Anomaly Detection in Streaming Time Series Based on Bounding Boxes

Heider Sanchez* and Benjamin Bustos

Department of Computer Science, University of Chile, Santiago, Chile
{hesanche,bebustos}@dcc.uchile.cl

Abstract. Anomaly detection in time series has been studied extensively by the scientific community utilizing a wide range of applications. One specific technique that obtains very good results is "HOT SAX", because it only requires a parameter the length of the subsequence, and it does not need a training model for detecting anomalies. However, its disadvantage is that it requires the use of a normalized Euclidean distance, which in turn requires setting a parameter ε to avoid detecting meaningless patterns (noise in the signal). Setting an appropriate ε requires an analysis of the domain of the values from the time series, which implies normalizing all subsequences before performing the detection. We propose an approach for anomaly detection based on bounding boxes, which does not require normalizing the subsequences, thus it does not need to set ε. Thereby, the proposed technique can be used directly for online detection, without any a priori knowledge and using the non-normalized Euclidean distance. Moreover, we show that our algorithm computes less CPU runtime in finding the anomaly than HOT SAX in normalized scenarios.

Keywords: Time Series, anomaly detection, indexing, streaming.

1 Introduction

Anomaly detection in time series has been studied extensively by the scientific community, who has contributed a wide variety of approaches for different types of applications [8]. In data mining, research is generally focused on searching for unusual patterns or outliers in a collection of time series using classification or clustering [21,10,9]. Most of the state-of-the-art techniques for anomaly detection in time series use a time series sample of "normal" behavior as a training model. However, data mining on subsequences from a streaming time series is a more complicated task because of particular challenges that need to be addressed. The main challenge of the subsequences is their level of overlapping: contiguous subsequences are similar to each other. This may produce a meaningless clustering result [23]. Moreover, the definition of "anomaly" is ambiguous and may be mistaken for irregularities that occur along a streaming time series, for example variations in amplitude scale and the presence of local noise.

* Work supported by a research grant from CONICYT-Chile.

A.J. Machado Traina et al. (Eds.): SISAP 2014, LNCS 8821, pp. 201–213, 2014.
DOI: 10.1007/978-3-319-11988-5_19

Keogh et al. [18] introduced a new anomaly concept, the "Time Series Discord", for finding the most unusual time series subsequence which does not need a training model. While similarity searching by content finds the object most similar to a query, the discord discovery process finds the time series subsequence that is least similar to all other subsequences. Moreover, Keogh proposed a generic heuristic for efficient discord discovery and a solution algorithm, the so-called HOT SAX, that is based on Symbolic Aggregate approXimation (SAX). Later, a series of related works were proposed to improve the performance of the basic heuristic [5,6,20]. All of these solutions use the normalized Euclidean distance (L2-norm), that is, each subsequence is normalized by a standard normalization procedure (Z-distribution) to obtain a symbolic sequence.

The problem with using normalized subsequences is the presence of local noise, which can result in a missed detection. This issue is easily solved by applying a parameter ε [22]. Given a subsequence $C = \{c_1, \ldots, c_m\}$, let σ and μ be the standard deviation and the mean of C, respectively. Then, if $\sigma < \varepsilon$ one sets $\forall i,\ c_i = \mu$. The problem with this approach is that one needs setting the parameter ε, which is context-dependent. On the other hand, the Euclidean distance over non-normalized subsequences (L2-raw) does not need to set ε and is more robust to local noise. Additionally, in real-time streaming, the future values of the time series are unknown. Therefore, obtaining an optimal ε is a complicated task. Moreover, there are anomalies related to amplitude changes and local oscillations (e.g., El Niño-Southern Oscillation Events [26]). These types of anomalies are at risk of not being detected in a normalized scenario. In these cases, using L2-raw is an effective option. Moreover, scalability is an important factor for many real time systems that generate large time series (e.g., seismic signals [12], electrocardiograms [19,11] and network traffic [2]).

This paper makes two contributions in streaming time series anomaly detection. First, we propose an algorithm for efficient time series discord discovery which supports both L2-norm and L2-raw distances. Second, we introduce a new automatic learning online algorithm to detect local discords in streaming time series. Our model is based on the previous works of Keogh et al. [16] and Vlachos et al. [27]. Specifically, they proposed the RTree-Index for time series using Bounding Boxes and the Dynamic Time Warping (DTW) distance. We propose modifying this structure to work directly with L2-raw and the anomaly detection algorithm. We experimentally show that our technique is faster than HOT SAX in normalized subsequences.

