Applied Intelligence (2022) 52:10060–10081
https://doi.org/10.1007/s10489-021-02897-z

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

Paolo Avogadro[1] · Matteo Alessandro Dominoni[1]

Accepted: 5 October 2021 / Published online: 11 January 2022

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 (Heuristically Ordered Time series using Symbolic Aggregate ApproXimation) by exploiting two main ideas: the warm-up process, and the similarity between sequences close in time. These improvements can reduce the size of the discord search space by orders of magnitude when compared with HOT SAX. The resulting algorithm, called HOT SAX Time (HST), has been validated with real and synthetic time series, and successfully compared with HOT SAX, RRA (Rare Rule Anomaly), SCAMP (SCAlable Matrix Profile), and DADD (Disk Aware Discord Discovery). 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. This is the first exact discord search algorithm that has been demonstrated being more than 100 times faster than HOT SAX.

Keywords Time series · Anomaly · Discord · Nearest neighbor distance

1 Introduction and related works

Anomaly discovery in time series is an active research field [6, 10] 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) [15] is a successful algorithm which goes in this direction associating each sequence to a symbolic sequence. Later on, an improved version, called iSAX (indexable SAX), was presented allowing for better clusterizations [23].

Discords were introduced in 2005 [13], where it was also suggested a fast algorithm for finding them: HOT SAX. The use of Haar wavelets [4] improved the pruning power in respect to SAX. HOTiSAX [5] is an algorithm based on iSAX which is up to four times faster than HOT SAX. Similar results were obtained with HS-Squeezer [7]. At variance, [14] makes use of iSAX and combines it with WAT (Wavelet and Augmented Trie) algorithm for discord searches. Recent research works improving of SAX include [25] and [27].

The most advanced methods for a complete characterization of the time series are those of the Matrix Profile series initiated in 2016 [31]. 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 [8, 35].

If the size of the time series is such that the RAM (Random Access Memory) is not enough, one needs to keep at least part of the data on disk, and for this purpose it was developed the DADD [30] (this algorithm is also known as DRAG, or Discord Range Aware Gathering), in 2020 it was proposed a variant of DADD that can be 2-3 times

✉ Paolo Avogadro
paolo.avogadro@unimib.it

Matteo Alessandro Dominoni
matteo.dominoni@unimib.it

[1] Università degli studi di Milano-Bicocca, viale Sarca 336, 22126, Milano, Italy

faster [24]. Discord analysis for multivariate time series has been studied by [11] and [29]. MERLIN is a new algorithm based on DADD which can quickly scan all the discords within a given length range [17].

A first route to speed up the calculation regards parallelizing the existing algorithms, for example in [37] it has been developed a version of HOT SAX for Intel Many-Core Systems. More recently, GPU approaches have shown to be very competitive [32] for motif and discord mining. The fastest matrix profile algorithm involving graphic card accelerators is SCAMP [36].

Another path to speed up the calculations regards using approximate algorithms. Rare Rule Anomaly [20], 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 [26], 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 times 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 Section 4.2). The *cps* allows one to compare discord searches on time series of different length and rate them in terms of complexity. With the help of this indicator we found a set of discord searches where HST is particularly better than other algorithms. There are two important novelties that make HST faster. At the beginning it calculates an approximate nearest neighbor distance for all the sequences, and bases the outer loop order on these quantities. The second important idea is to exploit the similar properties of time-close sequences. In particular, if one sequence is ruled out of the discord search space, usually one can remove many of its time-neighbors with a very limited computational complexity (e.g. one call to the distance function). This dynamic process continuously reduces the size of the search space and pushes likely discords at the beginning of the search. Although this search space reduction is very cheap (in terms of calls to the distance function), the results obtained are exact.

The rest of the paper is organized as follows:

- After clarifying the notations and defining useful quantities, Section 2 provides a brief explanation of the brute force approach and of HOT SAX.
- Section 3 details HST and the rationale of the modifications in respect to HOT SAX.
- Section 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.
- Section 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}, ..., p_{k+s-1}. In this paper, unless specifically stated, we consider z-normalized sequences [12].
- If the number of points of a time series is denoted with N_{tot}, the number of sequences of length s therein is smaller, and denoted with the letter $N = N_{tot} - s + 1$. 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:

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

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:

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

as pointed out by [35], the same result can be obtained with the help of the scalar product:

$$d(k,l) = \sqrt{2s\left(1 - \frac{k \cdot l - s\mu_k\mu_l}{s\sigma_k\sigma_l}\right)}, \quad (3)$$

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

– The nearest neighbor distance (nnd) for sequence k is obtained as:

$$nnd(k) = \min_{j:|k-j|\geq s} d(k,j). \quad (4)$$

The search space for the minimization includes all the sequences, j, of the times 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 nnd due to overlaps.

– The quantity $ngh(i)$ returns the position of the nearest neighbor of sequence i:

$$nnd(i) = d\,(i, ngh(i))\,.$$

– The nnd profile is the set of the current nnds of all the sequences of a time series. In practice many of the nnds returned by HST are only approximate values. Had these values been exact, the nnd profile would be identical to a special case of *Matrix Profile*: the self-similarity join profile [31].
– In order to highlight that some quantities are stored in an array or variable of the code, we use typewriter characters. For example `nnd[i]`, or `ngh[i]` are the arrays containing the information regarding the $nnd(i)$ and $ngh(i)$.
– 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.

$$discord = \arg\max_i \Big(nnd(i)\Big)$$

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* is the true 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 is a clever heuristic modification of the brute force approach that skips a large part of the distance calculations, while HST adds further improvements that can lead to speedups of more than two orders of magnitude in respect to HOT SAX. Via the symbolic aggregate approximation (SAX) [15] it is possible to quickly assign every sequence to a much shorter symbolic sequence (or SAX cluster). As a quick reminder, SAX divides each sequence into P subsequences, and for each of these subsequences calculates the average value. The range of possible average values is divided into a number of segments equal to the "alphabet" (the quantity of available symbols). Each sub-sequence is then converted into the symbol corresponding to its average value. As a result, SAX turns a s point sequence into a P symbol sequence (P$<<$ s). This dimensionality reduction procedure clusterizes sequences quickly. Because of the properties of SAX, sequences belonging to the same SAX cluster can also be Euclidean neighbors. HOT SAX exploits the relation between these two notions of closeness (topology) to quickly reduce the discord search space. In the following we will see that HST adds another topology to this picture, the one induced by time, to further speed up discord searches. In HOT SAX the outer and inner loops are re-arranged following the indications obtained with SAX. The external loop is now structured according to the size 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. At the beginning, the algorithm calculates the distance between the sequence defined by the external loop and all of its SAX neighbors (the other sequences belonging to the same SAX cluster). 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 that a discord is 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 providing a better approximate *nnd* for the sequence under consideration. 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 close 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 *nnd*s 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 *nnd*s. 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 [4] 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 *nnd*s in the inner loop. The memory cost of keeping track of the approximate *nnd*s is just the size of the time series O(N). Although these approximate *nnd*s are rough estimates of the exact ones, they are very useful, since the real anomalies (discords) have *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 *nnd*. If the sequence under consideration exhibits an *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 to apply the aforementioned technique also in the case of the first discord.

