A fast algorithm for complex discord searches in time series: HOT SAX Time

Paolo Avogadro · Matteo Alessandro Dominoni

Abstract  Time series analysis is quickly proceeding towards long and complex tasks. In recent years, fast approximate algorithms for discord search have been proposed in order to compensate for the increasing size of the time series. It is more interesting, however, to find quick exact solutions. In this research, we improved HOT SAX by exploiting two main ideas: the warm-up process, and the similarity between sequences close in time. The resulting algorithm, called HOT SAX Time (HST), has been validated with real and synthetic time series, and successfully compared with HOT SAX, RRA, SCAMP, and DADD. The complexity of a discord search has been evaluated with a new indicator, the cost per sequence (cps), which allows one to compare searches on time series of different lengths. Numerical evidence suggests that two conditions are involved in determining the complexity of a discord search in a non-trivial way: the length of the discords, and the noise/signal ratio. In the case of complex searches, HST can be more than 100 times faster than HOT SAX, thus being at the forefront of the exact discord search.

Keywords  Time series, Anomaly, Discord, Nearest neighbor distance.
1 Introduction and related Works

Anomaly discovery in time series is an active research field (Chandola et al., 2009) (Gupta et al., 2014) where many approaches are taken into consideration.

One of the first steps when analysing a time series is to reduce its dimensionality. Symbolic aggregate approximation (SAX) (Lin et al., 2003) is a successful algorithm which goes in this direction associating each sequence to a symbolic sequence. Later on, an improved version, called iSAX, was presented allowing for better clusterizations (Shieh and Keogh, 2008).

Discords were introduced in 2005 (Keogh et al., 2005), where it was also suggested a fast algorithm for finding them: HOT SAX. The use of Haar wavelets (Bu et al., 2007) improved the pruning power in respect to SAX. HOTiSAX (Buu and Anh, 2011) is an algorithm based on iSAX which is up to four times faster than HOT SAX.

The most advanced methods for a complete characterization of the time series are those of the Matrix Profile series initiated in 2016 (Yeh et al., 2016). The algorithms of the Matrix Profile provide a quick calculation of the distance among all the sequences and as a result they can also return discords, however given this broad nature their complexity grows quadratically with the length of the time series. The research state of the art has been progressing in order to take into account for the increasing length of the time series and complexity of the tasks. We are now in the hundred million points era, for example regarding motif analysis (Zhu et al., 2018) and (Gao and Lin, 2018).

If the size of the time series is such that the RAM memory is not enough, one needs to keep at least part of the data on disk, and for this purpose it was developed the Disk-Aware Discord Discovery (DADD, also known as DRAG) algorithm (Yankov et al., 2008). MERLIN is a new algorithm based on DADD which can quickly scan all the discords within a given length range (Nakamura et al., 2020).

A first route that can be followed in order to speed up the calculation regards parallelizing the existing algorithms, as was done for example by Zymbler et al. (2019), where a version of HOT SAX has been developed for Intel Many-Core Systems. The fastest algorithm involving graphic card accelerators is SCAMP of the Matrix Profile series (Zimmerman et al., 2019). A different direction involves developing approximate discord algorithms. Rare Rule Anomaly (Senin et al., 2014), for example, exploits the Kolmogorov complexity of the SAX symbolic sequences to extract anomalies close to the definition of discords. At variance, a recent approximate discord search method (Thuy et al., 2018), based on segmentation and clustering, automatically suggests the size of the anomaly, and it is faster than HOT SAX. The overlap between these anomalies and discords is good, however if one searches for the first $k$–anomalies there is an increasing probability to get wrong results. All the algorithms which return approximate discords have this common drawback: the price to pay for an improved speed is a drop in accuracy in respect to an exact discord search.

Although there are many articles regarding discord search, there has not been an important attempt to understand why the heuristics perform well on some time series, while the results are poorer on others. It is thus desirable to obtain a general view of the problem for better understanding the nature of the anomalies. As a result, one
could obtain new ideas regarding time series and anomaly detection. Our methodology was to understand in detail the mechanism at the basis of HOT SAX and check if the neighborhood properties of the time series might help in finding shortcuts. Once we found areas where HOT SAX could be improved, we implemented the new functions and we checked if these ideas were able to produce significant improvements. We used HOT SAX as a comparison for our tests since it is a benchmark on the subject; we also included DADD and SCAMP, and a fast freely available approximate algorithm: RRA. Notice that our algorithm has not yet been parallelized, so all the comparisons are limited to the state of the art serial codes. During the evaluation process, the great diversity in computational costs between different discord searches became particularly striking. In order to disambiguate the roles of the different parameters of a discord search, we defined a quantity: the “cost per sequence” (cps) (detailed in Sec. 4.2). The cps allows one to compare discord searches on time series of different length and rate them in terms of complexity. Following this new definition, we were able to notice that HST is particularly faster than HOT SAX at finding discords for complex searches.

The rest of the paper is organized as follows:

– After clarifying the notations and defining useful quantities, Sec. 2 provides a brief explanation of the brute force approach and of HOT SAX.
– Sec. 3 details HST and the rationale of the modifications in respect to HOT SAX.
– Sec. 4 compares the execution speeds of HST and its main available competitors. We introduce a new indicator: the cost per sequence of a discord search and we detail cases where HST is particularly fit for the job.
– Sec. 5 contains the conclusions and the future work.

2 Theoretical background

2.1 Terminology

– The points of a time series are denoted with \( p_j \), where the subscript indicates the time.
– A sequence, containing \( s \) points, is described with the time of its first point. For example, the points of sequence \( k \) are: \( p_k, p_{k+1}, \ldots, p_{k+s-1} \). In this paper, unless specifically stated, we consider z-normalized sequences (Keogh and Kasetty, 2003).
– If the number of points of a time series is denoted with \( N_{\text{tot}} \), the number of sequences of length \( s \) therein is smaller, and denoted with the letter \( N = N_{\text{tot}} - s + 1 \). The last sequences are not complete and so they are removed from the search space.
– The Euclidean distance among two sequences \( k \) and \( l \) is:

\[
d(k, l) = \sqrt{\sum_{i=0}^{s-1} (p_{k+i} - p_{l+i})^2}. \quad (1)
\]

The process of z-normalization implies a large memory expenditure. It is possible to save memory by storing the averages and standard deviations of all of the
sequences and proceed with the following function instead:

\[ d(k, l) = \sqrt{\sum_{i=0}^{s-1} \left( \frac{p_{k+i} - \mu_k}{\sigma_k} - \frac{p_{l+i} - \mu_l}{\sigma_l} \right)^2}, \]  

as pointed out by [Zhu et al. (2018)](https://www.sciencedirect.com/science/article/abs/pii/S0004370217312618), the same result can be obtained with the help of the scalar product:

\[ d(k, l) = 2s \left( 1 - k \cdot l - s\mu_k \mu_l \right), \]  

the notation \( k \cdot l \) is associated with the scalar product between the two sequences (vectors), \( s \) is the length of the sequences, \( \mu_k \) and \( \sigma_k \) are the mean and standard deviation of sequence \( k \).

- The nearest neighbor distance (\( \text{nnd} \)) for sequence \( k \) is obtained as:

\[ \text{nnd}(k) = \min_{j:|k-j| \geq s} d(k, j). \]  

The search space for the minimization includes all the sequences, \( j \), of the time series, with the exception of those that overlap \( (j:|k-j| \geq s) \) with \( k \). This non self-match condition serves in order to avoid spurious low values of \( \text{nnd} \) due to overlaps.

- The quantity \( \text{ngh}(i) \) returns the position of the nearest neighbor of sequence \( i \):

\[ \text{nnd}(i) = d(i, \text{ngh}(i)). \]  

- The \( \text{nnd} \) profile is the set of the current \( \text{nnds} \) of all the sequences of a time series. In practice many of the \( \text{nnds} \) returned by HST are only approximate values. Had these values been exact, the \( \text{nnd} \) profile would be identical to a special case of Matrix Profile: the self-similarity join profile, or \( \text{PAA} \) [Yeh et al. (2016)](https://www.sciencedirect.com/science/article/pii/S0004370217312618).

- In order to highlight that some quantities are stored in an array or variable of the code, we use typewriter characters. For example \( \text{nnd}[i] \), or \( \text{ngh}[i] \) are the arrays containing the information regarding the \( \text{nnd}(i) \) and \( \text{ngh}(i) \).

- In the rest of the paper we will refer to time-distance between two sequences \( i, k \) as \( |i-k| \). For example the nearest time-neighbors of sequence \( i \) are \( i-1 \) and \( i+1 \).

- In order to compare the performance of two algorithms we can resort at using the distance-speedup (D-speedup) calculated as the ratio of the number of distance calls between the algorithms. We can also use the time-speedup (T-speedup) obtained as the ratio of the runtimes of the two algorithms on the same dataset for the same task.