2 Background and Related Work

2.1 Time Series

Definition 1. *Streaming Time Series. A sequence of observations* $T = \{t(i)/i = 1 \cdots \infty\}$ *taken at various time moments, evenly spaced and chronologically sorted.*

Definition 2. *Sliding Window. Given a time series* T *of length* n*, we use a overlapping sliding window of length* $m \ll n$ *to extract all possible subsequences*

C_p, $p \in \{1, \ldots, (n-m+1)\}$, from T. This window generates overlapping subsequences of contiguous position.

Definition 3. *Normalized Subsequence. Given a subsequence $C = \{c_1, \ldots, c_m\}$, its normalized version is defined as $C' = \frac{C-\mu}{\sigma}$, where μ and σ are the mean and standard deviation of C.*

The main indexing techniques of subsequences use a reduced representation of the time series to avoid the High-Dimensionality problem. They provide a lower bounding function of the true distance between two time series. Examples are the RTree for Dynamic Time Warping (DTW) [16], iSAX for L2-norm [25] and the TS-Tree that uses the best of both techniques [4].

2.2 Discord Discovery and State-of-the-Art Solutions

Anomaly definition in a time series is ambiguous and is strongly related to the application context and data properties. In streaming time series, one usually associates an anomaly with a subsequence that produces a qualitatively significant change in the data. Furthermore, most anomaly detection approaches work on the basis of a normal behavior model of a time series. However, in many real contexts, obtaining this a priori knowledge is a difficult task. Keogh et al. [19] introduced a new definition to avoid creating workable definitions for "the most unusual subsequence", which does not require a training model.

Definition 4. *Non-self match. Given a time series T, containing a subsequence C_p of length m and a matching subsequence C_q, we say that C_q is a non-self match to C_p if $|p-q| \geq m$, where p and q are their respective starting positions in T.*

Definition 5. *Time Series Discord. Given a time series T, the subsequence C of length m is said to be the discord of T if C has the largest distance to its nearest non-self match.*

This problem can be easily solved by a brute force search using a nested loop. The outer loop takes each subsequence as a possible candidate, and the inner loop is used to search the candidate's nearest non-self match. The candidate that has the greatest such value is the discord. The computational complexity is $O(N^2)$, where N is the number of subsequences. To improve this complexity, Keogh et al. [19] proposed a generic algorithm for efficient detection. This algorithm requires two heuristics that generate two ordered lists of subsequences; one for the outer loop and the other one for the inner loop. The heuristic *Outer* is useful for quickly finding the best candidate, and the heuristic *Inner* is useful for quickly finding the best nearest non-self match. We break out of the inner loop if the distance is less than the best-so-far discord distance.

The main related methods for discord discovery are based on SAX representation, which is a discretization technique for time series introduced by Lin et al. [24] and it is used in many application domains for different purposes.

Definition 6. *Breakpoints. "Breakpoints are a sorted list of numbers* $\beta = \{\beta_1, \ldots, \beta_{\alpha-1}\}$ *such that the area under a* $N(0,1)$ *Gaussian curve from* β_i *to* $\beta_{i+1} = 1/\alpha$ *(* β_0 *and* β_α *area defined as* $-\infty$ *and* $+\infty$*, respectively)." [24]*

Definition 7. *SAX Representation. Given a normalized subsequence* $C = \{c_1, \ldots, c_m\}$*, first, we obtain the reduced dimension with Piecewise Aggregate Approximation (PAA [15]): the time series is divided into D equal sized segments, the mean value of each segment is calculated and a vector of these values becomes the reduced representation* $P = \{p_1, \ldots, p_D\}$*. Afterwards, P is transformed into a symbolic sequence* $W = \{w_1, \ldots, w_D\}$*, where* p_i *is mapped to symbol* w_i *of an alphabet of size* α*. It uses the predetermined breakpoints to define the symbols.*

HOT SAX: It is the first algorithm for efficient discord discovery introduced by Keogh et al. [19]. HOT SAX uses an *augmented trie* to embed all the SAX words, where leaf nodes contain a linked list index of all word occurrences that map there. The second structure is an array of SAX words of all extracted subsequences, counting the frequency of each word occurrence in the array. After building these data structures, HOT SAX uses the following heuristics: (a) *Outer loop heuristic:* It first visits the subsequences associated with the SAX words that have the smallest word count, and then it visits the rest of the subsequences in random order; (b) *Inner loop heuristic:* For each candidate in outer loop, it first searches its nearest non-self match in the leaf node of tree that has the same SAX word, and then it visits the rest of the subsequences in random order.