3.3 Warm-up

In this section we explain the first main novelty 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 *nnd* value for all the sequences. We now detail a fast procedure to build such an approximate *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 *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 can also be 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 *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 *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 *nnd*. This procedure resembles closely the original HOT SAX, since we are calculating distances among sequences belonging to the same cluster. A real advantage can be obtained only if the approximate *nnd*s 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 *nnd*s obtained with the warm up procedure.

Time series display a form of time correlation first described as Consecutive Neighborhood Preserving (CNP) Property [34]. This property can be regarded as a form of time topology which makes possible to improve an approximate *nnd* profile of a time series at a low cost [1]. 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 *nnd*s which is particularly noticeable regarding the *ngh* profile where, often times, the following formula is true:

$$ngh(i+1) = ngh(i) + 1. \tag{5}$$

This property can be used to improve the low quality *nnd*s obtained with the warm-up, where (5) is almost never verified. In fact, it is enough to check the following distance: $d\left(i+1,\ ngh(i)+1\right)$ for all of the sequences in order to obtain a better *nnd* 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: $d\left(i-1,\ ngh(i)-1\right)$. 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 *nnd* profile is much better than the one resulting from the warm-up procedure. We can now use the *nnd*s to reorder the loops for the discord search. This usage of the properties of time-close sequences adds a new kind of topology (notion of closeness) to the original HOT SAX idea. The new interplay between the SAX, Euclidean and time topologies allows the code to quickly collect important information regarding the position of the discord.

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

The novel approach of HST is to use a better parameter to order the external loop. Instead of using sequences

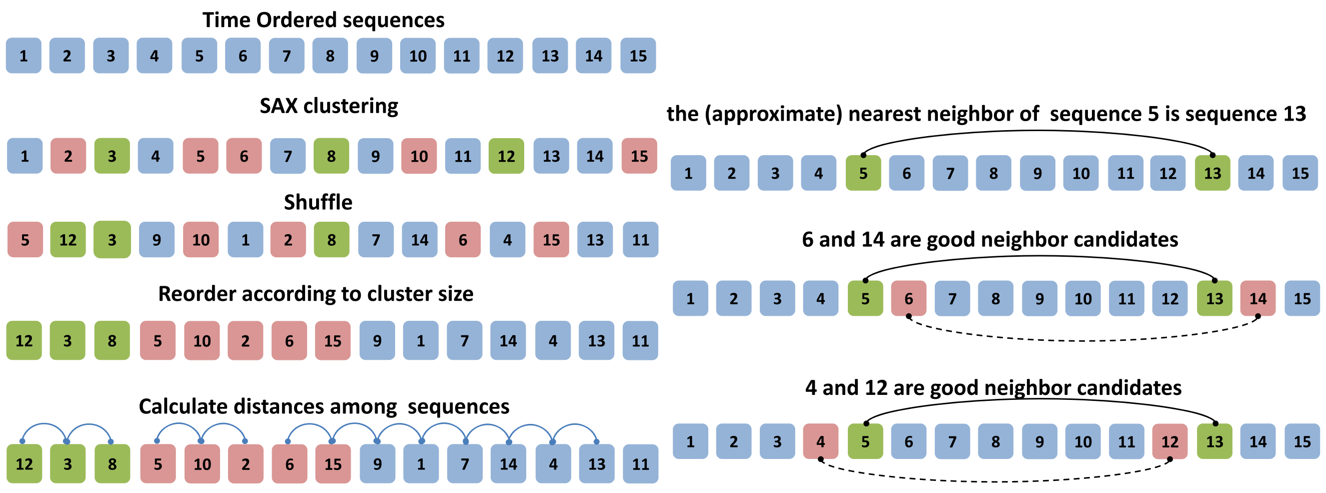


Fig. 1 (Left) At the beginning, all the sequences are simply ordered according to their first point. We suppose that the length of the sequences is 5, we show here only the beginning point of each sequence. As a first step, the SAX procedure assigns each sequence to a cluster (in the example there are 3 clusters, defined by color: green, blue, and pink). The sequences are then shuffled in order to avoid that many close sequences are one after the other. The sequences are then grouped according to the cluster size: at the beginning the green cluster containing 3 sequences, then the pink one and the blue one. The final step consists in calculating the distance between adjacent sequences (according to this new order). Because of the self-match condition the following distances cannot be calculated: $d(8, 5)$, $d(2, 6)$, and $d(13, 11)$. In the case of adjacent clusters, the distance is calculated between the last sequence of one cluster and the first one the next cluster. For example, the last sequence of the pink cluster, 15, is connected with the first one of the blue cluster, 9. Notice that for sequence 11 it is not possible to calculate any distance, and for this reason, a default *nnd* high value is assigned. (Right). The time topology suggests where to find good neighbor candidates. If we know that sequence 5 and 13 are neighbors, then 6 and 14 are likely good neighbors (also 4 and 12)

belonging to small clusters we determine the order of the sequences of the external loop with their approximate *nnd*. The combination of warm-up plus time topology returns an approximate *nnd* for all the sequences that contains part of the information of the exact matrix profile. One is sure to make a reasonable guess by putting at the beginning of the external loop those sequences with a high *nnd* value. This choice has two important advantages in respect to picking small SAX clusters:

– The external loop order becomes a function of the properties of the sequences, not of the containing clusters. The order defined by the SAX clusters involves a lot of uncertainty since all the sequences in the cluster are on equal footing. For example, if the cluster contains hundreds of sequences one might have to wait a lot before processing the sequence with the highest *nnd*.
– With this new approach the external loop order is no longer fixed but it can be improved dynamically as the algorithm proceeds its execution.

