MERLIN: Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives

Saifeddine Barkia saifedinebarkia@gmail.com
Aymane Nohair aymane.nohair@polytechnique.edu

March 23, 2022

1 Introduction

Time series anomaly detection is an important topic in Data Science, with work dating back to the 1950s. In recent years, however, there has been an explosion of interest in this topic, due in large part to the success of Deep Learning in other fields and for other time-series tasks. In this project, we study the paper published by Takaaki Nakamura, Makoto Imamura and Ryan Mercer "MERLIN, Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives", where they elaborate an anomalies detection heuristic as competitive as already existing advanced Deep learning methods. We start with the notion of discords which are basically subsequences of a Time series that are maximally far away from their nearest neighbors. Unlike many other methods with many parameters, Discords requires the user to specify only one parameter: the length of the subsequence. In this project, we show that the utility of discords is reduced by sensitivity to this single user decision. Instead of computing discords of all lengths and then selecting the best anomalies (according to some measure), which seems computationally untenable, we choose to follow MERLIN, an algorithm that can efficiently and accurately find discords of all lengths in large time series archives.

Regarding the code, we have implemented it ourselves in Python; the different algorithms presented in the paper using the provided pseudo-codes and referring also to the author's MatLab implementation. For the different experiments that we implemented, we used datasets that were provided in the paper. As for the improvement, we implemented a generalization of the MERLIN algorithm to detect not only the top 1, but also the top-k discords.

2 Model & Definitions

2.1 Time Series Discords

We begin by the definition of some key notions that revolve around anomalies detection.

In this project, the study will focus on time series subsequences called **Discords**. A subsequence D of length L beginning at position i is said to be the discord of T if D has the largest distance to its nearest non-selfmatch. In other words, for any subsequence C of T, non-selfmatch M_D of D, and non-selfmatch M_C of C, $min(Dist(D, M_D)) \geq min(Dist(C, M_C))$.

A subsequence M of length L starting at position p is said to be a non self-match for a subsequence D starting at position q if and only if $|p - q| \geq L$.

Discords are identified by two parameters: the index of the first element and its length. Unlike other efforts that have been made in detecting anomalies with discords where only a single length was looked at, this project aims at exploring an unsupervised where a wide array of discord lengths is considered.

2.2 DRAG Algorithm

After this short introduction, we are set to explore our anomalies detection algorithm. First, we begin with the DRAG algorithm, where **DRAG** stands for Discord Range Aware Gathering. We input a single parameter r which corresponds to the minimum distance of the discord that we expect to spot. Ideally, this parameter should not be very small to avoid falling into the trap of big complexity, nor too big in which case we would not detect anything. Hence this value should ideally be set such that it is a little less than the discord distance, which is in most of cases a hard task to accomplish.

The DRAG algorithm is built in two steps:

- **Step I :** Let C be a set of candidate discords. The algorithm consists of sliding along the time series: we examine each subsequence, if the subsequence examined is farther than r from an element in the set, it could be the unconformity and is added to the set. However, if an element in set C is less than r away from the subsequence under investigation, we know that it cannot be an unconformity and it is removed from the set. At the end of Step I, the set C is guaranteed to contain the true unconformity, possibly with some additional false positives.
- **Step II :** The goal of this phase is to remove as many as possible false positives that we have computed in the first phase. It consists of sorting the discords found in the set above based on their distances. Only discords whose distance is greater than r are allowed to pass. The largest score corresponds to the top 1 discord in the time series.

2.3 MERLIN Algorithm

If we want to get around this problem of finding the right r parameter, the most natural method would be to widen the range of possible discords where we conduct our research. However, by exploring a large value of r, we show below that it is computationally quite expensive and demands a real effort of accuracy. To get the idea of how expensive it is, we apply the DRAG algorithm on a dataset that we will explore further in the next chapters (The Taxi demand in NYC): if our guess of r was smaller by 2 distant points, the drag algorithm would take around 50 times longer to run. In the opposite direction, even if the execution time seems to drastically decrease, choosing a higher value for r by only 5% would condemn the algorithm to fail. So choosing a good value for r is very critical to make the DRAG algorithm efficient.