2.2 Discord

The concept of discord follows an intuitive idea regarding anomaly search in time series. A sequence of length \( s \) can be considered as an \( s \)-dimensional vector (or point of an \( s \)-dimensional space). With this view, an isolated point is an anomaly. In practice,
the discord is defined as the sequence with the highest \( nnd \) in respect to all the other sequences of the time series.

\[
discord = \arg \max_i (nnd(i))
\]

The second discord is defined as the sequence with the highest \( nnd \) as long as it does not overlap the first discord. The \( k \)-th discord is the sequence with the highest \( nnd \) as long as it does not overlap any of the previous \( k - 1 \) discords.

We will often call a sequence a *good discord candidate* if it has the highest \( nnd \) after its distance has been calculated with all the others during the execution of the algorithm. According to this definition, the last sequence which becomes a *good discord candidate* will be the discord.

### 2.3 Brute Force approach

A brute force algorithm for discord search requires two nested loops:

1. The external one runs on all the sequences of the time series. It is a maximization procedure for finding the sequence with the highest \( nnd \).
2. The internal loop provides the minimization procedure for obtaining the nearest neighbor distance of each sequence. This process involves the calculation of the distances between the selected sequence and all the others (excluding self-matches).

As a result, the complexity of a brute force calculation as a function of the number of sequences \( N \) is \( O(N^2) \).

### 2.4 HOT SAX

HOT SAX shows a way to skip a large part of the distance calculations of a brute force approach. Via the symbolic aggregate approximation (SAX) [Lin et al., 2003] it is possible to quickly assign every sequence to a much shorter symbolic sequence (or SAX cluster). This dimensionality reduction procedure clusterizes sequences efficiently. Because of the properties of SAX, sequences belonging to the same SAX cluster can also be Euclidean neighbors, and it is possible to exploit this fact for skipping most of the calculations. In order to improve the brute force approach, the outer and inner loops are re-arranged following the indications obtained with SAX. The external loop is now structured according to the SAX clusters: from the smallest to the biggest ones (containing more sequences). The order of the internal loop is dynamic, depending on the current sequence of the external loop. If the external loop has arrived on a sequence belonging to a given cluster, the sequences of that cluster are set at the beginning of the internal loop. The remaining part of the inner loop follows a pseudo-random order. At any point in the inner loop, as soon as the \( nnd \) of the candidate sequence becomes smaller than the best so far value, the rest of the inner loop can be skipped (since the sequence under observation cannot be a discord). The
rationale of these choices is that small clusters, containing only a few sequences, are good candidates for finding “isolated sequences” or sequences for which the \( nnd \) is high. Moreover, close Euclidean neighbors are likely found in the same cluster of a sequence. If one finds a close neighbor of a sequence, its approximate \( nnd \) can become smaller than the current best (highest) one, allowing one to skip the remaining distance calls of the inner loop. Those calls, in fact, can only lower than the actual \( nnd \) value of the sequence.

3 HOT SAX Time

3.1 The model to be improved

In the following, we will detail how to obtain the new algorithm from HOT SAX and the origin of these modifications.

In order to improve HOT SAX we need a model regarding discords and their search. Let’s recall the discord definition as the sequence with the highest \( nnd \) value. Following this definition, one can think of a discord search as a way to find the maximum of a profile when the profile is not known beforehand. An intuitive way to see HOT SAX is the following. SAX provides a sort of hazy vision of the \( nnd \) profile where one can see that there are peaks (small clusters) but it is not possible to distinguish their height. At this point one has an idea regarding where to search the discords. The inner loop clarifies this view by returning an approximate \( nnd \) value for each of the sequences. If the sequence goes through the whole inner loop, its \( nnd \) is exact and the sequence is a good discord candidate. However, the cost for each one of these “perfect clarifications” (the number of distance calls) is essentially identical to the number of sequences the time series. The total cost of the “clarification” process determines how difficult it is to find the discord. If one is sure that the approximate \( nnds \) of all the other sequences are lower than the best so far candidate, that sequence is the discord.

In the rest of the paper we will follow the same general mechanism, but we will try to improve the “view” which guides the search process. With the help of the similarity between time-close sequences, we will be able to obtain a better idea of the position of the discords with an indication of their \( nnds \). At this point, the search will become easier since most of the sequences, having a low \( nnd \), will be discarded immediately.

3.2 Speed-up the k-th discord

Here we detail a well-known technique \([\text{Bu et al.}, 2007]\) that can be used to diminish the number of calculations for the \( k \)-th discord (once the first \( k - 1 \) discords have been found). We will use it later to find a good \( nnd \) profile at a low computational cost.

It is useful to remind the concept of \( k \)-th discord, which is the sequence with the highest \( nnd \) non-overlapping any of the previous \( k - 1 \) discords. During the calculation of each discord, the code should update the approximate \( nnd \) values for all the
sequences. These quantities are obtained by refreshing the \textit{nnd}s in the inner loop. The memory cost of keeping track of the approximate \textit{nnd}s is just the size of the time series \(O(N)\). Although these approximate \textit{nnd}s are rough estimates of the exact ones, they are very useful, since the real anomalies (discords) have \textit{nnd} values usually quite higher than those of the “common” sequences. During the search for the second discord, even before starting the inner loop for a sequence, one should check its current approximate \textit{nnd}. If the sequence under consideration exhibits an \textit{nnd} value lower than the present best value, the whole inner loop of that sequence can be skipped (since it is a minimization). Thanks to this simple procedure, it is possible to skip most of the calculations for the \(k\)-th discord. In the next section, we provide a solution that allows applying the aforementioned technique also in the case of the first discord.

3.3 Warm-up

In this section we explain the first main procedure at the basis of HST. It would be particularly interesting to exploit the idea of the previous section also at the beginning of the search. Unfortunately, at the time of the calculation of the first discord there is no available approximate \textit{nnd} value for all the sequences. We now detail a fast procedure to build such an approximate \textit{nnd} profile. The idea is to find a new
order where similar, non overlapping sequences are one after another. Once this new ordering has been obtained one can perform a chain of calls to the distance function from one sequence to the next one and so on. As a result, for each sequence there are two distance calls to two “neighbors” and thus it is possible to obtain a rough \textit{nnd} profile. The total number of distance calls for this procedure is essentially equal to the number sequences of the time series.

Since the rationale of HOT SAX is that sequences belonging to the same cluster are also likely Euclidean neighbors, it becomes natural to calculate the distance between two sequences within the same cluster.

Let’s consider a situation in which all the sequences have already been grouped in clusters by SAX. There can be different variations regarding how to retrieve the sequences from the SAX clusters, and for some of them, the temporal order of the sequences is preserved. This can be a problem since we want to obtain a chain of distance calls, and time consecutive sequences are self-matches (for which the distance should not be calculated). In order to avoid this problem, it is useful to shuffle randomly the sequences. With this simple procedure one can avoid many of the possible self-matches and increase the number of “valid” distance calls. Sometimes the randomization procedure cannot avoid self-matches. For example, if a cluster contains only 3 sequences (of length 10) beginning at time 67, 73, and 75, non-self-match distance calls among them are impossible (independent of their order). Before the warm-up procedure we initialize all of the \textit{nnd}s with a very high value. With this choice, no possible discord candidate is neglected. The last sequence of a cluster is coupled with the first one of the next cluster.

To summarize, the warm-up procedure (see Fig. 1, left) consists of 3 steps (and 2 pre-requisites: the initialization of the \textit{nnd}s, and the SAX procedure):

1. Shuffle the sequences randomly.
2. Build a new order for the sequences by placing the clusters one after the other following their size (from the smallest to the biggest).
3. Calculate the distance between consecutive sequences following the new order. Avoid calculating the distance in case of self-match.

After the Warm-up, each sequence has an approximate \textit{nnd}. This procedure resembles the original HOT SAX, since we are essentially calculating distances among sequences belonging to the same cluster. A real advantage can be obtained only if the approximate \textit{nnds} can be improved substantially at a low computational cost.

### 3.4 Short range time topology

Here we detail how to exploit another property of the time series in order to improve the approximate \textit{nnds} obtained with the warm up procedure.