At the beginning, the *nnd* profile is still rough, so it is better to smear it with a moving average. It has been shown, in fact, that the exact *nnd* profile (matrix profile) displays a "sort of smoothness" [1, 34]. A rough *nnd* 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 *nnd*s spans approximately the length of a sequence [1], for this reason the moving average takes into account $s + 1$ sequences:

$$\overline{nnd}(i) = \frac{1}{s+1} \sum_{j=-s/2}^{s/2} nnd(i + j) \tag{6}$$

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 *nnd*s. In summary, at the beginning of the external loop HST puts the sequences with the highest $\overline{nnd}$s, 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 *nnd*s, in fact, become smaller and smaller converging to the exact ones during the execution of the algorithm. Since the knowledge regarding sequences is refined during the execution of the code, it becomes reasonable to re-arrange the remaining parts of external loop in order to prompt high *nnd* 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 *nnds* 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

Here we introduce a new usage of the CNP property that allows the algorithm to greatly reduce the size of the search space.

Let's consider what happens when a good discord candidate is found during the search process. This sequence has the highest *nnd* so far. In general this means that it belongs to a peak composed of many sequences with a high *nnd* value due to the "pseudo-smoothness" of the matrix profile. There are three observations regarding this peak:

1. Each of the sequences of the peak might require a large number of distance calls of the inner loop to be ruled out from the discord candidates.
2. The number of sequences with high *nnd* (the width of the peak) is not easily determined.
3. Distance calls involving a sequence of the peak do not contribute to the *nnd*s of its time-neighbors since they are self-matches. For this reason one needs to perform independent calculations for all the sequences of the peak.

The width of the peak (the number of sequences that belong to it), is likely proportional to the length of the sequences s. This is a result of the non-self-match condition, since overlapping sequences do not contribute to each others *nnd*s. The time topology is very useful since it tells us that the Euclidean neighbors of: $i+1, i-1, i+2, i-2, ...$ are likely time-close. This means that in many cases the following conditions are true:

- $ngh(i+1) = ngh(i)+1$
- $ngh(i-1) = ngh(i)-1$
- $ngh(i+2) = ngh(i)+2$
- $\dots$

With these suggestions, it is possible to make a very limited number of calls to the distance function and likely obtain approximate *nnd*s close to the exact one, thus allowing one to disambiguate which of them is the top value. In practice, the *long_range_time_topology* functions remove peaks as shown in Fig. 2. The computational cost for this result is very limited (few distance calls per sequence) if compared with the normal HOT SAX procedure that might require $O(sN)$ calculations (each of the $\approx s$ sequences of the peak might require $\approx N$ distance calls). Notice that within the

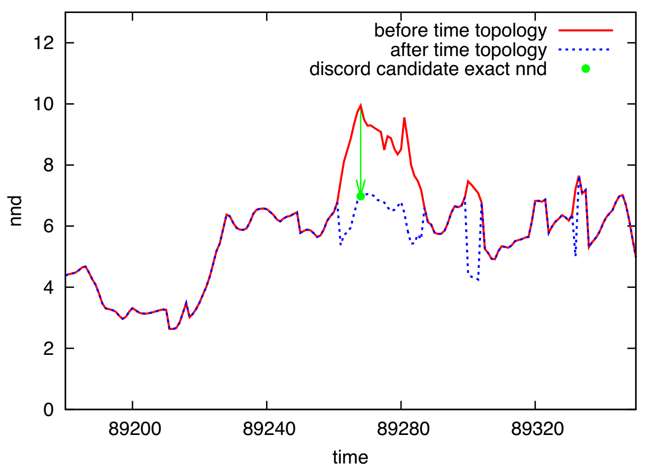


Fig. 2 Sequence 89268 has the highest approximate *nnd* value (9.94) of a time series, and it is surrounded by a peak about 30 sequences wide. It is a good discord candidate so its distance with all the other sequences (but self-matches) has been calculated ($\approx 2 \cdot 10^5$ calls). As a result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

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] < 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 `ngh[i+j]`.

It can happen that: `nnd[i+j] < d( i+j, 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:

- *Long_range_time_topology_forw* (see Listing 1)
- *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 [15].

Listing 1
Long_range_time_topology_forw

```
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
```

2. The *Warm-up*() procedure obtains approximate *nnd*s for most of the sequences, as explained in Section 3.3. The *nnd*s are stored in the homonym integer array `nnd[]`, while the neighbors (of each sequence) are stored in the integer array `ngh[]`.
3. The *Short_range_time_topology*() function improves the quality of the *nnd*s (Section 3.4).
4. The *Sort_External*(`order`) function creates an integer array containing the order of the sequences of the external loop, from the highest $\overline{nnd}$ 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

Listing 2 HST Pseudocode

```
 1  nnd = 99999999.9            // initialize array with all the nnds
 2
 3  SAX()
 4  Warm-up()
 5  Short_range_time_topology()
 6  Sort_External(order)        // fill the array with the new ordering
 7
 8  bestDist= 0.0               // initialize the current max nnd-value
 9
10  for  j=1; j<= N; j++ do     // the external loop
11
12    i= order[j]               // "i" is the sequence processed now
13    can_be_discord = true     // sequence "i" can be the discord
14
15    Avoid_low_nnds (i, bestDist) // skip sequences with low nnd
16
17    if (can_be_discord == true) then //is "i" still a canidate?
18      Current_cluster(i, bestDist)//minimization (HOT SAX inner loop)
19    endif
20
21    if (can_be_discord == true) then //is "i" still a canidate?
22      Other_clusters(i, bestDist)//minimization (HOT SAX inner loop)
23    end if
24
25    Long_range_time_topology_forw (i, bestDist) // level peaks
26    Long_range_time_topology_back (i, bestDist) // level peaks
27
28    if (can_be_discord == true) then // i is a good discord candidate
29      Update(bestDist)             // update the current max nnd-value
30      Sort_Remaining_Ext(order)    // re-sort the external loop
31    end if
32  end for   //  external loop
```

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_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`.