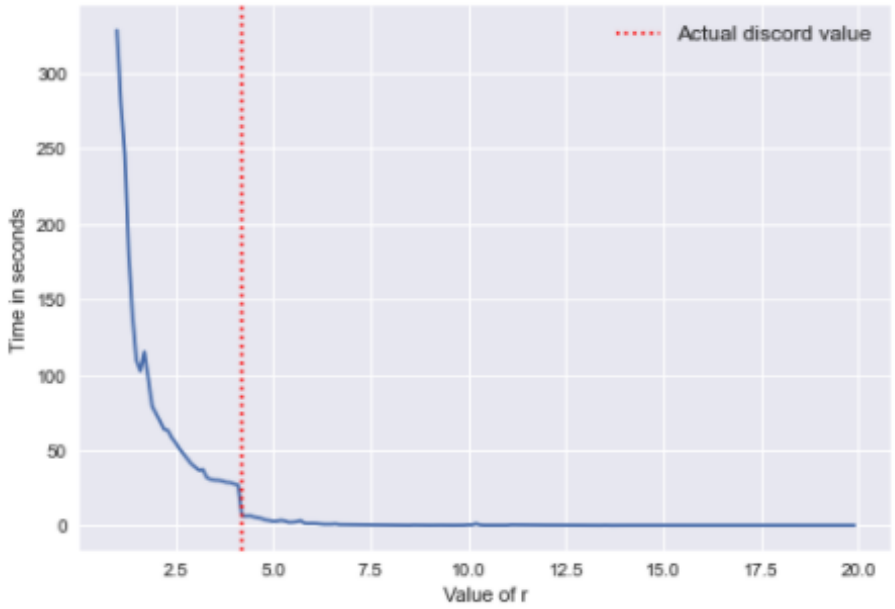


Figure 1: DRAG Execution Time for values of r ranging from 1.0 to 20.0

The goal of the MERLIN algorithm is to set the optimal value for r so that the DRAG algorithm would be efficient. The authors observed that the value of r for discord of length L is likely to be very similar to discord of length L-1. So an appropriate algorithm is to compute r starting from the minimum discord length to the maximum discord length and we would like to reduce the r little by little depending on the data. For that reason, at each stage, the authors computed the mean and the standard deviation for the last 5 discords and used $r = \mu - 2\sigma$, they repeated that until they a success. As for the first 5 discords, they set the value of $r = 2\sqrt{L}$ which is an upper bound for the discord distance and they keep halving it until success is achieved. However, this bound is very pessimistic, since it produces many failure modes, for that reason the authors used it to generate only the first discord and for the next 4, they subtract at each time 1% of the discord distance found for the minimum length until they report a success.

3 Data

In this paper, the authors tested the performance of the MERLIN algorithm on different datasets that were taken from the real-world problem where we have some unusual events. For the purpose of this project, we decided to re-implement the algorithm scheme that was provided and we try to regenerate the different results that were obtained for some datasets (see Figure 4).

First, we will examine a synthetic noisy signal and try to detect the present anomaly. After that, we will be working on the "NYC taxi" which is a dataset consisting in counting the Taxi passenger frequency from 2014-07-01 to 2015-01-31. The sampling frequency is every 30 minutes consisting of 10,320 data points. There are five anomalies that occur during the NYC marathon, Thanksgiving, Christmas, New Year's day, and a snowstorm. we will try to detect them and see if we can detect even more anomalies that were not mentioned above.

After that, we will be exploring some of the signals of Yahoo dataset. Yahoo has a set of 371 anomaly benchmark time series. It will be impossible to explore them all in the report, for that reason we will report only one of the signals presented in this dataset.

4 Results

In this section, we will report the different discords that we have obtained using the different implemented algorithms.

First, we will start with a synthetic signal to test the efficiency of the implemented DRAG algorithm. After choosing correctly the values of m and r, we were able to detect the anomaly in the signal. However, we noted that those values were a little tricky to choose since choosing any other combination can lead to different results.

