taxonomy of anomalies categorized into point anomalies, *contextual* anomalies and collective anomalies (Ahmed and Mahmood 2014). While we refer the reader to the original paper for the exact definitions, a review of this work shows that MERLIN++ was able to discover examples of each type. For example, Fig. 22 shows *point* anomalies that Yahoo embedded into a dataset. The Queen’s Birthday example in Fig. 18 is a classic example of a *contextual* anomaly. The shape of the day is smooth, missing the sharper features caused by typical weekday rush-hour commuting. Such days are not intrinsically rare, they happen on most weekends, but one only sees three such days in a row in the context of a three-day weekend. Finally, the anomalies shown in the Gasoil dataset in Fig. 24, are classic *collective* anomalies. This observation is suggestive of the generality of MERLIN++.

Some researchers in the community had noted the utility of discords, but waived off from using them, noting, “*discords are limited (because) a fixed length must be specified in advance, making it a clearly suboptimal approach for applications dealing with climate data events of varying length*” (Barz et al. 2017). The introduction of MERLIN++ removed this last barrier to adoption.

Finally, we would like to end with a note of caution for the anomaly detection research community. In recent years there has been an explosion of deep learning work on anomaly detection, including works that introduced or evaluated the four

benchmarks we consider in this work (Däubener et al. 2019; Hundman et al. 2018; Ahmad et al. 2017). However, there appears to be little evidence presented that the complexity of these approaches is warranted. Recall that for the most part we can reproduce or improve upon these results without even looking at the training data and using a method that is, by any reasonable standard, an order of magnitude simpler.<sup>5</sup> Please note that we do not doubt the utility of deep learning in general, or the ingenuity of these papers. However, we believe that the community needs to:

- Expand the list of strawmen it compares to. Perhaps half the benchmark problems can be solved by algorithms created in the 1950s (Page<sup>6</sup> 1957; Wu and Keogh 2021). See Appendix A for some examples. Simple ideas should be compared to, if the community is to justify complexity of deep learning approaches.
- Consider more challenging benchmarks. With this paper we have added a handful of more challenging benchmarks to the community's pool of benchmarks. Of course, these datasets may reflect our bias towards datasets that we know how to obtain. The community would greatly benefit from a distributed community-wide effort to produce anomaly detection benchmarks.
- Directly *visualize* algorithm predictions on many examples, to give the reader a better appreciation of strengths and weaknesses of the proposed approach. Internally, we did this for over a dozen methods (not shown due to space limitations) and found it incredibly useful to understand when methods work, and when they fail. It is somewhat remarkable to note that there are over a dozen papers on time series anomaly detection that do not show a single plot of any time series, this, for what is one of the most visually intuitive data types.

There are several directions for future work, the most pressing of which are generalizing MERLIN++ to handle multi-dimensional data. For example, it is not clear if the best multi-dimensional anomaly detector is the *sum* of, the *max* of, or some other combination of, the individual dimensions.

In addition, recall that all the results shown in this work (except for Table 4) completely ignored the training data. In the future we plan to exploit such data if only to learn reasonable values for *MinL* and *MaxL*.

Finally, we note that we have made all datasets used in this paper, and an easy-to-use version of MERLIN++ , available in perpetuity at (Nakamura 2021). This will allow the community to easily confirm (or refute) our findings and further build on our work.

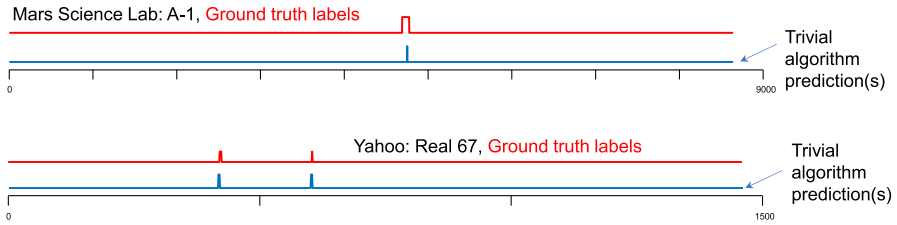
**Funding** Funding was provided by National Science Foundation (Grant no. 1631776)

## Appendix A: Some benchmark datasets are trivial

In the main text we noted that some fraction of the benchmark data yield to simple algorithms from the 1950s (Page 1957). Here we demonstrate that claim. This is important because it confounds any comparison of algorithms. For example, suppose

<sup>5</sup> An order of magnitude simpler in terms of number of parameters to set, of the number of lines of code written etc.

<sup>6</sup> To preempt confusion, we note that Page is an author.



**Fig. 25** Two (of many) examples of benchmark datasets that yield to the trivial hard-coded algorithm shown above. *Top* From NASA (Hundman et al. 2018). *Bottom* From Yahoo (Laptev and Amizadeh, 2015)

we find that Olympic powerlifter Long Qingquan can lift 1, 2, 3 and 300 kg, and that the current author can lift 1, 2 and 3 kg. It would be foolish to conclude that because they agree on  $\frac{3}{4}$  of the lifting tasks, that they are almost equally strong.