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, and `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 *nnd*s of the time neighbors of the sequence i (sequences beginning at $i+1$, $i-1$, $i+2$ ...) as explained in Section 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 the function *Update*().
- The condition `can_be_discord = true` after the minimization procedure implies that almost 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 Section 3.5.2. The remaining sequences of the external loop are re-arranged according to the values of their updated *nnd*s: from those with the highest *nnd*s 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 [13] (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 [35] (which is faster than the explicit distance among z-normalized sequences), this condition might no longer be true. 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 Section 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 ($\approx 1s$), the T-speedup is not particularly a good indicator since it is affected by other functions that require tenths of 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*, [3]. 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 [21, 22] 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 used them also in the present article. Some of these time series are extracts of longer datasets, for example those of the Physionet repository [9, 16]. In order to provide a uniform comparison with RRA we stick with the ones already used for its validation.

HOT SAX, RRA, and HST use an underlying SAX clustering. If we use the same SAX parameters among different algorithms we are sure to provide a fair comparison of the results. In the present case we choose to use a set of SAX parameters that is very close (see Section 4.3 for details) to the ones used to validate RRA for each dataset, to compare our results with theirs. These parameters are listed in Table 1 as s (length of the sequences), P (number of segments for the each SAX cluster), and the alphabet (amount of available symbols). Unless otherwise specified we will keep those parameters for the rest of the calculations.

We added a real dataset containing more than 10^8 points [28]. 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.

4.1 HOT SAX and HST

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 (Table 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, The time added by these functions would be negligible in the case of more complex searches but have an important impact for very short calculations. We keep these tests since they have the same structure as those used for the evaluation of RRA [21]. The mean D-speedup obtained when searching 1 discord is 6.26 while the maximum is 13.65.

Since HST has been built to have good performances on complex searches, we increase the difficulty of the calculations of Table 1 by expanding the search on 10 discords in Table 2. 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 $(N/s) + 1$

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

			# of distance calls			Runtimes [s]
File	s, P, alphabet	Length	HOT SAX	HST	D-speedup	HST
Daily commute	345, 15, 4	17 175	819 802	260 615	3.14	0.18
Dutch Power	750, 6, 3	35 040	3 428 728	259 820	13.19	0.32
ECG 0606	120, 4, 4	2 299	20 621	8 166	2.52	0.017
ECG 308	300, 4, 4	5 400	149 329	25 959	5.75	0.039
ECG 15	300, 4, 4	15 000	215 928	91 970	2.35	0.088
ECG 108	300, 4, 4	21 600	1 456 777	106 737	13.65	0.22
ECG 300	300, 4, 4	536 976	46 382 574	6 547 211	7.08	4.18
ECG 318	300, 4, 4	586 086	46 827 423	4 426 685	10.58	3.21
NPRS 43	128, 4, 4	4 000	79 340	35 466	2.23	0.02
NPRS 44	128, 4, 4	24 125	398 471	136 658	2.91	0.10
Video	150, 5, 3	11 251	210 089	91 397	2.30	0.056
Shuttle, TEK 14	128, 4, 4	5 000	490 342	65 353	7.50	0.06
Shuttle, TEK 16	128, 4, 4	5 000	546 369	69 912	7.81	0.055
Shuttle, TEK 17	128, 4, 4	5 000	476 616	71 436	6.67	0.057

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. For these calculations the min and max D-speedups are 2.23 and 13.65, while the mean D-speedup is 6.26, while the standard deviation i 4.00

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

	# of distance calls			Runtimes [s]		
File	HOT SAX	HST	D-speedup	HOT SAX	HST	T-speedup
Daily commute	4 373 481	819 880	5.33	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

Statistics for the D-speedup: Max = 19.48, Min = 4.05, Mean = 8.44. Std. Dev. = 4.85. Statistics for the T-speedup: Max = 15.29, Min = 2.52, Mean = 6.14, Std. Dev. = 3.96. Although for 10 out of 12 of these calculations the execution time for HST requires less than 1s (and so the beginning functions play an important role in determining the total execution speed), the values of the mean T-speedup (8.44) and D-Speedup (6.14) are rather close

discords. For these reasons, we excluded ECG 308 and ECG 0606. In Table 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. We point out that, for these datasets, even when searching 10 discords, HST cannot express at best since the total execution times are very small. Its execution times are in fact less than 1 second in 10 out of 12 calculations (thus the parts of the code that take tenths of seconds play an important role in the overall execution time). Nonetheless we can observe that the gap between HOT SAX and HST increases since the maximum D-speedup is now 19.48 and the mean is 8.44. At this point, one might be interested in understanding the characteristics of the discord search which determine this variety of results.

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. 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 k.

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 Table 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 large). 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" (or anomalies), 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" (anomalies) 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* ≈ 2. In an unfortunate event where the "magic" ordering fails completely, one 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 Table 3 those datasets of Table 1 where HOT SAX performs better, and at the end those requiring more calculations. For these problems, HST *cps* is in between 4 and 15 (the mean *cps* is 9, and standard deviation is 3.84). The *cps* obtained for

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

	Cost per sequence		
File	HS	HST	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

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. Statistics for Hot Sax cps: Max = 109, Min = 9, Mean = 56, Std. Dev. = 37.83 Statistics for HST cps: Max = 15, Min = 4, Mean = 9, Std. Dev. = 3.84 The standard deviation of the *cps* for HOT SAX is one order of magnitude higher (37.83 vs 3.84) than that of HST, showing that this former algorithm is more subject to the nature of the time series

HOT SAX shows much more variation (mean *cps* 56, and std. dev. = 37.83), indicating that the performance of this algorithm depends strongly on the underlying data, while HST is less affected by the kind of data to be analyzed. These calculations are in general much easier for HST since its average *cps* is about 6 times smaller than the one of HOT SAX.

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*∼ 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 is ∼ 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. In the case of "easy searches" HOT SAX has already near to optimal performances and it is not possible to improve it very much. 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.

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. 3 (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:

$$p_i = \frac{\sin(0.1 \cdot i) + E\varepsilon + 1}{2.5} \tag{7}$$

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, (Table 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). The mean *cps* for HOT SAX is 486, while for HST it is 32. For both of the algorithms the *cps* has a minimum when the amplitude of the noise is 0.5. In summary, the maximum D-speedup of HST over HOT SAX is 104 for ECG 300 for very low noise time series where E= 0.0001 (we use the same SAX parameters of Table 1, a part for the sequence length).