Time series display a form of time correlation first described as Consecutive Neighborhood Preserving (CNP) Property \cite{Zhu2018}. This property can be regarded as a form of time topology which makes possible to improve an approximate \textit{nnd} profile of a time series at a low cost \cite{AvogadroDominoni2019}. This can be summarized by noticing that there is a high chance that the Euclidean nearest
neighbors of two time-close sequences (i.e. starting at time \( i \) and \( i + 1 \)) are also time-close, as in Fig. 1 (right). In practice, this can be seen as a form of autocorrelation of the \( nnds \) which is particularly noticeable regarding the \( ngh \) profile where, often times, the following formula is true:

\[
ngh(i+1) = ngh(i) + 1.
\]  

(5)

This property can be used to improve the low quality \( nnds \) obtained with the warm-up, where Eq. 5 is almost never verified. In fact, it is enough to check the following distance: \( d(i+1, ngh(i)+1) \) for all of the sequences in order to obtain a better \( nmd \) profile closer to the exact matrix profile. Since the time ordering is irrelevant when searching for the nearest neighbor, the opposite direction can also be employed: \( d(i-1, ngh(i)-1) \). Also in this case, the number of distance calls is essentially identical to the size of the time series. In order to avoid useless calculations, before calculating the distance, it can be checked if it is already true that the neighbor of sequence at \( i \pm 1 \) is at \( ngh(i) \pm 1 \). This new \( nmd \) profile is much better than the one resulting from the warm-up procedure. We can now use the \( nnds \) to reorder the loops for the discord search.

3.5 Rearranging the external loop

HOT SAX begins the external loop with likely discords, this is implemented by selecting at first those sequences that belong to small SAX clusters.

3.5.1 Initial re-ordering process

For HST, we want to follow the same reasoning used in HOT SAX but enhance it. After the warm-up procedure and supplementing it with the time topology (Sec. 3.4) all the sequences have an approximate \( nnd \) value. One is sure to make a reasonable guess putting at the beginning of the external loop those sequences which have a high \( nnd \) value, and this choice is likely better than just picking the smallest SAX clusters. However, this \( nmd \) profile is still rough, so it is better to smear it with a moving average. It has been shown, in fact, that the exact \( nmd \) profile (matrix profile) displays a “sort of smoothness” \cite{Zhu2018,Avogadro2019}. A rough \( nmd \) profile is likely to contain also spikes surrounded by dips. These spikes correspond to “false discord candidates”. At variance, a true good discord candidate should belong to a “peak”: it should be surrounded by quite a few other sequences with high \( nnd \) values. This time coherence of the \( nnds \) spans approximately the length of a sequence \cite{Avogadro2019}, for this reason the moving average takes into account \( s + 1 \) sequences:

\[
\overline{nnd}(i) = \frac{1}{s+1} \sum_{j=-s/2}^{s/2} nnd(i+j)
\]  

(6)
With this procedure, one gets rid of spikes not related to peaks. At the borders, where
the moving average cannot be done, we simply use the approximate nds. In summary, at the beginning of the external loop HST puts the sequences with the highest
nds, and at the end those with the lowest values.

3.5.2 Dynamic external loop order

In HOT SAX, the external loop is fixed once for all. In HST the order of the external
loop changes dynamically during the search. The approximate nds, in fact, become
smaller and smaller converging to the exact ones during the execution of the algo-

rithm. As a result, those sequences that are more likely discords change during the
execution of the code. It becomes reasonable to re-arrange the remaining parts of ex-
ternal loop in order to prompt high nds sequences at the beginning. This procedure
takes into account the increasing information obtained as the algorithm proceeds. In
this case, using a moving average does not produce good results since the quality of
the approximation is becoming more accurate (a moving average would just diminish
its precision). Every time that a good discord candidate is found there is a distance
call for (almost) all the sequences of the time series. So it is likely that many nds
have changed during the process. For this reason, finding a good discord candidate is
a natural point after which HST re-orders the remaining part of the external loop.

3.6 Long range time topology

During the execution of HST we make use of the CNT property in order to solve an-
other problem. Let’s consider what happens when a good discord candidate is found
during the search process. The “pseudo-smoothness” of the nd profile induced by
the similarity between time-close sequences implies that a discord is not an isolated
sequence but it belongs to a wide peak where time-close sequences show high nd
values, so:

- In the peak there are many sequences having a high nd value. For each of them
  the inner loop might require to scan most of the other sequences.
- The width of the peak is not known.
- The distance calls of a sequence do not contribute to the nds of its time-neighbors
  (they are self-matches). For this reason one needs to perform independent calcu-
lations for each of the sequences of the peak.

The width of the peak (the number of sequences which belong to it), is likely propor-
tional to the length of the sequences. This is a result of the non-self-match condition,
since overlapping sequences do not contribute to each others nd and have similar
properties. The time topology is very useful since it tells us that the Euclidean neigh-

bors of: $i+1, i-1, i+2, i-2, \ldots$ are likely time-close. This means that in many cases
the following conditions are true:

- $\text{ngn}(i+1) = \text{ngn}(i) + 1$
- $\text{ngn}(i-1) = \text{ngn}(i) - 1$
- $\text{ngn}(i+2) = \text{ngn}(i) + 2$
A fast algorithm for complex discord searches in time series: HOT SAX Time

Fig. 2: Sequence 89268 has the highest approximate \( nnd \) value (9.94) of a time series, and it is surrounded by a peak about 30 sequences wide. It is a good discord candidate so its distance with all the other sequences (but self-matches) has been calculated (\( \approx 2 \cdot 10^5 \) calls). As a result its \( nnd \) drops (green arrow) to the exact value 6.97 (the green dot). The \( nnds \) of its time neighbors (red curve) are not affected (since they are self-matches). The long range time topology improves the approximate \( nnds \) of many of the time neighbors of sequence 89268 (blue dashed line) with a limited number of distance calls (\( \leq 2 \cdot s \)).

With these suggestions, it is possible to make a very limited number of calls to the distance function and likely obtain much better \( nnds \) for all of the sequences of the peak. In practice, the \texttt{long\_range\_time\_topology} functions essentially remove peaks as shown in Fig. 2. Notice that within the present algorithm one does not need to know the exact \( nnd \)s of the sequences of the peak. It is enough to find if these values are lower than the current highest exact one. It is important for these functions to be well balanced, in order to avoid useless calculations. A function of this kind contains a loop running on the \( s \) time-closest sequences of the one under consideration, exactly because they overlap (beginning at position \( i \), see Listing 1). If \( nnd[i+j] < \text{bestDist} \) (the highest so far exact \( nnd \) value) there is no reason to try to improve it, since we already know that sequence \( i+j \) cannot be a discord. Otherwise the algorithm checks the distance between \( i+j \) and \( \text{ngh}[i+j] \).

It can happen that: \( nnd[i+j] < d(\ i+j\ ,\ \text{ngh}[i]+j) \). In this case, the time topology is losing coherence. It becomes reasonable to skip the remaining part of the loop. Going to longer ranges would simply result in useless distance calls, i.e. slowing down the execution of the algorithm. Notice that both time directions should be taken into account. These procedures are embedded in two functions:

- \texttt{Long\_range\_time\_topology\_forw} (see Listing 1)
Listing 1: Long range time topology forw

```plaintext
for j=1; j <= s; j++ do
    if (nnd[i+j] < bestDist) break // not a discord: check next one
    if (ngh[i+j] = ngh[i]+j) return // distance already calculated
    if (i+j > N) return // outside time series limits
    if (ngh[i]+j > N) return // outside time series limits
    d = distance (i+j, ngh[i]+j) // calculate distance
    if (d < nnd[i+j]) then
        nnd[i+j] = d // update distance
        ngh[i+j] = ngh[i]+j // update neighbor
    else
        return // the time topology provides no improvement!
end if
end for
```

- Long range time topology back.

3.7 Pseudocode: details

Here we detail the functions of HST as described in Listing 2. The following functions are called before the external loop:

1. SAX(), this function clusters the sequences \textcolor{blue}{[Lin et al., 2003]}.
2. The \textit{Warm-up()} procedure obtains approximate nnds for most of the sequences, as explained in Sec. 3.3. The nnds are stored in the homonym integer array nnd[], while the neighbors (of each sequence) are stored in the integer array ngh[].
3. The \textit{Short range time topology()} function improves the quality of the nnds (Sec. 3.4).
4. The \textit{Sort External(order)} function creates an integer array containing the order of the sequences of the external loop, from the highest nnd to the lowest. This new order is stored into the integer array order[].

Notice that no exact nnd value has been found before the beginning of the external loop (up to line 6 of Listing 2). The double precision variable bestDist stores the current highest exact nnd. At the beginning, this quantity is initialized to 0 (the external loop is a maximization procedure so we initialize it with the lowest possible value). As a result the first sequence of the external loop will become the first good discord candidate and its exact nnd will turn into the next bestDist. The index j denotes the external loop and it runs from 1 to N (all the possible sequences). The order of the sequences of the external loop follows instead the index i=order[j]. In principle every sequence can be the discord, so the flag can_be_discord is set to true.