A further simplified version of the sixty-three-year-old algorithm in (Page 1957) is:

---

```
flag = zeros(size(T));           %% Code can be run in Matlab
for i = 4 : length(T)-4
    if std(T(i + 1:i + 4)) - td(T(i-3:i)) > 1,    flag(i) = 1;,    end;
end;
```

---

In Fig. 25 we show the results of running this code on two benchmark datasets that yield to such simple algorithms.

## Appendix B: choosing a value for $L$

A reasonable value for  $L$  is about one period, say one heartbeat, one day, one machine cycle etc. The following snippet of Matlab code will return one period.

---

```
[autocor, lags] = xcorr(T, 'coeff');
[~, m] = findpeaks(autocor(length(T) + 10:length(T) +
    1000), ...
lags(length(T) + 10:length(T) +
    1000), 'SortStr', 'descend', 'NPeaks', 1);
m(isempty(m)) = 1000;
m = floor(m);
```

---

While this is reasonable heuristic, it is not perfect. For example, an anomaly may show up only at a fraction of a period length, but then get swamped if you consider a full period. This is why the multiscale property of MERLIN++ is so useful, it frees the user from needing to be careful in their choice of a parameter.

## References

- Alireza A, Sara A, Shima I, Murillo AC, Gerry AC, Hickie L, Keogh EJ (2020) Fitbit for chickens?: time series data mining can increase the productivity of poultry farms. *KDD 2020*:3328–3336
- Ahmed M, and Abdun Naser M (2014) "Network traffic pattern analysis using improved information theoretic co-clustering based collective anomaly detection." In *International conference on security and privacy in communication networks*, pp. 204–219. Springer, Cham
- Ahmad S, Lavin A, Purdy S, Agha Z (2017) Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262:134–147
- 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
- Barz B, Yanira Guanache G, Erik R, and Joachim D (2017) "Maximally divergent intervals for extreme weather event detection." In *OCEANS 2017-Aberdeen*, pp. 1–9. IEEE
- Beyer K, Jonathan G, Raghu R, and Uri S (1999) "When is "nearest neighbor" meaningful?" In *International conference on database theory*, pp. 217–235. Springer, Berlin, Heidelberg
- Boniol P, Linardi M, Roncallo F et al (2021) Unsupervised and scalable subsequence anomaly detection in large data series. *VLDB J*. <https://doi.org/10.1007/s00778-021-00655-8>
- Bu Y, Lei C, Ada W-CF, and Dawei L (2009) "Efficient anomaly monitoring over moving object trajectory streams." In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 159–168. 2009.
- Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. In *ACM Comput Surv (CSUR)* 41(3):1–58
- Daigavane A, Wagstaff KL, Doran G, Cochrane C, Jackman C, and Rymer A (2020). Detection of Environment Transitions in Time Series Data for Responsive Science. In *MileTS '20: 6th KDD Workshop on Mining and Learning from Time Series*, August 24th, 2020, San Diego, California, USA. ACM, New York, NY, USA, 5 pages.
- Däubener Sina, Sebastian Schmitt, Hao Wang, Thomas Bäck, Peter krause, "Large Anomaly Detection in Univariate Time Series: An Empirical Comparison of Machine Learning Algorithms," *19th Industrial Conference on Data Mining ICDM*, 2019.
- Ding H, Trajcevski G, Scheuermann P et al (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc VLDB Endow* 1:1542–1552. <https://doi.org/10.14778/1454159.1454226>
- Doan Minh T, Sutharshan R, Mahsa S, Masud M, and Christopher L (2015) "Profiling pedestrian distribution and anomaly detection in a dynamic environment." In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 1827–1830. 2015.
- Filonov P, Andrey L, and Artem V (2016) "Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model." <http://arxiv.org/1612.06676>
- Heldt T, Oefinger MB, Hoshiyama M, Mark RG (2003) Circulatory response to passive and active changes in posture. *Comput Cardiol* 30:263–266
- Huet A, Navarro JM, Rossi D (2022) Local Evaluation of Time Series Anomaly Detection Algorithms. *arXiv preprint arXiv:2206.13167*.
- Hundman K, Valentino C, Christopher L, Ian C, and Tom S (2018) "Detecting spacecraft anomalies using lsmns and nonparametric dynamic thresholding." In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 387–395. 2018.
- Hwang W-S, Yun J-H, Kim J, Min BG (2022) Do you know existing accuracy metrics overrate time-series anomaly detections? In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. Association for Computing Machinery, New York, NY, USA, pp 403–412