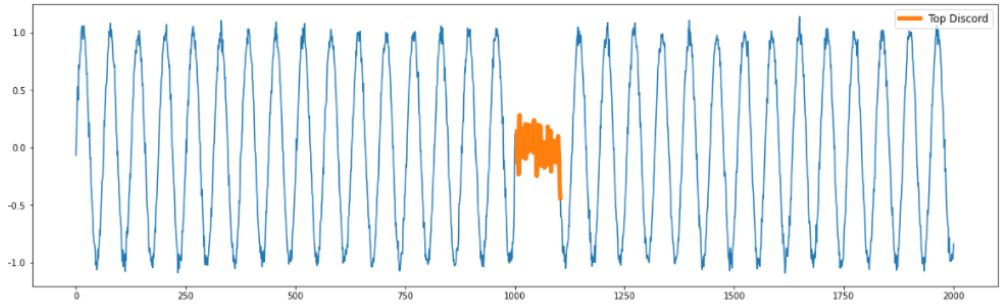


Figure 2: Top one discord detection using DRAG algorithm with well chosen m and r

To be sure that our DRAG algorithm is performing well, we tested it also on a real dataset (NYC taxi). The reported discord corresponds to the blizzard that happened in New York City between the 25^{th} and the 27^{th} of January 2015.

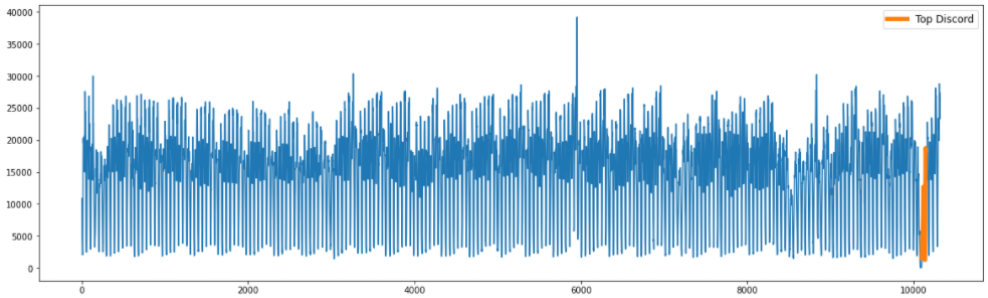


Figure 3: Top one discord detection using DRAG algorithm with $m = 48$ (2 days) and $r = 4.2$

Now that we are confident that our DRAG algorithm is working well, we tested the MERLIN algorithm to see its capacity for detecting anomalies. We choose an interval of subsequences between the frames of 5 hours and 48 hours (2 days). In figure 5, we can see that we have detected many outliers. Checking with the main events in New York this period we report them in the following order. The first discord corresponds to the 4^{th} of July (the independence day), then we

have the 'St Luke' day on October 18^{th}, then we have detected many discords at different lengths which correspond to the NYC marathon on 02/11/2014. We also detected Christmas Eve with only one discord length on 24/12/2014 and finally, we detected many discords with high subsequence lengths which correspond to the blizzard that happened between the 25^{th} and 29^{th} in the USA.

Surprisingly, we were unable to detect both the New Year Event and the Thanksgiving. The only explanation for that is that the blizzard had more impact since we are only considering the top-1 discord. In order to detect such events, we are left with two options: either we exclude the period of the blizzard or we implement a version of MERLIN that detects the top-K discords. We have decided to explore the second path. In figure 7 we find the result of MERLIN for the top 4 discords. We can clearly see that we now detect the Thanksgiving and the New Year event but also with some false negatives.

We also computed the discords using the Matrix Profil methods to see if the detected discords match. The results shown in figure 6 show that those discords are also detected using the MERLIN algorithm.

We report here also the results from one of the signals presented in the yahoo dataset. As we can see in figure 9, we were able to use the MERLIN algorithm to detect 5 anomalies compared to 6 ground truths (we spot one false negative). After running MERLIN to detect top-k discords, we were able to detect the other anomaly with some false positives.