The function Avoid\textit{low nnds}() checks if nnd[i] is smaller than bestDist (the highest so far exact nnd), in that case the flag can_be_discord is set to false.
A fast algorithm for complex discord searches in time series: HOT SAX Time

Listing 2: HST Pseudocode

```java
nnd = 99999999.9 // initialize array with all the nnds
SAX() // Warm-up()
Short_range_time_topology()
Sort_External(order) // fill the array with the new ordering
bestDist = 0.0 // initialize the current max nnd-value
for j=1; j<=N; j++ do // the external loop
  i = order[j] // “i” is the sequence processed now
  can_be_discord = true // sequence “i” can be the discord
  Avoid_low_nnds (i, bestDist) // skip sequences with low nnd
  if (can_be_discord == true) then // is “i” still a candidate?
    Current_cluster(i, bestDist)// minimization (HOT SAX inner loop)
  endif
  if (can_be_discord == true) then // is “i” still a candidate?
    Other_clusters(i, bestDist)// minimization (HOT SAX inner loop)
  endif
  Long_range_time_topology_forw (i, bestDist) // level peaks
  Long_range_time_topology_back (i, bestDist) // level peaks
  if (can_be_discord == true) then // i is a good discord candidate
    Update(bestDist) // update the current max nnd-value
  endif
  Sort_Remaining_Ext(order) // re-sort the external loop
end for // external loop
```

The minimization phase comprises of two functions, scanning different parts of the search space:

1. The function `Current_cluster()` calculates the distance between sequence \(i\) and all the other sequences belonging to the same SAX cluster (but for self-matches). If the approximate \(nnd\) of sequence \(i\) drops below the \(bestDist\), then sequence \(i\) cannot be a discord, the function returns control to the main program, setting the flag `can_be_discord` to `false`.

2. The function `Other_clusters()` calculates the distance between sequence \(i\) and all the other sequences of the time series. It begins scanning small clusters and then it moves to bigger ones (but skipping the cluster already checked in the function `Current_cluster`). At any point, if the approximate \(nnd(i)\) becomes smaller than `bestDist`, the function sets the flag `can_be_discord` to `false` and then returns. Instead, when this function goes through all the remaining sequences and the flag `can_be_discord` is still `true`, we are sure that \(nnd[i]\) is exact. The minimization procedure associated to sequence \(i\) ends with this function. So \(i\) is a good discord candidate.
The former two functions are almost identical to those of the inner loop of a normal HOT SAX execution. In HST, however, before passing to the next sequence of the external loop there are a few more steps:

- The functions `Long_range_time_topology_forw()` and `Long_range_time_topology_back()` improve the `nnds` of the time neighbors of the sequence `i` (sequences beginning at `i + 1, i − 1, i + 2 ...`) as explained in Sec. 3.6.
- If the flag `can_be_discord` is still true at the end of the inner loop, it means that `nnd(i)` is exact and is the highest value so far. It is thus necessary to update the best so far `nnd` value (`bestDist`), with the help of the function `Update()`.
- The condition `can_be_discord = true` after the minimization procedure implies that essentially all the sequences were involved in at least one distance call. It is now a good point where to re-order the array governing the external loop (`order`) as per Sec. 3.5.2. The remaining sequences of the external loop are re-arranged according to the values of their updated `nnds`: from those with the highest `nnds` to those with the lowest values.

4 Validation

The aim of this section is to provide comparisons with direct competitors of HST. Since HST has not yet been parallelized we will limit the comparisons only to serial algorithms.

There are two major families of metrics. The first one is related to the accuracy of the algorithm itself, i.e. to which extent the returned anomalies are discords (approximate algorithms do not return exact values). For example, this can be obtained as the overlap of the discords and the anomalies found. HOT SAX Time, on the other hand, does not need these tests since it returns the exact discords.

The other usual metric is a measure of the speed. Although one is interested in the amount of seconds required to find the anomaly, the actual calculation time can be masked by many factors (e.g. programming language, hardware, compilation flags...). As a result, the speed is often calculated as the number of calls to the distance function between sequences (Keogh et al., 2005) (lower number of calls is better). In the past, more than 99% of the calculation time was spent within this function. However, if one uses the distance function based on the scalar product as suggested by Zhu et al. (2018) (which is faster than the explicit distance among z-normalized sequences), this condition is no longer true, in particular if the number of calls is small. Although the time spent in the distance function is still most of the total, other functions add a non-negligible contribution to the total calculation time. In these cases both the number of distance calls and the running times might be needed. In order to compare HST with other algorithms we use the ratio of the distance calls (D-speedup) or the ratio of the running times (T-speedup) (see Sec. 2). If the calculation times are long enough (at least tens of seconds) the two indicators are expected to be close (within 20%-30%). At variance, when the running times are short (≈ 1s), the T-speedup is not particularly a good indicator since it is affected by other functions that require tenths of seconds.
Table 1: The first column contains the names of the datasets, then we include the lengths of the sequences (s), the SAX parameters (P, alphabet) and the length of the time series. The number of distance calls for the first discord for HOT SAX and HST (lower is better) are in the columns with the names of the two algorithms. The D-speedup shows that in all the cases under observation HST is at least two times faster than HOT SAX, on four occasions it is more than 5 times faster and for three datasets it is more than 9 times faster. The last column shows the running times for HST in seconds.

If the number of distance calls is either not available or not accurate, like in the case of SCAMP and DADD, we will consider only the execution times. Notice that the available implementations of these two algorithms are written in C/C++. HST instead, has been implemented in Fortran. This is not an important problem since these programming languages have very similar performances, as it can be seen for example at The Computer Language Benchmarks Game. In the case of RRA the available implementation is written in JAVA, this can slow down the execution times and for this algorithm, the comparisons are limited to the number of distance calls. The datasets used to validate RRA (Senin et al., 2018) (Senin et al., 2015) are a heterogeneous collection of time series often used in literature, ranging from ECG to breathing time series. Their length goes from few thousand to a few hundred thousand points. For this reason, we also used them in the present article. Some of these time series are extracts of longer datasets, for example those of the Physionet repository (Moody and Mark, 2001) (Goldberger et al., 2000). In order to provide a uniform comparison with RRA we stick with the ones already used for its validation. We included a real dataset containing more than 10^8 points (Willett et al., 2016). We also used synthetic time series since they allow us to have full control of their properties.

It is important to say that all these algorithms (but SCAMP) include pseudo random processes, so the number of distance calls for a given dataset fluctuates. Determining the speed of the algorithm with a single test might be misleading. We overcome this problem by averaging 10 runs on each dataset.
At the beginning, we compare HOT SAX and HST. The reference HOT SAX code (written by us) has many similarities with its HST counterpart, it is written in the same language and many of the subroutines are essentially identical. We consider the same datasets and conditions that were used to compare RRA and HOT SAX (Tab. 1). In 8 cases out of 14, HST is more than 5 times faster than HOT SAX, reaching D-speedup peaks of 13 for two datasets. For these calculations, it is not particularly meaningful to consider the T-speedup. The reason is that the execution times for the first discords are very short. As a result, an important percentage of the running time is spent in functions other than the distance one. We keep these tests since they have the same structure as those used for the evaluation of RRA Senin et al. (2015). In order to obtain conditions that represent better the speed difference between the two algorithms, we expand the calculation over 10 discords. This is legitimate since both HOT SAX and HST report exact values. Notice that the “non self-match condition” implies that the total number of discords in a dataset is limited. At most, there can be \(\left(\frac{N}{s}\right) + 1\) discords. For these reasons, we excluded ECG 308 and ECG 0606. In Tab. 2 we can see that HST is substantially faster than HOT SAX. The runtimes for HST are still short and so there is an important impact from other functions. Only for the longer time series (EGG 300 and ECG 318), HST requires more than 1 second for calculating the first 10 discords. In those cases, the D-speedup and T-speedup are within 20% of each other, while for shorter calculations the gap can be wider. Although for quick calculations the T-speedup is not very accurate we can use the D-speedup as a guide to better understand the kind of problems where HST is faster. When the task becomes more demanding (10 discords instead of 1), HST becomes from 4 to 19 times faster than HOT SAX, in terms of distance calls and 3 to 15 times faster in terms of runtimes. At this point, one might be interested in understanding the characteristics of the discord search which determine this variety of results.

