Luo W, Gallagher M, Wiles J. Parameter-free search of time-series discord. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(2): 300–310 Mar. 2013. DOI 10.1007/s11390-013-1330-8

Parameter-Free Search of Time-Series Discord

Wei Luo[1], Marcus Gallagher[2], *Member*, *IEEE*, and Janet Wiles[2], *Member*, *IEEE*

[1]*School of Information Technology, Deakin University, Geelong, VIC 3220, Australia*

[2]*School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, Australia*

E-mail: wei.luo@deakin.edu.au; {marcusg, wiles}@itee.uq.edu.au

Received February 15, 2012; revised November 14, 2012.

Abstract Time-series discord is widely used in data mining applications to characterize anomalous subsequences in time series. Compared to some other discord search algorithms, the direct search algorithm based on the recurrence plot shows the advantage of being fast and parameter free. The direct search algorithm, however, relies on quasi-periodicity in input time series, an assumption that limits the algorithm's applicability. In this paper, we eliminate the periodicity assumption from the direct search algorithm by proposing a reference function for subsequences and a new sampling strategy based on the reference function. These measures result in a new algorithm with improved efficiency and robustness, as evidenced by our empirical evaluation.

Keywords time series anomaly detection, recurrence structure, direct discord search, parameter-free algorithm

1 Introduction

Anomaly detection in time series has a broad range of applications (see for example [1-4]). Diverse applications have led to various characterizations of anomalies and consequently a large number of detection methods (see for example [2, 5-8]). In the data mining community, a popular definition of time series anomalies is the *time series discord*[9]. Intuitively, the discord of a time series is the subsequence most different from all other non-overlapping subsequences. The definition, despite its simplicity, captures an important class of anomalies, the relevance of which has been shown in several data mining applications[9-12]. For example, Fig.1 shows an electrocardiography (ECG) recording used to detect premature ventricular contraction arrhythmia[13]. The presence of arrhythmia is reflected in the time series discord shown in the circle.

The discord of a time series can be found by computing the pair-wise distances among all subsequences. That means $O(m^2)$ comparisons for a time series of length m. With the emergence of affordable sensor technologies and the prevalence of automatic data collection mechanisms, long time series are being created every day. For such long time series, $O(m^2)$ comparisons are still computationally prohibitive. For example in a typical epilepsy application with 256 Hz electroencephalography (EEG, e.g., [14]), 24 hours of recording would produce a time series of more than 22 million values and more than 200 trillion pairs of subsequences. For faster discord search, various algorithms have been proposed. These algorithms can be classified into two groups: index-based search and direct search.

1.1 Index-Based Discord Search

Index-based search was first proposed by Keogh and co-authors in [9]. In index-based search, all real valued subsequences are first encoded as character strings. In [9], this was done through discretizing the time series. Then the strings are indexed so that similar subsequences have similar index keys (this is similar to how character strings in a database are indexed for faster data retrieval). Finally, pair-wise comparisons of subsequences are carried out with embedded for-loops, with

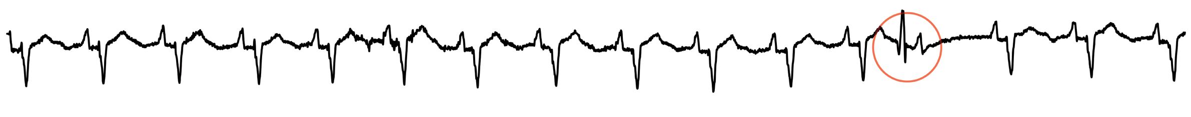
Fig.1. ECG segment for a patient with premature ventricular contraction arrhythmia. The anomalous subsequence circled indicates the presence of the arrhythmia.

Regular Paper

Support by Australian Research Council Linkage Grant No. LP 0776417.

pruning of the subspace determined by the index. These steps are explained with more details in Subsection 2.2.

The efficiency of an index-based search algorithm is determined by how subsequences are encoded as strings and how the for-loops are ordered. To improve the discretization in [9], Pham[15] proposed a way to select break points based on k-means clustering. Instead of simple discretization, coefficients resulted from wavelet transformations were used to encode subsequences[11,16]. A multi-resolution extension of [9] has also been proposed (see [17]). To realize early exit of the for-loops, a heuristic based on symbol frequency was recently proposed[18].