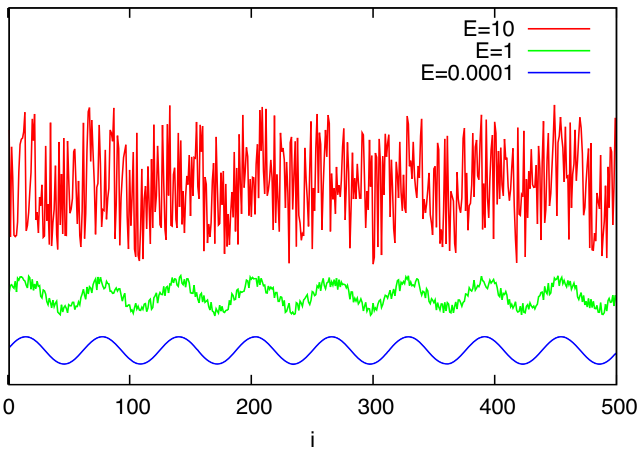
Table 4 Number of distance calls and *cps* (for one discord, $k = 1$) as a function of the noise amplitude E for the two algorithms, HOT SAX and HST

	# of distance calls		cps	
E	HOT SAX	HST	HS	HST
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

Statistics for Hot Sax *cps*: Max = 1226, Min = 67, Mean = 486, Std. Dev. = 496.20. Statistics for HST cps: Max = 155, Min = 8, Mean = 32, Std. Dev. = 50.06. The max *cps* value for HST is reached for very noisy time series (where the amplitude of the noise is 10 times the amplitude of the signal), nonetheless the cps for HOT SAX is still one order of magnitude worse than that of HST (1020 vs 155). For very low noise time series, where disambiguating the peaks of the matrix profile becomes difficult, HST (*cps*=12) is two orders of magnitude faster than HOT SAX (*cps*=1226)

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. HST remains about 6-7 times faster than HOT SAX also in this condition. An anomaly search, however, is useful when most of the sequences are expected to be



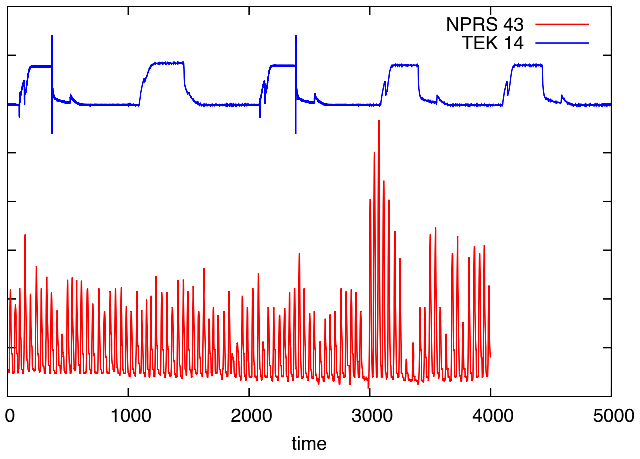


Fig. 3 (Left) Extracts of synthetic time series with different levels of noise E. In order to avoid overlaps, they have been shifted vertically. (Right) NPRS 43 and TEK 14 time series in arbitrary units. TEK 14 looks simpler and smoother, but it has a higher cost per sequence than NPRS 43

"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 (more detail). It becomes desirable to understand to which extent the calculation speed is affected by the length of the discords. Of course, we can expect an increase of the execution time when searching for longer sequences. 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 analyse 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 Table 1, ECG 300 and ECG 318, by increasing the length of the sequences s.

The values of Table 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 detail, the maximum D-speedup (101) has been obtained for ECG 318 in the case of 2340 points sequences, while the mean D-speedup is 56. In the case of ECG 300, the maximum D-speedup is 83 (obtained for sequences of 1880 points) and the mean D-speedup is 51.

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 involved in the calculation. This set depends strongly on the length of the sequences.

4.3 RRA and HST

RRA is freely available on the internet [21, 22], 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

Table 5 Cost per sequence and speedup as a function of the length of the sequences s for HOT SAX and HST, for the dataset ECG 300 (left) and for ECG 318 (right)

ECG 300			
	Cost per sequence		
s	HOT SAX	HST	D-speedup
300	87	12	7
460	201	11	17
920	494	10	50
1380	1 553	19	82
1880	857	10	83
2340	750	10	71
ECG 318			
300	80	7	11
460	113	6	18
920	510	9	56
1380	703	12	59
1880	2 026	21	94
2340	3 137	31	101

The other SAX parameters are $P = 4$, and alphabet=4. Statistics regarding the D-speedup as a function of the sequence length s for the time series ECG 300: Max = 83, Min = 7, Mean = 51, Std. Dev. = 33.09. In this case the D-speedup reaches its maximum for $s = 1880$. Both HOT SAX and HST experience the highest cps for $s = 1380$, this is an indication that the signal has becomes particularly complex for this sequence length. Statistics regarding the D-speedup for ECG 318: Max = 101, Min = 11, Mean = 56, Std. Dev. = 37.27. For this dataset the D-speedup increases with increasing s and the maximum is obtained with the longest sequences

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.

The results reported for RRA [19, 21] are substantially different in respect to the numerical experiments we obtained with their code (Table 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 Table 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 RRA in respect to HOT SAX becomes more limited. According to our numerical experiments (Tables 6 and 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 [19] 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 letters describing one SAX cluster (the quantity P in Table 1) 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 (and they are listed in Table 6).

The results of Table 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). The mean D-speedup (6.61) shows that HST is not only exact, but also generally much faster than RRA.

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 with HST (although the nature of the two algorithms is rather different). The version of DADD used for these tests is a freely available C++ code by [30]. This code is expected to process non-overlapping sequences

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"

			# of distance calls		
File	s, P, alphabet	Length	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

"Length" refers to the number of points in the time series under observation. The values of s, P, and alphabet follow those of [21]. 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. Statistics regarding the D-speedup: Max = 30.52, Min = 1.49, Mean = 6.61, Std. Dev. = 7.45 All these datasets can be downloaded from GitHub [19]

arranged in pages one after another. For this reason, the concept of self match becomes irrelevant and it has not been implemented. The algorithm computes the Euclidean distance between sequences, without Z-normalization (but the sequences can be z-normalized before being processed by DADD). Each page contains 10^4 sequences of length 512 points. Given the different data formats processed by HST and DADD, we had to re-arrange the time series we used for the comparison in order to obtain coinciding results. From the datasets of Table 6 we selected those with more than 10511 points, and we created one page of 10000 sequences for each of them (the datasets that did not contain enough points to create one page and were discarded). The pages thus formed contain overlapping sequences, however the aim of these experiments is to pinpoint the execution speed only and to obtain coinciding results.