### Table 2: The number of distance calls and running times for the first 10 discords for HOT SAX and HST.

| file          | # of distance calls | Runtimes [s] |          |          |          |
|---------------|---------------------|--------------|----------|----------|----------|
|               | HOT SAX             | HST          | D-speedup| HOT SAX  | HST      | T-speedup |
| Daily commute | 4 373 481           | 819 880      | 3.33     | 1.78     | 0.45     | 3.97      |
| Dutch Power   | 20 326 437          | 1 043 572    | 19.48    | 14.40    | 0.94     | 15.29     |
| ECG 15        | 10 947 552          | 705 152      | 15.53    | 3.64     | 0.30     | 12.26     |
| ECG 108       | 10 194 725          | 856 132      | 11.91    | 4.07     | 0.73     | 5.59      |
| ECG 300       | 447 184 547         | 44 697 489   | 10.00    | 147.49   | 17.14    | 8.60      |
| ECG 318       | 269 580 847         | 37 740 624   | 7.14     | 90.99    | 14.54    | 6.26      |
| NPRS 43       | 1 005 254           | 187 478      | 5.36     | 0.20     | 0.056    | 3.64      |
| NPRS 44       | 6 748 679           | 1 666 487    | 4.05     | 1.13     | 0.45     | 2.52      |
| Video         | 2 742 811           | 481 800      | 5.69     | 0.62     | 0.15     | 4.05      |
| Shuttle, TEK 14 | 1 500 550       | 265 364      | 5.65     | 0.34     | 0.086    | 3.98      |
| Shuttle, TEK 16 | 1 613 129      | 274 172      | 5.88     | 0.38     | 0.095    | 3.98      |
| Shuttle, TEK 17 | 1 460 009       | 276 351      | 5.28     | 0.33     | 0.096    | 3.50      |

4.1 HOT SAX and HST
4.2 The complexity of a search: cost per sequence

In this section, we provide an analysis that will lead to the characterization of complex searches. We need an indicator that allows us to compare searches on time series of different lengths and according to the different parameters of the search. There are two main approaches regarding algorithms and the problems to be solved:

- One can study the asymptotic complexity of the algorithm, as a function of some parameters of the problem, for example the size of the dataset. This approach is particularly useful when the algorithms do not depend on the data to be analyzed (like the Matrix Profile ones), since it allows to precisely compare different algorithms.
- If the execution is sensitive to the specific dataset under investigation, one can use the execution properties (time, space required,...) to characterize specific instances of the problem. For example, one can order the problems in terms of difficulty.

In this section, we take the second point of view since we want to characterize the problems. A discord search is influenced by:

- the content of the time series under investigation.
- the length of the time series $N$.
- the length of the sequences $s$.
- the number of discords to be found.

In the following, we will define an indicator to measure the difficulty of discord searches. As a benchmark, we will use the results of HOT SAX, since it is popular, easy to implement, and it is not optimized on a specific kind of time series. Moreover, both HST and HOT SAX are based on SAX, so it is safe to compare two searches when the SAX parameters are identical. For some searches, HOT SAX already represents a valid solution, while we are interested in understating in which instances HST is much better.

Since the number of distance calls grows with the length of the time series, one might think that also the speedup between HST and HOT SAX might grow with $N$. The experiments, however, do not confirm this hypothesis. In Tab. 1, the best results of HST have been obtained with ECG 108 whose length is 21600 (speedup of 13 in respect to HOT SAX). At variance ECG 300 is more than 24 times longer than ECG 108 but the speedup is about 7. Since the length of the time series is not the main parameter determining the speedup, we will now try to find other quantities with a more clear impact. The performance difference is likely related to both the structure of the signal and the length of the sequences. The total number of distance calls naturally grows with the length of the time series since each sequence must be checked. For this reason, it is meaningless to rank the discord searches of two time series having very different lengths, for example one containing $10^6$ points and the other $10^8$ points. However, some discord searches are clearly easier than others (even if the number of points is big). For example, if one time series is constant but for a single bump, even a person at first glance can identify the anomaly with the bump. On the opposite side, there are discord searches on “short” time series which require
lengthy computations even on fast machines. For these reasons, we define the cost per sequence \((cps)\) as the number of distance calls needed to find the first k-discords divided by the total number of sequences \(N\) and by the number of discords \(k\):

\[
\text{cost per sequence} = cps = \frac{\# \text{ of distance calls}}{Nk}.
\]

The \(cps\) can be used to compare searches on time series of different lengths and to order them in terms of difficulty. In general, there are two complementary interpretations of the \(cps\):

- It is the average number of distance calls per sequence and per discord.
- If the exact \(nnd\) profile (matrix profile) has a few big “bumps”, and if it is possible to identify with little computational cost (1 distance call each) those sequences which are “normal” (the background with low \(nnd\) values), then the \(cps\) can be considered as a hint regarding the quantity of “good discord candidates” (bumps) encountered during the search process, since each them requires about \(N\) distance calls.

Let’s recall that HOT SAX is based on a “magic” ordering of the loops which allows one to skip most of the calls of a brute force approach. From this perspective, the \(cps\) can be seen as the inverse of the “magic”: a low \(cps\) is an indication that the “magic” ordering is working well, while if the \(cps\) is high, more distance calls are needed and the “magic” is less effective. Let’s consider two opposite cases. If one has access to a “perfect magic” all the sequences which are not the discord require just one distance call to a close neighbor to be discarded, while only for the discord it is required to calculate the distance with all of the \(N - 1\) sequences. In this case, the total number of calls is \(2(N - 1)\), and the \(cps\) \(\approx 2\). In an unfortunate event where the “magic” ordering fails completely, one essentially re-obtains a brute force approach that returns the maximum possible \(cps\) \(\approx N\).

Although the \(cps\) is a rather intuitive concept, as far as the authors know, it has never been formalized or explicitly used to classify the complexity of discord searches.

We will now use HOT SAX \(cps\) for defining complex discord searches. It is possible to notice that HOT SAX \(cps\) can exhibit strong variations, while HST \(cps\) is more stable. For better visualizing this fact we set at the beginning of Tab. those datasets where HOT SAX performs better, and at the end those which require more calculations. For these problems, HST \(cps\) is in between 4 and 15. Notice that the Warm-up() and Short_range_time_topology() procedures already account for about 2 distance calls per sequence. Even in the case in which the first sequence to be analysed turns out to be the discord, HST should require at least \(cps \sim 3\). For less “complex searches” (those for which HOT SAX needs less than 20 distance calls per sequence), the maximum speedup obtainable by HST is limited by the structure of the algorithm itself. For example, HOT SAX requires just 9 distance calls per sequence for ECG 0606. The maximum theoretical D-speedup attainable in this case by HST is \(\sim 3\); while the actual D-speedup is 2.52. In fact, we can notice that for none of the time series with low \(cps\) (less than 20) the D-speedup is higher than 3. While for all the sequences with a cost per sequence equal to or higher than 67 the D-speedup is greater than 6, reaching peaks of 13 in two cases.
| file       | Cost per sequence | D-speedup |
|------------|-------------------|-----------|
| ECG 0606   | 9                 | 4         | 2.52     |
| ECG 15     | 14                | 6         | 2.35     |
| NPRS 44    | 16                | 6         | 2.91     |
| Video      | 19                | 8         | 2.30     |
| NPRS 43    | 20                | 9         | 2.23     |
| ECG 308    | 28                | 5         | 5.75     |
| Daily Com. | 48                | 15        | 3.14     |
| ECG 108    | 67                | 5         | 13.65    |
| ECG 318    | 80                | 8         | 10.58    |
| ECG 300    | 87                | 12        | 7.08     |
| Shuttle, TEK 17 | 95            | 14        | 6.67     |
| Dutch Power | 98               | 7         | 13.19    |
| Shuttle, TEK 14 | 98            | 13        | 7.50     |
| Shuttle, TEK 16 | 109           | 14        | 7.81     |

Table 3: The second column shows the “cost per sequence” \( k = 1 \) expressed as the number of distance calls over the time series length (for HOT SAX) rounded to the first integer value. The third column contains the same quantity but referred to HST. The fourth column contains the D-speedup. In this table, the files are ordered according to an increasing HOT SAX cost per sequence.

4.2.1 Cost per sequence and “easy-looking” time series

It is interesting to notice that HOT SAX requires a lot of distance calls on many time series which look “easier” from a human point of view. For example the Shuttle Marotta Valve time series, TEK14, TEK 16, and TEK 17, appear simpler if compared with the respiration time series NPRS 43 and NPRS 44. As a comparison, we report NPRS 43 and TEK 14 in Fig. 4(right) since they have approximately the same length: TEK 14 shows 5 main patterns, while the interpretation of NPRS 43 is less easy. We can further investigate this anti-intuitive behavior with the help of synthetic time series. In the following, we consider a very simple function consisting of a re-scaled sine function plus a uniform random noise:

\[
p_i = \frac{\sin(0.1 \cdot i) + E \cdot \epsilon + 1}{2.5}
\]  

(7)
Table 4: Number of distance calls and cps (for one discord, \(k = 1\)) as a function of the noise amplitude \(E\) for the two algorithms, HOT SAX and HST.

| \(E\) | HOT SAX # of distance calls | HST # of distance calls | CPS |
|------|-----------------------------|--------------------------|-----|
| 0.0001 | 24 527 170 | 234 707 | 1 226 | 12 |
| 0.001 | 19 560 251 | 329 397 | 978 | 16 |
| 0.01 | 5 183 885 | 313 363 | 259 | 16 |
| 0.1 | 1 912 774 | 207 881 | 96 | 10 |
| 0.5 | 1 331 203 | 165 142 | 67 | 8 |
| 1.0 | 1 564 755 | 219 777 | 78 | 11 |
| 5.0 | 3 310 974 | 685 889 | 165 | 34 |
| 10.0 | 20 395 837 | 3 105 995 | 1 020 | 155 |

Fig. 4: noise

Fig. 5: The T-speedup and D-speedup of HST as a function of the noise amplitude \(E\) in the case of a synthetic time series. All the time series comprise 20,000 points, \(s = 120\), \(P = 4\), and alphabet=4.