HOTiSAX: It extends HOT SAX for working with the iSAX index [6]. iSAX is an optimized structure for SAX binary representation [25], which provides different levels of resolution for the same SAX word changing the symbolic alphabet by binary numbers. The bits are used for building the iSAX index, which allows an efficient hierarchical access to data. The array of SAX words is refined to work effectively with the iSAX representation. Afterwards, the same heuristics of HOT SAX are used for discord discovery. Moreover, it incorporates auxiliary functions to exclude trivial matches.

3 Bounding Boxes for Discord Discovery

A minimum bounding box is a term used in geometry for enclosing a set of D-dimensional points [13]. This concept was used by Vlachos et al. [27] for bounding two-multidimensional time series by a sequence of Minimum Bounding Rectangles (MBRs). All the MBRs are indexing in a RTree in order to allow the efficient searching. Similarly, Keogh et al. [16] applied bounding boxes over the PAA of time series. The goal of this work was to generate reduced dimension vectors for the time series being indexed with a RTree and the DTW distance. Furthermore, in both works the authors provided a lower bounding function of the true distance for exact searching. In another related project, Chan et al. [7] proposed the use of a sequence of minimal bounding boxes to contain all of the training time series for detecting anomalies in trajectories.

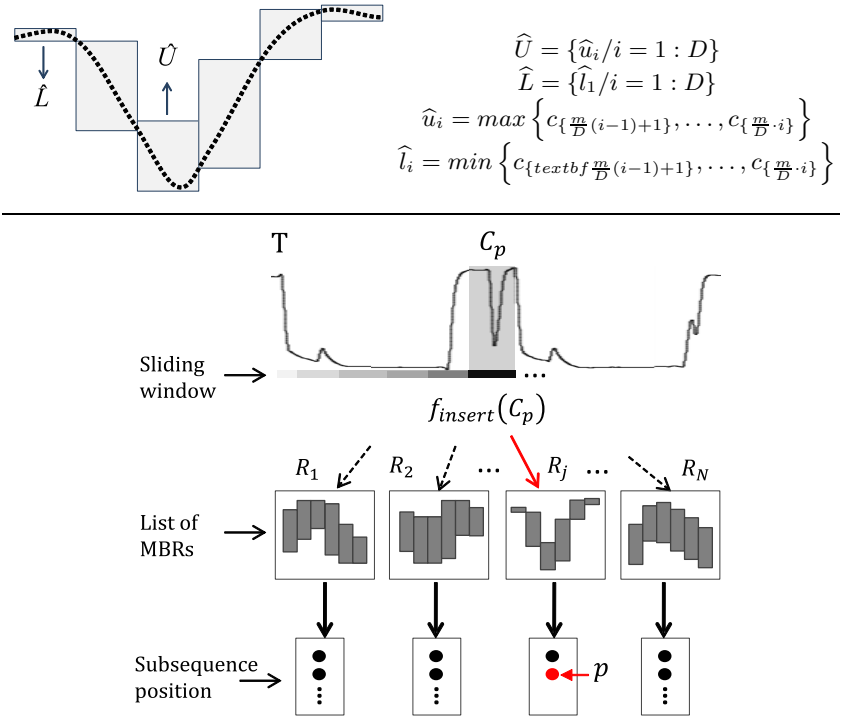

Fig. 1. Bounding of the subsequence C_p (top) and our indexing model for discord discovery (bottom)

In this paper, we use both the modeling of MBRs and the PAA representation for discord discovery in time series. To obtain efficient results on a streaming context, we use the Euclidean distance, due to its linear complexity.