5 Conclusion

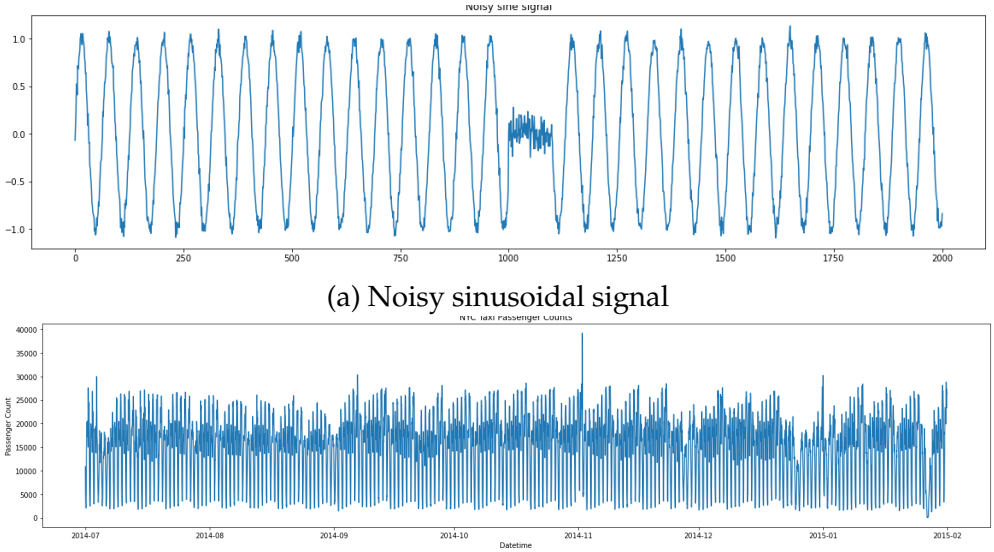
This project allowed us to dive into the heuristics proposed in the paper **MERLIN: Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives**. Unlike Merlin, we do not compute the pairwise distance between all subsequences in DRAG. Instead, we restrict down our search area by excluding those that can't be a discord based on a user-defined $\min(distance)$ metric. In other words, we start scanning the time series T and discard both subsequences whose distance is less than min dist as we compute the distance between them (i.e. Those two subsequences are considered to be "close" and, thus, none of them can be a discord).

The added value of Merlin's algorithm lies in the fact that $\min(distance)$ criterion is no longer required. Although one can have their own initialization, they do not need to supply one in order to receive the discord. When the algorithm produces no discord, MERLIN fixes the problem by rerunning it with a smaller $\min(distance)$. Furthermore, it can swiftly find the discords of length $L+1$ after finding the discords of length L. All in all, this project taught us about the quality improvement of time series through an unprecedented data processing technology.

Contribution

- *Saifeddine Barkia* : Algorithms Implementation + NYC Taxi Dataset Experiment + Report
- *Aymane Nohair* : Algorithms Implementation + Yahoo Dataset Experiment + Report

Appendix



(a) Noisy sinusoidal signal

(b) NYC Taxi Dataset

Figure 4: Datasets illustration

Figure 5: Detected discords on the NYC taxi dataset using MERLIN algorithm with minimum subsequence length = 5 and maximum subsequence length = 48.

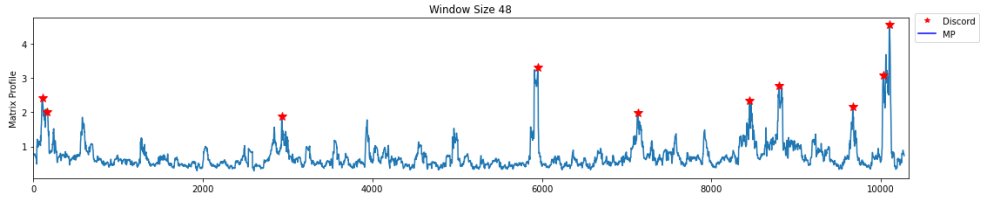


Figure 6: Discords detected using the Matrix Profile method.