DADD is a two-step algorithm, the first step is aimed at building a restricted pool of sequences with an *nnd* higher than a certain threshold *r* (discord defining range). During the second phase, the discords are searched among the sequences selected in the first phase. The value *r* needs to be imputed at the beginning of the calculation and it affects the calculation time. For example, if one selects a low *r* value, the first phase will return many sequences, slowing down the second phase. The opposite can also happen, selecting a high *r* value might lead to a restricted pool that does not contain all the required discords (those with a $nnd > r$). In such a scenario these discords cannot be found and require another calculation with a smaller *r* value. The value of *r* is usually obtained by sampling a subset of all of the sequences (for example 1/100 of the sequences). The first k-discords are then obtained from the sample (for the calculations of this section case we choose $k = 10$) with the help of a fast discord algorithm. The nearest neighbor distance for the 10th discord from the sample is used as the *r* parameter for DADD. One does not expect to obtain the exact *nnd* of the k-th discord but just an approximation. Unfortunately, this sampling procedure does not work particularly well for the datasets under consideration. The returned 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.

Listing 3 RRA script for TEK 14

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

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 $0.99r$ 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 has a computational cost 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 Table 7). In particular, when we use $0.99r$ the max T-speedup obtained is 25.37 while the mean T-speedup is 18.61. In the case when we input the exact r value the max T-speedup is 24.8 and the mean T-speedup is 15.98. All these results show that HST perform faster than DADD (we use the same SAX parameters as reported in Table 1)

4.5 SCAMP

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 [18]. 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 very similar to another matrix profile algorithm: STOMP [33]). 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. 4 refer to the dataset truncated at different lengths (we used the same parameters of Table 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 SAX parameters are those reported in Table 1. 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. 4, left) . 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

Table 7 The second and third columns show the calculation times (seconds) for 10 discords for DADD and HST

	0.99 r Runtimes [s]			exact r Runtimes [s]		
Dataset	DADD	HST	T-speedup	DADD	HST	T-speedup
Daily commute	10.29	0.69	14.91	10.20	0.69	14.80
Dutch Power	7.42	0.59	12.60	7.02	0.59	11.92
ECG 15	17.10	0.72	23.84	9.63	0.72	13.43
ECG 108	11.81	0.61	19.51	8.76	0.61	14.47
ECG 300	8.05	0.43	18.76	6.72	0.43	15.66
ECG 318	6.65	0.47	14.20	6.22	0.47	13.29
NPRS 44	10.82	0.55	19.71	10.71	0.55	19.50
Video	15.25	0.60	25.37	14.91	0.60	24.80

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 10000 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%). Statistics regarding the T-speedup for 0.99 of the exact r: Max = 25.37, Min = 12.6, Mean = 18.61, Std. Dev. = 4.54 Statistics regarding the T-speedup for the exact r: Max = 24.8, Min = 11.92, Mean = 15.98, Std. Dev. = 4.21. The mean T-speedup of HST is between 16 and 19 (and these timings do not include the cost of obtaining r)

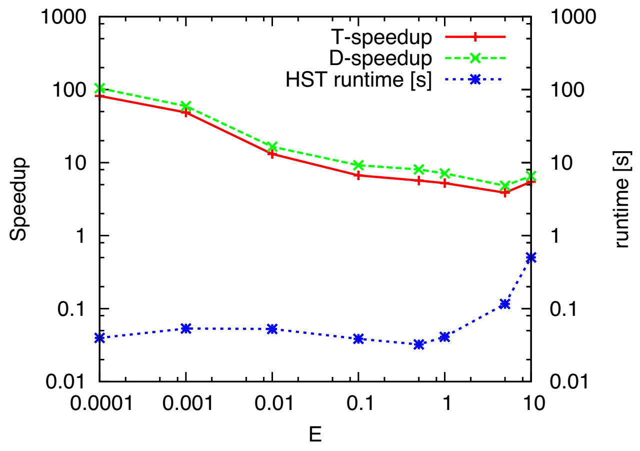


Fig. 4 The T-speedup and D-speedup of HST as a function of the noise amplitude E in the case of a synthetic time series. All the time series comprise 20 000 points, $s = 120$, $P = 4$, and alphabet=4

writing the results on the disc. For a slightly longer time series (TEK 14, 5000 points) the two algorithms show comparable times. HST has been designed to be useful for long and complex tasks, since for times series of comprising only a few thousand points even a brute force approach would execute in a matter of seconds. 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, this corresponds to a modification of the discord concept more closely related to anomalies, called: *significant discord*, [2]. 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 [34] 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 [28]. 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 $s = 512$, $P = 128$, $alphabet = 4$.

This timing is comparable with the fastest exact Matrix Profile parallel implementations exploiting graphic cards [34, 36], 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 Table 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. 5 (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. 5 (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 roughly proportional to the length of the sequences s (Fig. 6).

The last factor determining the runtimes is the length of the time series. The experiments on ECG 300 show (Fig. 4, 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. 4 (left) seem to indicate that it can be useful.

4.8 Analysis

The numerical results of the previous sections show that HST performs very well if compared to the other discord search algorithms. In this section we provide a theoretical analysis, taking into account the heuristic nature of HST and the fact that most of the other algorithms used for these tests are based on very different ideas.

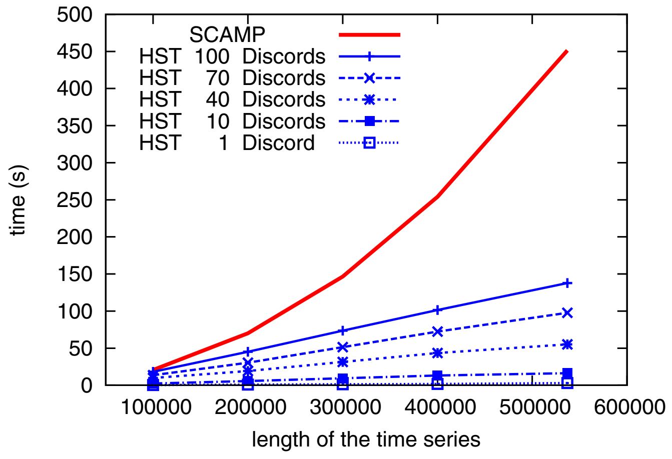
HST and RRA have two completely different approaches. RRA are based on the grammar rules obtained from the SAX words. If one uses different SAX parameters, this leads to different words and as a consequence the returned anomalies can change. This implies that RRA results are not exact and they also depend on the choice of grammar algorithms. It might be useful to notice that also HST is based on SAX, but HST removes sequences from the search space only when their *nnd* becomes smaller than the actual best value. The results are exact and the search space reduction is based on quickly finding close neighbors for all the sequences. This reduction phase improves as the algorithm proceeds and it is not fixed once for all.