3.1 List of MBRs

Figure 1 shows our indexing model of time series subsequences. We build a List of MBRs $= \{R_1, \ldots, R_N\}$, where each R_i envelops a set of similar time series subsequences. Below, we describe the insertion process phases:

1. Given a time series T, we extract a subsequence $C_p = \{c_1, \ldots, c_m\}$ using a sliding window. Next, we generate the minimum bounding boxes $(\widehat{U}, \widehat{L})$ of C_p (Figure 1.top). This function requires a reduced dimension D to split the subsequence C_p in D equal length segments, then, $\widehat{u}_i$ and $\widehat{l}_i$ is the maximum and minimum value of the *ith* segment respectively.
2. We search a MBRs R_j that produces the least expansion to C_p. That is, $volume(R_j \cup (\widehat{U}, \widehat{L})) < volume(R_{i,\forall i \neq j} \cup (\widehat{U}, \widehat{L}))$. We then stretch R_j to envelop the minimum bounding boxes of C_p. Finally, the position p is inserted into an integer number array, which is associated with R_j.

3. If R_j is full, we apply a splitting algorithm. We use a size threshold th_{max} to control the maximum number of elements in a MBRs.

We evaluate three classic splitting algorithms: Guttman's quadratic and linear algorithms [14], and an optimized linear algorithm [3]. For the quadratic algorithm, we considered two criteria of distribution; balanced (qua_0.50) and non-balanced (qua_0.25). Figure 2 shows the number of created MBRs and the CPU runtime in indexing and searching of the most unusual subsequence. We observe that the optimized linear algorithm generates the fewest MBRs, and therefore gets the least memory space. However, the balanced quadratic algorithm achieves the best runtime because it performs a better grouping of subsequences allowing fast discrimination in searching time. Therefore, we use the quadratic algorithm in our indexing model.

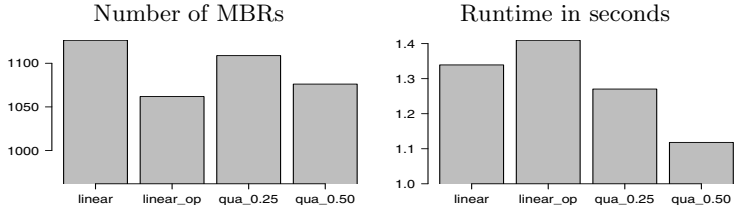

Fig. 2. Testing four splitting algorithms over a set of time series of length 16k. We show the total number of created MBRs (left) and the CPU runtime (right) for discord$_{128}$.

The time required to insert a subsequence in the list has an order of $O(N)$, where N is the total number of created MBRs. The space required for maintaining the index in memory also depends on N. For each $R_j = (H, L)$, we used two arrays of size D and another array of integer numbers for saving the subsequence positions; e.g., for a time series of length 16,000 we require 16.7KB for the index (N=1070) and 62KB for the arrays of positions.

We also build a Tree of MBRs (RTree variant) for maintaining the set of MBRs. It provides us hierarchical access to the data in order to accelerate the searching of similar objects regarding a query object. The RTree is generally used for managing large collections of data in secondary memory.

3.2 Discord Discovery Heuristics

After building the index, we reorder all subsequences for searching the best candidate using the following heuristics:

Outer Loop Heuristic: First, the algorithm visits all subsequences bounded in R_j, such that R_j contains the minimum number of subsequences. Then, the algorithm visits the rest of the subsequences in random order. This heuristic ensures that the subsequences that are most isolated will be visited at the beginning of the search as potential candidates. We then use an inner loop to search the best non-self match of each selected candidate C_q. To break the inner loop as early

as possible, we need to find a subsequence that has a distance to C_q lower than the best-so-far discord distance.

Inner Loop Heuristic: First, the algorithm visits all subsequences bounded in R_j, such that $MINDIST(\bar{C}_q, R_j) < MINDIST(\bar{C}_q, R_{i,\forall i \neq j})$. Then, the algorithm visits the rest of the subsequences in random order. This heuristic allows us to first visit all the subsequences C_p most similar to C_q, increasing the probability of early termination of the loop. $MINDIST$ function is calculated by Equation 1 and illustrated in Figure 3 using $\bar{C}_q$ as PAA representation of C_q.