- Imamura M, Takaaki N, Eamonn K (2020). Matrix Profile XXI: A Geometric Approach to Time Series Chains Improves Robustness. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining Pages 1114–1122
- Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001) Dimensionality reduction for fast similarity search in large time series databases. *Knowl Inf Syst* 3:263–286. <https://doi.org/10.1007/PL00011669>
- Keogh E, Jessica L, and Ada F (2005) "Hot sax: Efficiently finding the most unusual time series subsequence." In Fifth IEEE International Conference on Data Mining (ICDM'05), pp. 8–pp. Ieee
- Kim S, Choi K, Choi H-S et al (2022) Towards a rigorous evaluation of time-series anomaly detection. *Proc AAAI Conf Artif Intell* 36:7194–7201. <https://doi.org/10.1609/aaai.v36i7.20680>
- Laptev N and Saeed A, "S5 - A Labeled Anomaly Detection Dataset, version 1.0(16M)." 2015. Distributed by Yahoo Research. <https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70>.
- Lin J, Eamonn K, Ada F, and Helga Van H (2005) "Approximations to magic: Finding unusual medical time series." In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pp. 329–334. IEEE
- Linardi M, Zhu Y, Palpanas T, Keogh E (2020) Matrix profile goes MAD: variable-length motif and discord discovery in data series. *Data Min Knowl Discov* 34(4):1022–1071
- Marcus M, and Minc H (1992). *A survey of matrix theory and matrix inequalities* (Vol. 14). Courier Corporation.
- McRae M, Melbourne D- Bourke St Mall Flash Mob - 29th June 2013. (Jul. 20, 2013). Accessed: May 22, 2020. [Online Video]. Available: [www.youtube.com/watch?v=gLzDFjiRQE8](http://www.youtube.com/watch?v=gLzDFjiRQE8).
- Mueen A (2015). The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance, URL: <http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html>
- Murray D, Stankovic L, Stankovic V, and Liao J (2015) University of Strathclyde, PURE <http://dx.doi.org/https://doi.org/10.15129/31da3ecef902-4e95-a093-e0a9536983c4>
- Nakamura T (2021) MERLIN Webpage: <https://sites.google.com/view/merlin-find-anomalies>.
- Nakamura T, Imamura M, Mercer R, Keogh E (2020) "MERLIN: Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives". *ICDM*: 1190–1195
- Nichiforov C, Stancu I, Stamatescu I, Stamatescu G (2020) Information Extraction Approach for Energy Time Series Modelling, 24th International Conference on System Theory, Control and Computing, ICSTCC 2020, October 8–10, 2020, Sinaia, Romania.
- Orchard MT (1991) A fast nearest-neighbor search algorithm. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2297–2300, IEEE Computer Society Press.
- Page, E. S. "On problems in which a change in a parameter occurs at an unknown point." *Biometrika* 44, no. 1/2 (1957): 248–252.
- 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>
- Pedestrian Counting System (2013) City of Melbourne—Pedestrian counting system. In: *Pedestrian Counting System*. <http://www.pedestrian.melbourne.vic.gov.au/#date=28-10-2021&time=8>. Accessed May 22, 2020.
- Thanawin R, Bilson JLC, Abdullah M, Gustavo EAPAB, Brandon Westover M, Qiang Z, Jesin Z, Eamonn JK (2013) Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. *ACM Trans Knowl Discov Data* 7(3):1–31
- Thirey B and Hickman R (2015) Distribution of euclidean distances between randomly distributed gaussian points in n-space. [arXiv:1508.02238](https://arxiv.org/abs/1508.02238).
- Vasheghani-Farahani I, Alex C, Russell EK, Michael GK, and Brad K (2019) "Time Series Anomaly Detection from a Markov Chain Perspective." In 2019 18th *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1000–1007. IEEE
- Wang JT-L, Wang X, Lin K-I, et al (1999) Evaluating a class of distance-mapping algorithms for data mining and clustering. In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '99*. ACM Press, San Diego, California, United States, pp 307–311
- Wei, Li, Keogh, Eamonn, Xi, Xiaopeng. SAXually Explicit Images: Finding Unusual Shapes. *ICDM 2006*: 711–720
- Wikipedia (2022) "January 2017 Melbourne car attack.", last modified July 3, 2022, 1:21, [https://en.wikipedia.org/wiki/January\\_2017\\_Melbourne\\_car\\_attack](https://en.wikipedia.org/wiki/January_2017_Melbourne_car_attack).
- Wu, R and Keogh Eamonn (2021) Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. *CoRR abs/2009.13807* (2020)

- Yankov D, Keogh E, Rebbapragada U (2008) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. *Knowl Inf Syst* 17(2S):241–262
- Yeh C-CM, Yan Zhu LU, Nurjahan B, Yifei D, Hoang AD, Diego FS, Abdullah M, and Eamonn K (2016) "Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets." In *2016 IEEE 16th international conference on data mining (ICDM)*, pp. 1317–1322. Ieee.
- Zhang V (2019), "A tour of AI technologies in time series prediction." Society of Actuaries.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.