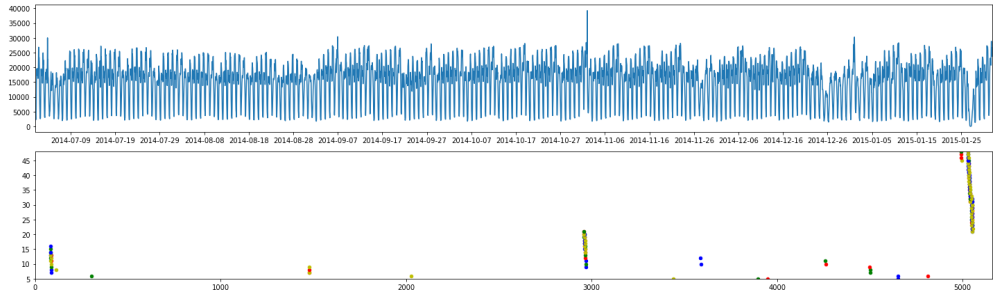


Figure 7: MERLIN output when we considered top 4 discords on NYC taxi dataset

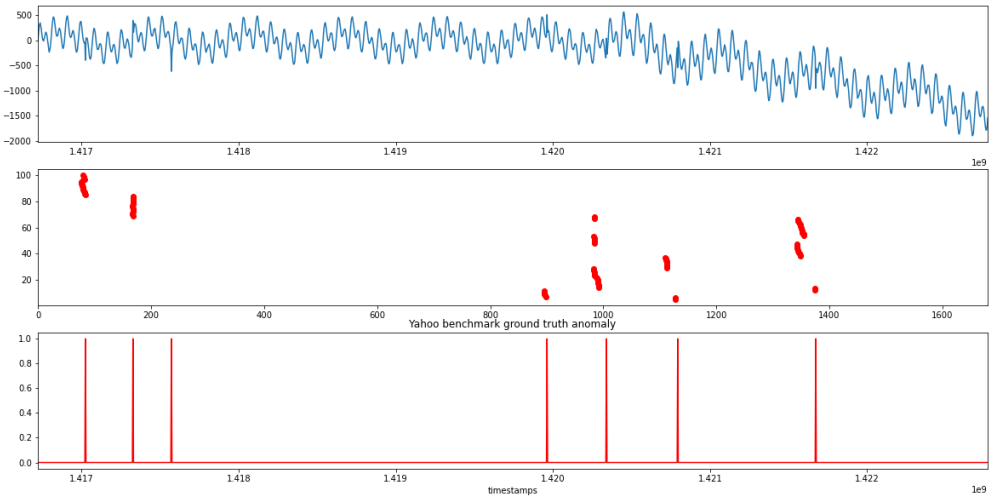


Figure 8: Detected discords on the Yahoo dataset using MERLIN algorithm with minimum subsequence length = 5 and maximum subsequence length = 100

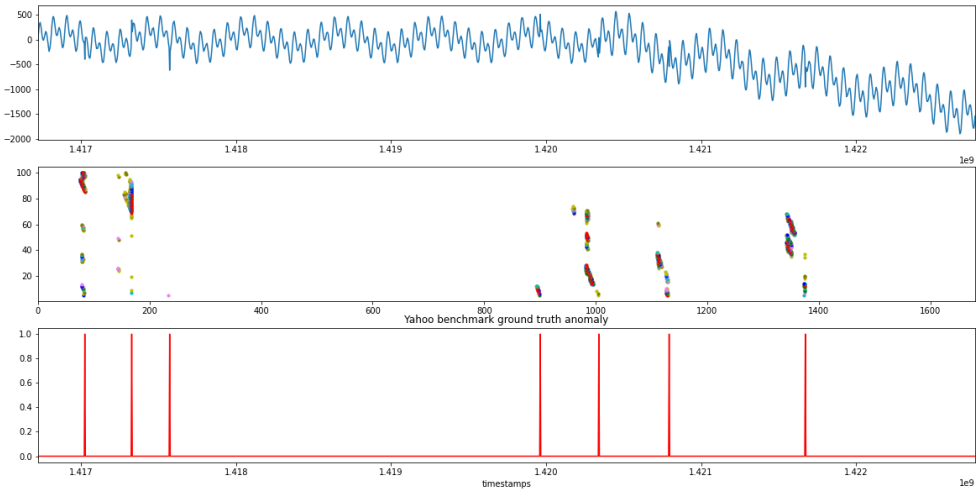


Figure 9: Top 7 Detected discords on the yahoo benchmark dataset using MERLIN algorithm with minimum subsequence length = 5 and maximum subsequence length = 100 with the real ground truth anomalies.

References

[1] MERLIN Webpage *https://sites.google.com/view/merlin-find-anomalies*.

[2] A. Bagnall et al. (2017) *The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances,"*, Data Min.

[3] N. Laptev and S. Amizadeh (2015) *S5 - A Labeled Anomaly Detection Dataset*, version 1.0(16M).

[4] Vasheghani-Farahani, et. al (2019) *Time Series Anomaly Detection from a Markov Chain Perspective*, ICMLA.

[5] S. Däubener, et. al (2019) *Large Anomaly Detection in Univariate Time Series: An Empirical Comparison of Machine Learning Algorithms*, ICDM.

[6] J. Lin, E. Keogh, A. Fu, H. Van Herle (2005) *Approximations to Magic: Finding Unusual Medical Time Series*, CBMS.