$$MINDIST(\bar{C}_q, R) = \sqrt{\sum_{i=1}^{D} \frac{m}{D} \begin{cases} (\bar{c}_i - h_i)^2 & \bar{c}_i > h_i \\ (\bar{c}_i - l_i)^2 & \bar{c}_i < l_i \\ 0 & \text{else,} \end{cases}} \quad \text{where:} \quad \bar{c}_i = \frac{D}{m} \sum_{j=\frac{m}{D}(i-1)+1}^{\frac{m}{D}i} c_j. \tag{1}$$

3.3 Online Anomaly Detection

We can use discord discovery for detecting anomalous subsequences in streaming time series. A simple idea is to use normal behavior data to build a static training model. Discord discovery is then used for online detection of new inputs. This approach is very efficient, because the model retains its size, although the streaming is increased. However, it does not evolve to new states of the system. Any input that generates a new behavior is always considered anomalous.

We propose a scalable method for detecting local anomalies, where online learning is required for adapting the index model with the behavior of the input data. Our method is based on the inner loop heuristic (Algorithm 1). It requires a detection starting point in the stream, which is used to determine a "base history". Then, we apply discord discovery up to this point to obtain a threshold distance (line 1). This threshold is used to reduce the number of calls to $Dist$ in the inner loop. Our method is fed back with each data input (line 6) in order to avoid detecting recurrent anomalies. That is, if a subsequence is detected as anomalous, the next similar subsequences will not be detected as anomalous. Finally, we alert when an value input generates an anomalous subsequence.

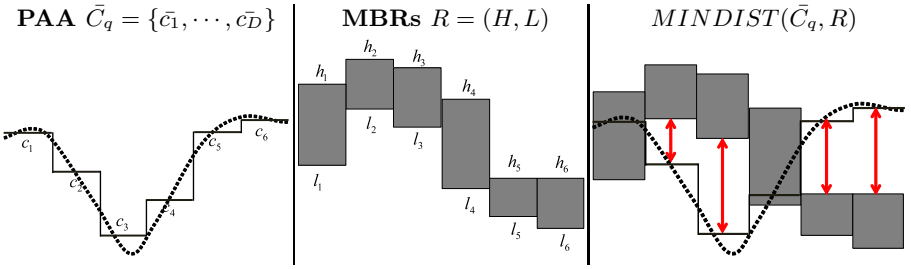

Fig. 3. An illustration of the $MINDIST$ function. The lengths of the red arrow lines, squared, scaled by m/D, summed and square rooted, are returned as the minimum distance between C_q and any sequence bounded in R [16].

Algorithm 1 . Online Anomaly Detection

Require: (Streaming T, Window length m, Starting Point sp)

```
threshold_dist = TheMostDiscord(T[1 : sp])
while Input == True do
    T[i] = AcquireValue()
    C_q = T[(i − m + 1) : i]                          ▷ extract the subsequence
    insert(C_q)                                        ▷ feedback
    l_inner = All C_p from T ordered by heuristic Inner
    nearest_neighbor_dist = ∞
    for C_p ∈ l_inner do
        if |p − q| ≥ m then                            ▷ non-self match?
            if Dist(C_p, C_q) < nearest_neighbor_dist then
                nearest_neighbor_dist = Dist(C_p, C_q)
            end if
            if Dist(C_p, C_q) < threshold_dist then
                Break                                  ▷ Break out of loop
            end if
        end if
    end for
    if nearest_neighbor_dist > threshold_dist then
        Alarm("Anomaly Detected")
    end if
end while
```

The starting point can be automatically computed in real time. This is possible because the distance to the best non-self match usually reduces its value when increasing the stream length (see Figure 6.bottom). The idea is to find the stabilization point and set it as the starting point. A simple way to achieve this is by counting the distances outside the range of the deviation standard $[\mu - \sigma : \mu + \sigma]$ and applying a stop proportion.

In a long stream, it is important to update the threshold with the actual context to improve the detection of local anomalies. For this, we recommend periodically removing the past information and again, computing the new threshold with the rebuilt index. This is a scalability improvement to apply at runtime.

4 Experimental Evaluation

In this section, we evaluate the performance of our approach for discord discovery in different datasets. An Intel Core i7 3.4GHz with 8GB RAM is used for conducting all our experiments. All algorithms are implemented in C++. We define $th_{max} = 25$ as the maximum size of elements in a MBRs, which was experimentally selected from the set $\{5, 10, 15, \cdots, 120\}$. Although better efficiency is obtained when we vary the value of th_{max} according the time series length. The value of this parameter does not alter the effectiveness.