DADD uses a two step procedure where the first step is aimed at quickly discarding sequences that cannot be the discord. This reduces the size of the search space but it is based on the knowledge of the r-parameter, i.e. the *nnd* of the first discord. An accurate knowledge of this quantity is essential and requires pre-processing, while HST does not need it. The search space reduction is essentially limited to the first phase, while in the second phase the algorithm needs to perform many more distance calls to disambiguate which sequence of the restricted pool of sequences is the discord. At variance HST keeps on reducing the discord search space during its execution and it also dynamically improves the guess regarding the best discord candidates (without needing an initial parameter).

SCAMP is an exact algorithm of the Matrix profile, its complexity is not dependent on the underlying time series and it is quadratic with the number of points.

Let's now consider in detail the difference between HOT SAX and HST. They are both heuristics but they share many common features and it becomes easier to understand why HST performs better.

	HOT SAX	HST
order of the outer loop based on:	SAX cluster sizes (static)	warm-up + CNP *nnd*s (dynamic)
order of the inner loop based on:	SAX + random	SAX
search space reduction after inner loop	absent	based on CNP



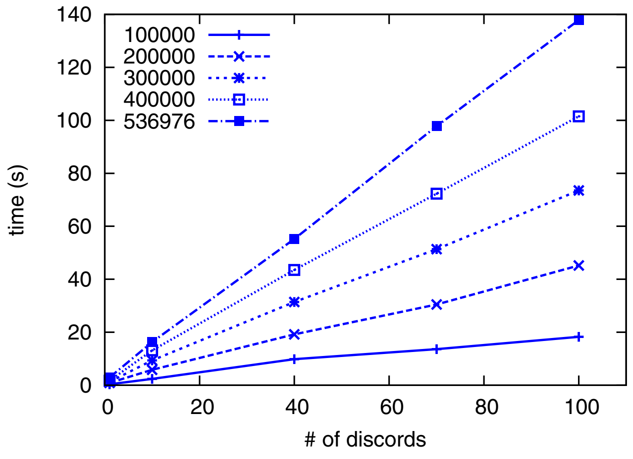
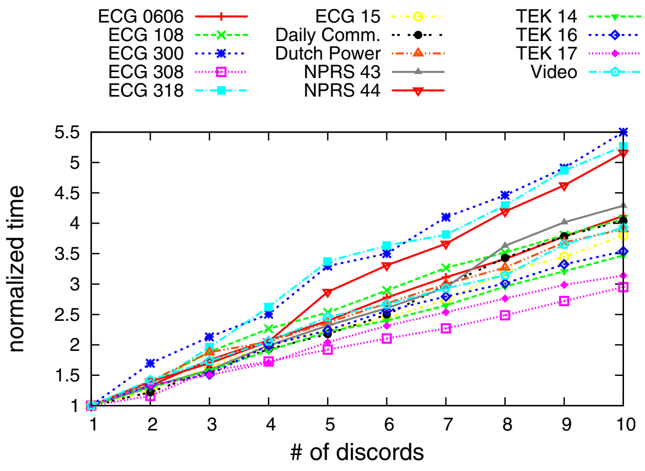


Fig. 5 (Left) Calculations performed on sections of the ECG 300 dataset. HST has been run using the same SAX parameters presented in Table 6l; for each slice of the time series five calculations were performed, for: 1, 10, 40, 70, and 100 discords. The runtimes on the same datasets for SCAMP include 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



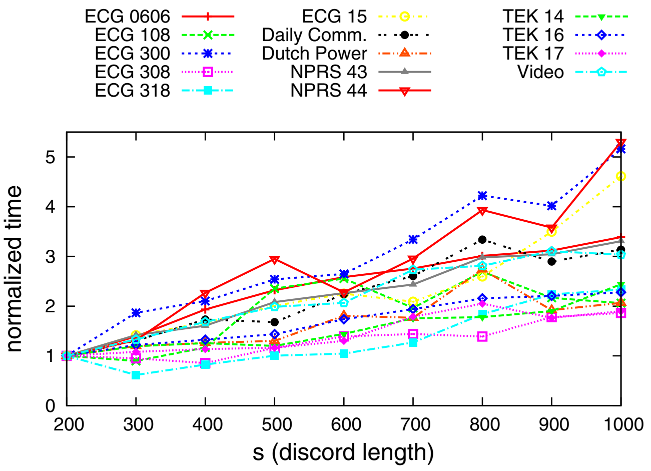


Fig. 6 (Left) Running times as a function of the number of discords searched, normalized with the time of the first discord. The length of the sequences in these cases is $s = 100$. (Right) The running times for the search of the first discord as a function of the length of the sequences s. These values have been normalized with the running times of each dataset for $s = 200$

HOT SAX bases its heuristic on the fact that discords should be found in small SAX clusters. The SAX procedure determines once for all the size of the clusters and thus the order of the outer loop. HST, instead, provides immediately an approximate nnd for all of the sequences (with the warm-up) and bases the order of the outer loop on this quantity. Since the nnds improve during the execution of the program, the order of the outer loop changes accordingly. As the algorithm proceeds the system keeps track of the added information. The new order of the external loop becomes more and more accurate during the execution, while HOT SAX does not take into account this information. The inner loops of the two algorithms are very similar, and follow the idea that they can be skipped as soon as the approximate nnds of the sequences under scrutiny drop below the best so far discord candidate. At the beginning one searches in the SAX cluster containing the sequence, and then the other clusters. The second important difference between HST and HOT SAX is the following: after each inner loop, HST performs inexpensive search space reductions (the long range time topology). If one sequence can be excluded from the discord search space, most of its time neighbors can likely be ruled out with just one distance call each. In the case of good discord candidates this can save a number of distance calls proportional to the product of the length of the sequences s and the total size of the search space N if compared to HOT SAX. Each peak contains approximately s sequences and each of them might need to be compared to most of the other sequences of the time series N.