Index-based algorithms rely on a set of tuning parameters for efficient encoding of subsequences. For example, HOT SAX[9] requires parameters *alphabet size* and *word size* to be set; WAT[11] uses an adaptive word size, yet alphabet size is still required. These parameters often have no natural correspondence in the application domain and few guidelines have been provided on selecting optimal index parameters. The difficulty of finding the optimal tuning parameters was discussed by Yankov and Keogh in [12].

1.2 Direct Discord Search

To eliminate the tuning parameters in index-based search, direct discord search was proposed in [19]. A direct search algorithm looks for a discord by directly sampling the pair-wise distances among subsequences. In [19], a sampling strategy was developed based on the quasi-periodicity present in many time series. The algorithm runs significantly faster on quasi-periodic time series than the typical index-based algorithms. More details of the algorithm can be found in Subsection 2.3.

In this paper we propose a generalization that eliminates the quasi-periodicity assumption in direct search. Like the original algorithm described in [19], the new algorithm exploits a time series' recurrence structure, but replaces the periodicity assumption with a reference function. The reference function leads to a new sampling strategy that makes the search algorithm more efficient and robust.

Direct discord search can be regarded as a novel application of the recurrence plots. Recurrence plots were first introduced by Eckmann[20] as a diagnostic and visualization tool for time series originating from the study of dynamic systems. A recurrence plot for a time series T is a 2-D plot whose value at location (i, j) is a function of $\|T[i] - T[j]\|$. The function values are usually converted into binary values by taking a threshold, but in this paper we shall use the unthresholded version[21-23]. Recurrence quantification analysis (RQA, see [24]) is an excellent technique to quantify features in a recurrence plot. However, RQA produces global characterizations that cannot be directly used for discord search. The direct discord search algorithm bridges the gap by exploiting both global and local features of a recurrence plot.

1.3 Paper Outline

This paper is organized as follows. Section 2 introduces the formal definition of time series discords and reviews existing discord search algorithms. In particular, direct discord search is reformulated to allow for further generalisation. Section 3 presents the general direct-search algorithm based on a reference function and a new sampling strategy. Section 4 evaluates the efficiency of the new algorithm. Section 5 discusses the results and their implications in time series anomaly detection and feature extraction.

2 Time Series Discords

This section reviews the definition of time-series discords and the two major types of discord search algorithms: index-based search and direct search.

2.1 Definition of Discords

This paper considers only discrete time series with equal time intervals (as in [25]). We use T to denote a time series $(t_1, t_2, \ldots, t_m)$ of length m, where each $t_i \in \mathbb{R}$; we use $T[p; n]$ to denote the length-n subsequence of T starting at the p-th position. Following the convention set by Keogh and co-authors[9], we assume a fixed length n for all subsequences of T; hence we also write $T[p]$ for $T[p; n]$. Also following the convention in [9], we standardize each subsequence:

$$T[p] \leftarrow \frac{T[p] - mean(T[p])}{sd(T[p])},$$

where $mean(T[p])$ is the mean of values in $T[p]$ and $sd(T[p])$ the standard deviation of the values in $T[p]$. The discord of T is defined over the set of all standardized subsequences $\{T[p] : 1 \leqslant p \leqslant m - n + 1\}$. The distance between two subsequences $T[p]$ and $T[q]$ is denoted as $dist(p, q)$. As in [9], by default $dist(p, q)$ is the Euclidean distance between $T[p]$ and $T[q]$ when they are considered points in vector space $\mathbb{R}^n$. Nevertheless, the results in this paper also apply to other common distance definitions. For convenience, we write M for $m-n+1$, the starting location of the right-most length-n subsequence in the whole series.

The distance matrix $\boldsymbol{D}$ for length-n subsequences of a time series T is an $M \times M$ matrix with $D[p, q] \triangleq dist(p, q)$. Fig.2 visualizes the distance matrix for the time series shown in Fig.1 when the subsequence length $n = 360$. Such a plot is called an unthresholded recurrence plot[21], a global recurrence plot[22], or simply a distance plot[23]. The plot is widely used in dynamical system analysis with time series data.