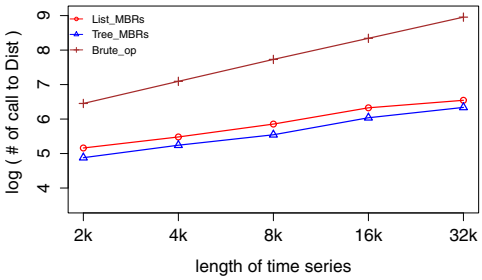

Fig. 4. The number of calls to the distance function (*Dist*) by brute force and our approach for discord$_{128}$ using L2-raw

4.1 Offline Discord Discovery

To evaluate the efficiency of our approach on static time series, we use the datasets ECG, EEG, ERP, Koski, Random Walk and Packet from "The UCR Time Series Data Mining Archive" [17]. We also use the "Time Series for Weather Data" from the National Oceanic and Atmospheric Administration in the USA [1]. For each dataset, we randomly extract time series of lengths 1k, 2k, 4k, 8k, 16k and 32k. The metrics used for comparison are the number of computed distances and the CPU runtime. The goal is to find the most unusual subsequence using the fewest distance computations and the minimum time of execution.

First, we compare our approach (List of MBRs and Tree of MBRs) with the brute force search over non-normalized subsequences using the L2-raw distance. We optimize the brute force search applying symmetry and triangle inequality properties. Figure 4 shows the performance of our approach in terms of the number of computed distances. We get the median of the results obtained for all the datasets for each time series length. From these results we note that our method clearly outperforms the brute force search, ≈ 2.5 orders of magnitude faster on all collections with time series of length 32k. This difference is correlative with the CPU runtime, where our approach was 150 times faster than the brute force.

Second, we compare our approach with the SAX techniques over normalized subsequences using the L2-norm distance. For SAX representation, the authors recommended using the following parameter setting: size of alphabet $a = 3$ and word length depending on data [19]. However, iSAX binary representation requires $a \in 2\mathbb{Z}$ (multiple of 2), we therefore set $a = 4$ [6]. Also, we set the reduced dimension based on the window length $D = \lfloor log_2(m) \rfloor$.

In Figure 5, we note our algorithms have less performance than the SAX techniques in terms of computed distances. However, our algorithms outperform HOT SAX in CPU runtime. This is explained by the low number of created

Table 1. The number of calls to MINDIST for discord$_{128}$

List of MBRs	Tree of MBRs	HOT SAX	HOTiSAX
226,259	482,256	1,950,445	314,407

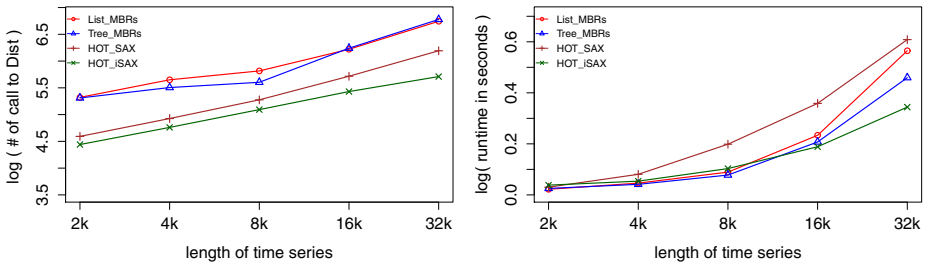

Fig. 5. The number of calls to the distance (left) and the CPU runtime (right) by SAX techniques and our algorithms for discord$_{128}$ using L2-norm and $\varepsilon = 0.05$

MBRs versus the number of created buckets by HOT SAX, resulting in the lower memory usage. Our approach uses a balanced splitting algorithm to maximize the number of elements in the MBRs. Thus, we obtain more closed groups and a better effect of the MINDIST function. Table 1 shows the number of computed MINDIST for each technique in a set of real time series.