4.9 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 from 50% to 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 nnds. At the beginning these quantities are obtained with the warm-up process. Later, the algorithm uses the CNP property to quickly improve the nnds and consequently reduce the discord search space. 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 *nnd*s 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 counter-intuitive 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 Section 4.6 and for the useful information regarding the Matrix Profile.

References

1. Avogadro P, Dominoni MA (2020) An approximate high quality nearest neighbor distance profile. In: Communications in Computer and Information Science, Springer International Publishing, pp 158–182. https://doi.org/10.1007/978-3-030-66196-0_8
2. Avogadro P, Palonca L, Dominoni MA (2020) Online anomaly search in time series: significant online discords. Knowledge and Information Systems. https://doi.org/10.1007/s10115-020-01453-4
3. Benchmarksgame-team (2020) The computer language benchmarks game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/
4. 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
5. 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. https://doi.org/10.1109/KSE.2011.11
6. Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. ACM Comput Surv 41(3):15:1–15:58. https://doi.org/10.1145/1541880.1541882
7. Chau PM, Duc BM, Anh DT (2018) Discord discovery in streaming time series based on an improved HOT SAX algorithm. In: Proceedings of the Ninth International Symposium on Information and Communication Technology - SoICT 2018, ACM Press. https://doi.org/10.1145/3287921.3287929
8. Gao Y, Lin J (2018) Exploring variable-length time series motifs in one hundred million length scale. Data Min Knowl Disc 32(5):1200–1228. https://doi.org/10.1007/s10618-018-0570-1
9. 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.fullPMID:1085218; https://doi.org/10.1161/01.CIR.101.23.e215
10. Gupta M, Gao J, Aggarwal C, Han J (2014) Outlier detection for temporal data: a survey. IEEE Trans Knowl Data Eng 26(9):2250–2267. https://doi.org/10.1109/TKDE.2013.184
11. Hu M, Feng X, Ji Z, Yan K, Zhou S (2019) A novel computational approach for discord search with local recurrence rates in multivariate time series. Inf Sci 477:220–233. https://doi.org/10.1016/j.ins.2018.10.047
12. Keogh E, Kasetty S (2003) On the need for time series data mining benchmarks: A survey and empirical demonstration. Data Min Knowl Discov 7(4):349–371. https://doi.org/10.1023/A:1024988512476
13. 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
14. Khanh NDK, Anh DT (2012) Time series discord discovery using WAT algorithm and iSAX representation. In: Proceedings of the Third Symposium on Information and Communication Technology - SoICT. ACM Press. https://doi.org/10.1145/2350716.2350748
15. 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, DMKD '03. ACM, New York, pp 2–11. https://doi.org/10.1145/882082.882086
16. Moody GB, Mark RG (2001) The impact of the mit-bih arrhythmia database. IEEE Eng Med Biol Mag 20(3):45–50. https://doi.org/10.1109/51.932724
17. Nakamura T, Imamura M, Mercer R, Keogh EJ (2020) (2020) Merlin: Parameter-Free discovery of arbitrary length anomalies in massive time series archives. In: ICDM
18. SCAMP (2020) Matrix profile on github. https://github.com/zpzim/SCAMP
19. Senin P (2019) Grammarviz 3.0. https://github.com/GrammarViz2/grammarviz2_src/

20. Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, 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, pp 468–472
21. Senin P, Lin J, Wang XTO, 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
22. 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. https://doi.org/10.1145/3051126
23. Shieh J, Keogh E (2008) isax: Indexing and mining terabyte sized time series. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, KDD '08, pp 623–631, https://doi.org/10.1145/1401890.1401966
24. Son NT (2020) An improvement of disk aware discord discovery algorithm for discovering time series discord. In: 2020 5th International Conference on Green Technology and Sustainable Development (GTSD). IEEE, https://doi.org/10.1109/gtsd50082.2020.9303111
25. Song K, Ryu M, Lee K (2020) Transitional SAX representation for knowledge discovery for time series. Appl Sci 10(19):6980. https://doi.org/10.3390/app10196980
26. 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. https://doi.org/10.1109/KSE.2018.8573409
27. Wang L, Lu F, Cui M, Bao Y (2019) Survey of methods for time series symbolic aggregate approximation. In: Communications in Computer and Information Science, Springer Singapore, pp 645–657. https://doi.org/10.1007/978-981-15-0118-0_50
28. Willett DS, George J, Willett NS, Stelinski LL, Lapointe SL (2016) Machine learning for characterization of insect vector feeding. PLOS Comput Biol 12(11):1–14. https://doi.org/10.1371/journal.pcbi.1005158
29. Yang CL, Darwin F, Sutrisno H (2019) Local recurrence rates with automatic time windows for discord search in multivariate time series. Procedia Manuf 39:1783–1792. https://doi.org/10.1016/j.promfg.2020.01.261
30. Yankov D, Keogh E, Rebbapragada U (2008) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowl Inf Syst 17:241–262. https://doi.org/10.1007/s10115-008-0131-9
31. 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. https://doi.org/10.1109/ICDM.2016.0179
32. Zhu B, Jiang Y, Gu M, Deng Y (2021) A GPU acceleration framework for motif and discord based pattern mining. IEEE Trans Parallel Distrib Syst 32(8):1987–2004. https://doi.org/10.1109/tpds.2021.3055765
33. 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. https://doi.org/10.1109/ICDM.2016.0085
34. 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. https://doi.org/10.1109/ICDM.2018.00099
35. 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. https://doi.org/10.1007/s10115-017-1138-x
36. 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, SoCC '19. Association for Computing Machinery, New York, p 74–86. https://doi.org/10.1145/3357223.3362721
37. 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

Paolo Avogadro Ph.D. His research interests include machine learning, parallel computing, data analysis, and in general, understanding nature through computation. His experiences include working at the University of Milan (Italy), at RIKEN (Japan), at Texas A&M-Commerce (USA), and at the University of Milano-Bicocca(Italy).

Matteo Alessandro Dominoni Ph.D. researcher at Computer Science Department, is the head of the Information Discovery and Application (ID&A) laboratory at University of Milano-Bicocca. Matteo Dominoni's team has been involved in research activities regarding learning and social networks, machine learning and data structures (best paper award at the KMIS'15, outstanding paper at Esociety'16 and best paper award at #EARTH 2018 international conferences). He has been principal investigator on the educational project "Pollicina," co-financed by the ROP ERDF 20142020, and is the director of the 2nd level Master degree "ICT Management" at the University of Milano-Bicocca.