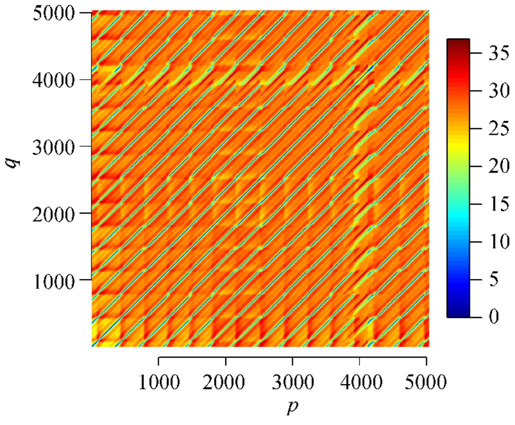

Fig.2. Distance plot showing the values of a distance matrix. The time series contains 5 400 values, part of which is shown in Fig.1. The subsequence length n is 360. Legend on the right shows the color encoding for distance.

The repeating diagonal pattern evident in Fig.2 shows that 1) at regularly spaced intervals there are locations p and q with relative small distance values, and 2) the small distance values exist along lines where p and q are changing at the same rate. This is an indicator of strong periodicity in the time series from Fig.1.

We define a distance query to be a function call to evaluate an element of $\boldsymbol{D}$; it clearly has time complexity $O(n)$. It is a convention to measure the efficiency of a discord search algorithm with the number of distance queries made by the algorithm[9].

For convenience of exposition, we define the following constructs. Given a distance matrix $\boldsymbol{D}$, a distance vector, denoted by $\boldsymbol{d}$, is a length-M vector with the p-th element $d[p] \triangleq \min_{q:|p-q|\geqslant n} D[p,q]$. A distance vector stores for each subsequence the distance to its nearest nonoverlapping neighbor. It is conventional[9] to consider only pairs of subsequences with no overlap (i.e., $|p-q| \geqslant n$). Fig.3 shows the distance vector derived from the distance matrix in Fig.2. The distance profile of position p, denoted $\boldsymbol{D}[p,\cdot]$, is the vector $(D[p,1], D[p,2], \ldots, D[p,M])$.

With the above notation, we can now reformulate the time series discord originally proposed in [9].

Definition 1 (Time Series Discord). *Let T be a time series of length m and consider only length-n subsequences of T. A subsequence $T[p^*]$ is a length-n discord of T if $d[p^*] = \max_p d[p]$ (or equivalently $\min_{q:|p^*-q|\geqslant n} D[p^*,q] = \max_p \min_{q:|p-q|\geqslant n} D[p,q]$). The position p^* is a discord location for T.*

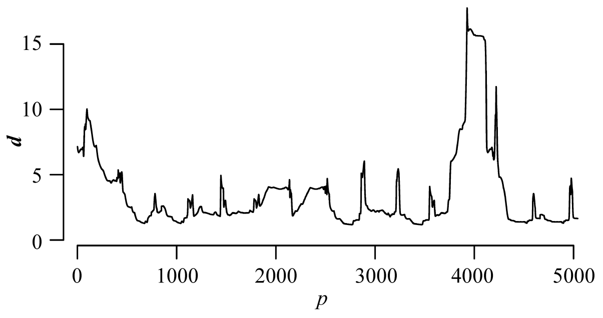

Fig.3. Distance vector values derived from the distance matrix in Fig.2.

Intuitively, a length-n discord is the most "isolated" length-n subsequence (when each subsequence is regarded as a point in the metric space $\langle \mathbb{R}^n, dist(\cdot,\cdot)\rangle$, as illustrated in Fig.4). Theoretically there can be more than one position p that maximizes $d[p]$, and hence more than one discord. In practice for a continuous-value time series (i.e., $t_i \in \mathbb{R}$), the likelihood of having two discords is negligible.