Finally, the Tree of MBRs does not get a wide lead over the List of MBRs. The first reason is that we do not need to save all subsequences in secondary memory as it was sufficient with the referential positions in main memory. The second due to the additional cost of splitting the internal nodes. Moreover, the Tree of MBRs produces more calls to MINDIST than the List of MBRs in the search task. In practice, the discord discovery process is just applied in limited ranges of time in order not to lose local significance of the anomaly.

4.2 Online Anomaly Detection

In real-time streaming, we do not know the future values of the time series. Therefore, it is not clear how one could obtain an appropriate value for ε in normalized subsequences. For this reason, in this experiment we use L2-raw to evaluate our online anomaly detection algorithm. We cannot use SAX techniques because it is not compatible with L2-raw.

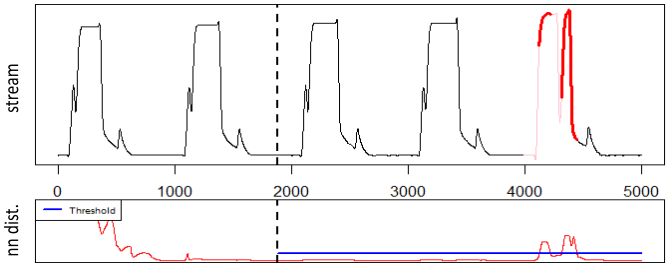

Fig. 6. Online anomaly detection on a space shuttle time series ($m = 128$)

In Figure 6, we evaluate our algorithm in a real case. The top graph shows the streaming time series and the detected anomalous subsection (red region). The bottom graphs represent the nearest non-self neighbor distance of each input subsequence where the dotted vertical line is the detection starting point that was automatically set. Also, we show the threshold distance (blue line), which is used to avoid unnecessary matches. We observe that our algorithm is successful at detecting the points that cause an anomalous subsequence at the right moment.

Figure 7 shows the number of computed distances over a set of real time series. We note a wide advantage of our indexing model vs. the brute force on all time series. In practice, our approach seems to compute far fewer distances than the quadratic brute force search algorithm. Moreover, we show the benefits of using our anomaly detection algorithm with regard to the similarity search.

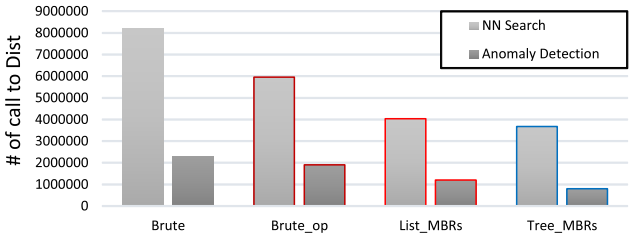

Fig. 7. The number of calls to the distance by both the nearest non-self neighbor search and the anomaly detection algorithm

5 Conclusions and Future Work

We proposed new algorithms for efficient discord discovery. We used bounding boxes for designing two indexing models List of MBRs and Tree of MBRs, which support L2-norm and L2-raw. Our approach outperforms HOT SAX in terms of the CPU runtime. In addition, we introduced a new algorithm for online anomaly detection, which does not require a training model and automatically fixes a detection starting point. We experimentally showed that this online detection algorithm is faster than the brute search force approach.

Using the discord discovery algorithm over a set of real time series [19], we obtain 79% correct detection using L2-raw. While it misses some anomalies, its advantage is that it avoids the exploration step of the data to set ε in smoothing noisy subsequences. We emphasize that L2-raw may be used as a baseline for automatic detection techniques for discord discovery, especially for quasi-periodic streaming time series.

Since we do not apply any discretization technique, we can easily extend our approach to multivariate time series, which is our next goal. Finally, we plan to optimize our online algorithm, focusing on scalability for larger amounts of data.

Acknowledgments. We would like to thank Prof. Eamonn Keogh for kindly providing us with many of the datasets used at the experimental evaluation.