By changing the amplitude \(E\) of the random noise, \(\varepsilon \sim U(0, 1)\), one can obtain a smoother or more ragged time series (Fig. 3, left). We found that HST performs much better than HOT SAX when the noise is very low. When the amplitude of the noise \(E\) is 0.0001, HST is about 104 times faster than HOT SAX (see Fig. 5), in terms of distance calls (and 82 times faster in terms of runtimes). In particular, this is due to the fact that the HST performances are not so much affected by low noise, (Tab. 4), while HOT SAX degrades significantly. In the cases of very low noise/signal ratios, HOT SAX cps exceeds 1200, while it is about 12 for HST. On the other side of the spectrum, when the noise is much higher than the signal, both HOT SAX and HST performances degrade (but HST keeps on being much faster).

We make here an educated guess on the reason why HOT SAX struggles to analyse “apparently easy” time series. When there are many similar sequences and patterns, the matrix profile is likely to have a large number of peaks with very similar heights. For this reason, HOT SAX needs a lot of calculations to disambiguate which of them is the highest one, while HST is quicker due to its better “aiming mechanism”.

HST is still an excellent choice when the noise is as high as the signal itself, while it is less effective when the noise level is much higher than the signal. This is not a surprise: when the noise is 10 times higher than the signal, it dominates the time series. As a result, the time topology (linked to autocorrelation) ceases to be helpful in finding discords. An anomaly search, however, is useful when most of the sequences are expected to be “normal”, and this condition is not satisfied when the noise is 10 times higher than the signal itself.

As far as the authors know these are the first results regarding an exact discord algorithm capable of being more than two orders of magnitude faster than HOT SAX.
4.2.2 Cost per sequence and length of the discords

An important challenge regarding anomaly detection is related to the increasing length of time series. This can be due to the longer data collection times, but also to improved accuracy of the sensors that are able to sample more frequently. The same phenomenon can thus be described with a larger number of points. It becomes desirable to understand to which extent the calculation speed is affected by the length of the discords. Of course, if we increase the length of the sequences we can expect that the total execution time increases. This is a direct consequence of the fact that most of the time is spent calculating distances, and the execution time of a single distance call is approximately proportional to the length of the sequences. On the other hand, there is no obvious reason why the number of distance calls should increase. Actually, the search space shrinks as $s$ increases because of the non self-match condition. According to our model, the cost per sequence is connected with the peaks of the $nnd$ profile. A large peak includes many sequences with high $nnd$ value which are difficult to analyze from HOT SAX. The width of a peak, depending on the non self-match condition, should grow with the length of the sequences, and consequently also the number of good discord candidates. Since the long range time topology procedure helps in leveling the peaks, HST should suffer this problem less than HOT SAX. As a result, one can expect high speedups when searching long discords. We tested this fact with the two longest datasets of Tab. 1 ECG 300 and ECG 318, by increasing the length of the sequences $s$. The values of Tab. 5 confirm that the cost per sequence for HOT SAX is strongly dependent on the length of the sequences. Even in the case of sequences of length 920, the speedup can exceed 50, reaching peaks of more than 100 for longer discords. This is another example where HST can outperform HOT SAX by more than 2 orders of magnitude, moreover this specific condition (longer sequences) is expected to be more relevant in the future.

In summary, by using the $cps$ we gained insights regarding problems that are particularly difficult to treat with HOT SAX, and can greatly benefit from using HST. In those cases, HST can be more than 2 orders of magnitude faster than HOT SAX. In particular, searches related to long discords become more complex. This is due to the fact that one should not look at the time series but rather at the structure of the
search space, i.e. the set of all the sequences, which has an important dependence in respect to the length of the sequences.

4.3 RRA and HST

RRA is freely available on the internet (Senin et al., 2018) (Senin et al., 2015), it is fast and so it represents an ideal comparison test for HST. RRA finds anomalies by exploiting the Kolmogorov complexity of the SAX words. At variance in respect to HOT SAX, the length of the discord is not required as an input parameter but it is obtained as a result of the calculation. The anomalies found with RRA, however, do not coincide with discords. Although there is a good overlap between the first discord of a time series and the anomaly found with RRA, the probability that they coincide is less than 1. Multiple applications of the algorithm in order to find the k-th discord, provide less and less correspondence with the exact solutions. For this reason, at the time of the comparison between HST and RRA we will limit the calculations to the first discord. It is also interesting to notice that the SAX parameters affect only the speed of the algorithm in the case of HST, while they modify also the position of RRA anomalies.

| file      | s, P, alphabet | length | # of distance calls | RRA | HST | D-speedup |
|-----------|----------------|--------|---------------------|-----|-----|-----------|
| Daily commute | 345, 15, 4     | 17,175 | 388,504             | 260,615 | 1.49 |
| Dutch Power  | 750, 6, 3      | 35,040 | 1,801,971           | 259,820 | 6.93 |
| ECG 0606    | 120, 4, 4      | 2,299  | 35,464              | 8,166  | 4.34 |
| ECG 308     | 300, 4, 4      | 5,400  | 101,850             | 25,959  | 3.92 |
| ECG 15      | 300, 4, 4      | 15,000 | 352,331             | 91,970  | 3.83 |
| ECG 108     | 300, 4, 4      | 21,600 | 532,476             | 106,737 | 4.99 |
| ECG 300     | 300, 4, 4      | 536,976 | 199,865,375       | 6,547,211 | 30.52 |
| ECG 318     | 300, 4, 4      | 586,086 | 58,462,005         | 4,426,685 | 13.2  |
| NPRS 43     | 128, 4, 4      | 4,000  | 89,620              | 35,466  | 2.52 |
| NPRS 44     | 128, 4, 4      | 24,125 | 438,957             | 136,658 | 3.21 |
| Video       | 150, 5, 3      | 11,251 | 165,758             | 91,397  | 1.81 |
| Shuttle, TEK 14   | 128, 4, 4  | 5,000  | 326,981             | 65,353  | 5.00 |
| Shuttle, TEK 16   | 128, 4, 4  | 5,000  | 341,405             | 69,912  | 4.88 |
| Shuttle, TEK 17   | 128, 4, 4  | 5,000  | 417,860             | 71,436  | 5.84 |

Table 6: Details of the calculations: “s” is the length of the sequences (also referred to as “window”), “P” is the number of letters in which the sequence is divided, while the total number of available letters is the parameter “alphabet”. “Length” refers to the number of points in the time series under observation. The values of s, P, and alphabet follow those of (Senin et al., 2015). The minor differences are due to the fact that, for our algorithm, P must divide exactly s (while this is not the case for the algorithm implemented in Grammarviz 3.0). Columns 3 and 4 show the average number of distance calls needed by RRA and HST for finding the first discord. The number of distance calls has been averaged over 10 runs for each dataset and algorithm. The distance-speedup (or D-speedup) has been calculated as the ratio between the number of distance calls of RRA over those of HST. All these datasets can be downloaded from GitHub (Senin, 2019).

The results reported for RRA (Senin et al., 2015) (Senin, 2019) are substantially different in respect to the numerical experiments we obtained with their code (Tab. 6). This might be due to the choice of the --strategy parameter (the available choices...
are NONE, EXACT, and MINDIST). This parameter is used to further reduce the search space and it can improve significantly the speed of the algorithm. For some datasets, one can obtain essentially the same anomaly with different flags. In many cases, however, the precision loss is not acceptable. As an example, the position of the exact discord of length 128 for the time series TEK 14 is 3852. The position of the RRA anomaly with --strategy EXACT is 4717, with --strategy MINDIST is 4320 (this time series contains 5000 points, as explained in Tab. 6). Since a user cannot know in advance if one strategy provides enough accuracy, a valid comparison should be limited to the execution flag --strategy NONE. Other strategies can provide faster executions but one needs to check afterward if the result is comparable with the discord (thus nullifying the improved speed). Following this choice, the advantage of

Listing 3: RRA script for TEK 14

```java
java -cp "target/grammarviz2-0.0.1-SNAPSHOT-jar-with-dependencies.jar" net.senin.grammarviz.GrammarVizAnomaly -alg RRA -n 1 -i TEK14.txt -w 128 -p 4 -a 4 -g Sequitur --strategy none
```

RRA in respect to HOT SAX becomes more limited. According to our numerical experiments (Tab. 6 and Tab. 1) in some cases, RRA is even slower than our version of HOT SAX. Notice however that it is always faster than the version of HOT SAX released by one of the authors of RRA, available in the same suite Grammarviz 3.0 (Senin, 2019) and not reported here. We utilize the last version of RRA present in the Grammarviz 3.0 suite on GitHub dating the 27th of October 2016. For completeness, Listing 3 contains the script used to produce RRA results for the time series TEK 14. When possible we keep the same SAX parametrization used for the validation of RRA. They correspond to optimal solutions where the lengths of the sequences are close to the values returned automatically by RRA. Our code, however, requires that the number of parts of the PAA is an exact divisor of the length of the sequences. If the parameters used for the validation of RRA do not comply with this condition we use values of $P$ which are as close as possible.