References

1. Web Page for Time Series for Weather Data of National Oceanic and Atmospheric Administration in USA, `http://www.esrl.noaa.gov/psd/boulder/`
2. Ahmed, T., Coates, M., Lakhina, A.: Multivariate online anomaly detection using kernel recursive least squares. In: IEEE INFOCOM, pp. 625–633 (2007)
3. Ang, C.-H., Tan, T.: New linear node splitting algorithm for r-trees. In: Scholl, M.O., Voisard, A. (eds.) SSD 1997. LNCS, vol. 1262, pp. 337–349. Springer, Heidelberg (1997)
4. Assent, I., Krieger, R., Afschari, F., Seidl, T.: The TS-tree: Efficient time series search and retrieval. In: Proc. 11th Intl. Conf. on Extending Database Technology: Advances in Database Technology, pp. 252–263. ACM (2008)
5. Bu, Y., Wing Leung, O.T., Chee Fu, A.W., Keogh, E.J., Pei, J., Meshkin, S.: WAT: Finding top-k discords in time series database. In: SIAM Intl. Conf. on Data Mining, pp. 449–454 (2007)
6. Buu, H.T.Q., Anh, D.T.: Time series discord discovery based on iSAX symbolic representation. In: 2011 Third Intl. Conf. on Knowledge and Systems Engineering (KSE), pp. 11–18 (2011)
7. Chan, P.K., Mahoney, M.V.: Modeling multiple time series for anomaly detection. In: IEEE Intl. Conf. on Data Mining, pp. 90–97 (2005)
8. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. 41, 1–58 (2009)
9. Chaovalit, P., Gangopadhyay, A., Karabatis, G., Chen, Z.: Discrete wavelet transform-based time series analysis and mining. ACM Comput. Surv. 43, 1–37 (2011)
10. Chis, M., Banerjee, S., Hassanien, A.: Clustering time series data: An evolutionary approach. In: Abraham, A., Hassanien, A.-E., de Carvalho, A.P.D.L.F., Snášel, V. (eds.) Foundations of Computational, Intelligence Volume 6. SCI, vol. 206, pp. 193–207. Springer, Heidelberg (2009)
11. Chuah, M.C., Fu, F.: ECG anomaly detection via time series analysis. In: Thulasiraman, P., He, X., Xu, T.L., Denko, M.K., Thulasiram, R.K., Yang, L.T. (eds.) ISPA Workshops 2007. LNCS, vol. 4743, pp. 123–135. Springer, Heidelberg (2007)
12. Gabarda, S., Cristóbal, G.: Detection of events in seismic time series by time-frequency methods. IET Signal Processing 4(4), 413–420 (2010)
13. Gottschalk, S., Lin, M.C., Manocha, D.: OBBTree: A hierarchical structure for rapid interference detection. In: Proc. 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 171–180. ACM (1996)
14. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Intl. Conf. on Management of Data, pp. 47–57 (1984)
15. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. Knowledge and Information Systems 3(3), 263–286 (2001)
16. Keogh, E., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowl. Inf. Syst. 7(3), 358–386 (2005)
17. Keogh, E., Xi, X., Wei, L., Ratanamahatana, C.: The UCR Time Series Classification/Clustering Homepage (2011)

18. Keogh, E.J., Lin, J., Fu, A.W.: HOT SAX: Efficiently finding the most unusual time series subsequence. In: IEEE Intl. Conf. on Data Mining, pp. 226–233 (2005)
19. Keogh, E.J., Lin, J., Hee Lee, S., Herle, H.V.: Finding the most unusual time series subsequence: algorithms and applications. Knowledge and Information Systems 11, 1–27 (2007)
20. Khanh, N.D.K., Anh, D.T.: Time series discord discovery using WAT algorithm and iSAX representation. In: Proc. Third Symposium on Information and Communication Technology, pp. 207–213. ACM (2012)
21. Liao, T.W.: Clustering of time series data: a survey. Pattern Recognition 38(11), 1857–1874 (2005)
22. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proc. 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 2–11 (2003)
23. Lin, J., Keogh, E., Truppel, W.: Clustering of streaming time series is meaningless. In: Proc. 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 56–65. ACM (2003)
24. Lin, J., Keogh, E.J., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Mining and Knowledge Discovery 15, 107–144 (2007)
25. Shieh, J., Keogh, E.: iSAX: indexing and mining terabyte sized time series. In: Proc. 14th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, pp. 623–631. ACM (2008)
26. Trenberth, K.E., Hoar, T.J.: The 1990-1995 El Niño-Southern oscillation event: Longest on record. Geophysical Research Letters 23(1), 57–60 (1996)
27. Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E.: Indexing multidimensional time-series with support for multiple distance measures. In: Proc. Ninth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, pp. 216–225. ACM (2003)