The results of Tab. 6 show that HST outperforms RRA. In the case of ECG 300, HST is about 30 times faster than RRA, while in the worst-case scenario, for the Daily Communications time series it is only 49% faster. These improvements are calculated as the ratio of the number of distance calls of RRA and HST, and denoted as D-speedup. Calculating the ratio of the run times for the two algorithms (T-speedup), does not provide useful information since RRA has been developed in a substantially slower programming language: JAVA (while HST has been developed in Fortran).

4.4 Disk-Aware Discord Discovery

DADD (or DRAG) is known to be a very fast algorithm for discord search aimed at those cases where the whole time series cannot reside on the RAM and requires to be stored on a disk. For completeness, we report here some running time comparisons
Table 7: The second and third columns show the calculation times (seconds) for 10 discords for DADD and HST. The fourth column displays the time-speedup calculated as the ratio of the DADD and HST running times. The page to be analysed has been built by taking the first 10,000 sequences of length 512 of each dataset of column 1. DADD calculations depend on the $r$ parameter, the timings of column 2 refer to an $r$ value equivalent to 99% of the $nnd$ of the 10th discord, while the timings of column 5 are associated with the exact value (100%).
r-parameters are usually too big and require manual adjustments. As a solution, we performed a full calculation on the whole page and we obtained the exact $nnd$ value of the 10th discord. We performed two kinds of calculations, in one case using the exact value and in the second 99% of it. The reason is that the r-parameter affects the computational times for DADD, in general the more distant it is from the exact $nnd$ of the k-th discord, the slower the code. In particular, even a small modification from the exact $r$ to 0.99$r$ can have a non-negligible impact on the calculation times. For example, on one dataset (ECG 15) a 1% difference of $r$ leads to a 77% increase of the runtime. Although obtaining this quantity requires some time we do not include it in the total computational time for the experiments we performed, focusing on the algorithm only.

We first verified the correspondence of the results (we turned off z-normalization in the HST code and we forced the distance calls between overlapping sequences to be allowed). With these modifications, the two codes produced identical results and we could compare the runtimes. In all the tests, HST is 12-25 times faster than DADD (see Tab. 7).

4.5 SCAMP

![Fig. 6: (Left) Calculations performed on sections of the ECG 300 dataset. HST has been run using the same SAX parameters presented in Tab. 2, for each slice of the time series five calculations were performed, for: 1, 10, 40, 70, and 100 discords. The runtimes on the same datasets for SCAMP include only the calculation of the matrix profile. (Right) HST runtimes as a function of the number of discords searched for different slices of the ECG 300 time series.](image)

SCAMP is the fastest algorithm of the Matrix Profile series. This algorithm has a broad purpose and it not specialized for discord search, however one of its peculiarities is that it is insensitive in respect to the length of the sequences. Another interesting feature is that looking for many discords is computationally inexpensive once the matrix profile is known. It is difficult to compare an exact algorithm (SCAMP) with a heuristic one (HST), however we provide here some running conditions which can help in deciding when to use one or the other. For the tests of this section, we use a C++ code released by the authors of the algorithm, available on GitHub [SCAMP, 2020]. This implementation is capable of exploiting multiple cores.
and graphic cards, however for a fair comparison with HST we report here the values obtained for a single core (in this case it is essentially identical to another matrix profile algorithm: STOMP (Zhu et al., 2016). SCAMP running times scale quadratically with the length of the time series, but do not depend on the underlying data. At variance, the running times for HST are affected by the characteristics of the specific dataset. For an indicative comparison we consider parts of the time series ECG 300. The calculations of Fig. 6 refer to the dataset truncated at different lengths (we used the same parameters of Tab. 1). The slices contain $10^5$, $2 \cdot 10^5$, $3 \cdot 10^5$, $4 \cdot 10^5$, and $536976$ points (the full time series); for each of these slices we check the running times for HST for 1, 10, 40, 70 and 100 discords. The timings for SCAMP refer only to the calculation of the matrix profile we do not include the time spent for the search of the discords. For all of the above tests, HST is (much) faster than SCAMP (Fig. 6). Moreover since HST scales linearly with the size of the time series, its advantage over SCAMP grows for the longer runs.

For very short time series, SCAMP can become faster than HST. We performed experiments with ECG 0606 (2299 points). In this case, the advantage of SCAMP over HST is so small that the total execution time is dominated by writing the results on the disc. For a slightly longer time series (TEK 14, 5000 points) the two algorithms show comparable times. We also notice that short time series can have only a few discords, because of the non-overlap condition.

SCAMP can become faster than HST if one is interested in finding hundreds of (long) discords. However, only longer time series can contain so many discords, so the advantage due to the insensitivity to the number of discords is hindered by the quadratic nature of SCAMP. Moreover, one should take into account that only a few of the discords are expected to be “real” anomalies. Some time series do not contain anomalies, while all of them contain many discords: $O(N/s)$. The reason is simple: discords are just maxima of the matrix profile. Some of these maxima might be abnormally higher than the rest of the matrix profile, thus being anomalies. Others might be just random fluctuations. One is most likely interested only in those discords which are also outliers: significant discords (Avogadro et al., 2020). For example, ECG 300 has only 5 significant discords of length $s = 300$. In this case, it would be useless to calculate the position of the first 100 discords.

It should be noted that there is an approximate algorithm of the Matrix Profile series called preSCRIMP (Zhu et al., 2018) which produces an approximate version of the matrix profile and which can be used for approximate discords search. Since its results depend on the quality of the approximation and it displays a quadratic complexity with the number of sequences and it is not a direct competitor of HST which is exact.

4.6 More than a hundred million points time series

As an example, we performed a calculation on a time series consisting of 170,326,411 points associated with the research on insect vector feeding by Willett et al. (2016). The total computational time for the first 10 discords on an Intel(R) Xeon(R) CPU
E5-2640 v3 @ 2.60GHz has been: 96288.93 s (less than 1 day and 3 hours). The parameters in use were $s = 512$, $P = 128$, $\text{alphabet} = 4$.

This timing is comparable with the fastest exact Matrix Profile parallel implementations exploiting graphic cards \cite{Zhu2018,Zimmerman2019}, while the serial implementations would be hundreds of times slower. One can also compare the performance of HST and HOT SAX. In this case, we limit the calculations to $k = 1$. We obtain a D-speedup = 21, while the T-speedup is 16. For HST the $cps$ is 79, while for HOT SAX it is 1547.

4.7 HST Scaling

In this section, we provide indications regarding how HST scales as a function of the parameters, in many “normal” conditions. The datasets of Tab. 6 include biological data (the ECG cardiograms and NPRS respiration time series), data regarding human activities (Daily commute, Dutch power and Video), and sensor data (the Shuttle TEK series). They also span a couple of orders of magnitude in terms of length (ranging from 2299 to 586086 points). This diversity can be used to gain insights regarding the costs of the calculations as a function of their parameters on “typical” time series. In the beginning, we checked the speed as a function of the quantity of discords to be found. Since the running times span a couple of orders of magnitude we normalized the results of each dataset with the running time for the first discord. In Fig. 7 (left) it is possible to notice that in all the cases the curves associated with each dataset are almost linear.

The dependence of the running times as a function of the length of the sequences is displayed in Fig. 7 (right), for this graph we normalize the values according to the calculations for discords of length 200. Also in this case the scaling is limited and essentially proportional to the length of the sequences $s$.

The last factor determining the runtimes is the length of the time series. The experiments on ECG 300 show (Fig. 6, left) that HST runtimes are approximately proportional also to this quantity. In summary, HST scales approximately linearly with:

- the number of discords searched ($k$)
- the length of the sequences ($s$)
- the length of the time series ($N$)

These scaling properties suggest a rule of thumb that can be used when facing a very long time series. It is enough to first perform a check on a short extract (e.g. $10^6$ points) and then extrapolate the total running times multiplying the $cps$, the length of the time series and the number of discords searched, to obtain a rough estimate of the total number of calls. Although this recipe is very dependent on the premise that the mechanism producing the time series does not change significantly with time (a non-trivial condition), the results of Fig. 6 (left) seem to indicate that it can be useful.
4.8 Summary of the results

The tests that we performed show that HST can be an important resource at the time of finding discords when the task is long and complex:

- it is 2-100+ times faster than HOT SAX, it is particularly faster when looking for longer discords or for “simple looking” time series.
- it is 50%-30 times faster than RRA (and HST returns exact discords)
- it is 12-25 times faster than DADD (DRAG)
- HST, in practical cases, is much faster than SCAMP and it can produce results comparable to GPU accelerated Matrix Profile algorithms on longer time series.

5 Conclusions and future works

In the present paper, we detail a new algorithm for exact discord search in time series, called HOT SAX Time. HST can be obtained from HOT SAX with rather easy modifications. In HST the external and inner loops are re-arranged following good quality approximate ndds. During the execution of the algorithm, the order of the sequences of the external loop is rearranged every time that a good discord candidate is found: sequences with a high approximate ndds are positioned at the beginning.

The performance of HST has been validated with real-life and synthetic time series and compared with the results of HOT SAX, RRA, DADD. HST has shown to be faster than these algorithms for all of the datasets under consideration. The only serial competitor for HST is SCAMP (running on a single core) in very special cases. The quadratic nature of the matrix profile codes hinders their performance on long time series. For more common cases, HST is many times faster than SCAMP. For long time series (>100 million points) the present serial HST code is essentially as fast as the parallel implementation of SCAMP exploiting graphic cards.

Since HOT SAX can be considered as a benchmark algorithm we used it to understand under which condition a discord search becomes complex. For this purpose,
we defined a complexity indicator, the cost per sequence, which allows one to compare searches on time series of different lengths. We singled out two main parameters that render discord searches particularly complex for HOT SAX. The first one is a counterintuitive property: low noise/signal ratios. The second parameter with a strong influence on the cost per sequence is the length of the discords. For searches involving long sequences, HST tends to be much faster than HOT SAX; in light of the fact that the technological improvements lead to sensors capable of producing higher sampling frequencies, the advantage provided by HST should grow with time. In particular, to the best knowledge of the authors, HST is the first exact discord algorithm that can be more than 100 times faster than HOT SAX.

Future developments of this research include more extensive tests in order to better define the characteristics of time series. It could also be interesting to use alternatives to SAX in order to improve the warm-up phase. Parallelizing HST is also a natural follow up of the present work.

Acknowledgments

PA would like to thank Prof. M. Zymbler for suggestions regarding DADD, and Prof. E. Keogh for providing the dataset used for Sec. 4.6 and for the useful information regarding the Matrix Profile.

References

Avogadro P, Dominoni MA (2019) Topological approach for finding nearest neighbor sequence in time series. In: IC3K 2019 11th International Joint Conference on Knowledge Discovery, Knowledge engineering and Knowledge Management, 17-19 September 2019, SCITEPRESS, vol 1: KDIR, pp 233–244

Avogadro P, Palonca L, Dominoni MA (2020) Online anomaly search in time series: significant online discords. Knowledge and Information Systems DOI 10.1007/s10115-020-01453-4, URL https://doi.org/10.1007/s10115-020-01453-4

Benchmarksgame-team (2020) The computer language benchmarks game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/

Bu Y, Leung TW, Fu AWC, Keogh E, Pei J, Meshkin S (2007) Wat: Finding top-k discords in time series database. In: Proceedings of the 2007 SIAM International Conference on Data Mining

Buu HTQ, Anh DT (2011) Time series discord discovery based on isax symbolic representation. In: 2011 Third International Conference on Knowledge and Systems Engineering, pp 11–18, DOI 10.1109/KSE.2011.11

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: A survey. ACM Comput Surv 41(3):15:1–15:58, DOI 10.1145/1541880.1541882, URL http://doi.acm.org/10.1145/1541880.1541882

Gao Y, Lin J (2018) Exploring variable-length time series motifs in one hundred million length scale. Data Mining and Knowledge Dis-
Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE (2000) PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation 101(23):e215–e220, circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215

Gupta M, Gao J, Aggarwal C, Han J (2014) Outlier detection for temporal data: A survey. IEEE Transactions on Knowledge and Data Engineering 26(9):2250–2267, DOI 10.1109/TKDE.2013.184

Keogh E, Kasetty S (2003) On the need for time series data mining benchmarks: A survey and empirical demonstration. Data Mining and Knowledge Discovery 7(4):349–371, DOI 10.1023/A:1024988512476, URL https://doi.org/10.1023/A:1024988512476

Keogh E, Lin J, Fu A (2005) Hot sax: efficiently finding the most unusual time series subsequence. In: Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM’05), pp 226–233

Lin J, Keogh E, Lonardi S, Chiu B (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, ACM, New York, NY, USA, DMKD ’03, pp 2–11, DOI 10.1145/882082.882086, URL http://doi.acm.org/10.1145/882082.882086

Moody GB, Mark RG (2001) The impact of the mit-bih arrhythmia database. IEEE Engineering in Medicine and Biology Magazine 20(3):45–50, DOI 10.1109/51.932724

Nakamura T, Imamura M, Mercer R, Keogh EJ (2020) Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In: ICDM 2020

SCAMP (2020) Matrix profile on github. https://github.com/zpzim/SCAMP

Senin P (2019) Grammarviz 3.0. https://github.com/GrammarViz2/grammarviz2_src/

Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo A, Chen C, Frankenstein S, Lerner M (2014) Grammarviz 2.0: A tool for grammar-based pattern discovery in time series. In: Calders T, Esposito F, Hüllermeier E, Meo R (eds) Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 468–472

Senin P, Lin J, Wang X, T O, Gandhi S, Boedihardjo A, Chen C, Frankenstein S, Lerner M (2015) Time series anomaly discovery with grammar-based compression. In: The International Conference on Extending Database Technology, EDBT 15., pp 276–281

Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, Chen C, Frankenstein S (2018) Grammarviz 3.0: Interactive discovery of variable-length time series patterns. ACM Trans Knowl Discov Data 12(1):10:1–10:28, DOI 10.1145/3051126, URL http://doi.acm.org/10.1145/3051126

Shieh J, Keogh E (2008) isax: Indexing and mining terabyte sized time series. In: Proceedings of the 14th ACM SIGKDD International Con-
A fast algorithm for complex discord searches in time series: HOT SAX Time

Thuy HTT, Anh DT, Chau VTN (2018) A novel method for time series anomaly detection based on segmentation and clustering. In: 2018 10th International Conference on Knowledge and Systems Engineering (KSE), pp 276–281, DOI 10.1109/KSE.2018.8573409

Willett DS, George J, Willett NS, Stelinski LL, Lapointe SL (2016) Machine learning for characterization of insect vector feeding. PLOS Computational Biology 12(11):1–14, DOI 10.1371/journal.pcbi.1005158, URL https://doi.org/10.1371/journal.pcbi.1005158

Yankov D, Keogh E, Rebbapragada U (2008) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowl Inf Syst 17 pp 241–262, DOI doi:10.1007/s10115-008-0131-9, URL http://alumni.cs.ucr.edu/~dyankov/projects/index.html

Yeh CM, Zhu Y, Ulanova L, Begum N, Ding Y, Dau HA, Silva DF, Mueen A, Keogh E (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, DOI 10.1109/ICDM.2016.0179

Zhu Y, Zimmerman Z, Senobari NS, Yeh CM, Funning G, Mueen A, Brisk P, Keogh E (2016) Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp 739–748, DOI 10.1109/ICDM.2016.0085

Zhu Y, Yeh CM, Zimmerman Z, Kamgar K, Keogh E (2018) Matrix profile xi: Scrimp++: Time series motif discovery at interactive speeds. In: 2018 IEEE International Conference on Data Mining (ICDM), pp 837–846, DOI 10.1109/ICDM.2018.00099

Zhu Y, Zimmerman Z, Shakibay Senobari N, Yeh CCM, Funning G, Mueen A, Brisk P, Keogh E (2018) Exploiting a novel algorithm and gpus to break the ten quadrillion pairwise comparisons barrier for time series motifs and joins. Knowl Inf Syst 54(1):203–236, DOI 10.1007/s10115-017-1138-x, URL https://doi.org/10.1007/s10115-017-1138-x

Zimmerman Z, Kamgar K, Senobari NS, Crites B, Funning G, Brisk P, Keogh E (2019) Matrix profile xiv: Scaling time series motif discovery with gpus to break a quintillion pairwise comparisons a day and beyond. In: Proceedings of the ACM Symposium on Cloud Computing, Association for Computing Machinery, New York, NY, USA, SoCC ’19, p 74–86, DOI 10.1145/3357223.3362721, URL https://doi.org/10.1145/3357223.3362721

Zymbler M, Polyakov A, Kipnis M (2019) Time series discord discovery on intel many-core systems. In: Sokolinsky L, Zymbler M (eds) Parallel Computational Technologies, Springer International Publishing, Cham, pp 168–182