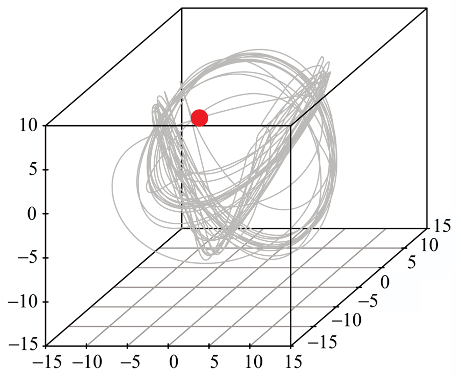

Fig.4. Location of a discord in the space spanned by the top 3 principal components for all 5 041 subsequences of the time series T in Fig.1. Each point in the plot represents a length-360 subsequence of T. The red point represents a discord for T.

On the surface, discord has a formulation similar to the minimax optimization problem, with $\max_p \min_{q:|p-q|\geqslant n} D[p,q] \leqslant \min_p \max_{q:|p-q|\geqslant n} D[p,q]$. But unfortunately for most time series, the distance matrix $\boldsymbol{D}$ does not have a single saddle point for a local search of discord to work (see Fig.2). On the other hand, a discord can always be located through full evaluation of the distance matrix $\boldsymbol{D}$, which requires $O(M^2)$ distance queries. For a reasonable length of $M = 20\,000$, it amounts to as many as 100 million distance queries. Fortunately more efficient discord search is often possible by making fewer distance queries to reach a "good-enough" partial evaluation $\widehat{\boldsymbol{D}}$. In general, there are two strategies for reducing the number of distance queries: index-based search and direct search.

2.2 Index-Based Discord Search

In index-based search, all subsequences are indexed with a small number of index keys so that similar subsequences share the same index key. Let K be a set of index keys, an index $I : \mathbb{R}^n \mapsto K$ puts all subsequences into $|K|$ bins of equal diameter $\Delta \triangleq \max_{s_1,s_2 \in I^{-1}(k)} dist(s_1, s_2)$, where k is a key in K. In other words, two subsequences $T[p]$ and $T[q]$ are in the same bins (i.e., $I(T[p]) = I(T[q])$) only if $dist(p, q) \leqslant \Delta$.

For an ideal index, a discord $T[p^*]$ will be mapped to a key shared by no other subsequence, and any other subsequence $T[p]$ will share a key with at least one non-overlapping subsequence $T[q]$. In other words, the bins produced by the index have a diameter Δ such that

$$\max_{p \neq p^*} d[p] < \Delta < \min_{p:p \neq p^*} D[p^*, p]. \tag{1}$$

With such an index, search is done iteratively following the logic outlined in Fig.5.

```
1:  Set $d^*$ to an extremely small positive number. {Initialization}
2:  for all locations $p$ ordered by the number of subsequences sharing the index key do {Outer loop}
3:      $\hat{d}[p] \leftarrow \infty$
4:      for all locations $q$ ordered by the similarity of its index key to that of $p$ do {Inner loop}
5:          if $|p - q| \geqslant n$ then
6:              Compute $D[p, q]$.
7:              if $D[p, q] < d^*$ then
8:                  Next $p$ in the outer loop.
9:              end if
10:             $\hat{d}[p] \leftarrow \min(\hat{d}[p], D[p, q])$.
11:         end if
12:     end for
13:     if $\hat{d}[p] > d^*$ then
14:         $d^* \leftarrow \hat{d}[p]$ and $p^* \leftarrow p$.
15:     end if
16: end for
17: return $p^*$ as the discord location.
```

Fig.5. Outline of index-based discord search algorithms.

The efficiency of index-based discord search depends on the orders in the outer and inner loops, which in turn rely on a good index with the right bin size. In practice, an index satisfying (1) is often difficult to find, as shown in Fig.6.

In general, an index with a diameter Δ should be found such that $\{p : D[p^*, p] < \Delta \text{ and } p \neq p^*\}$ is a set smaller than $\{q : D[p, q] < \Delta \text{ and } p \neq q\}$ for each $p \neq p^*$. This will let the index key for p^* be identified and evaluated first in the outer loop. The following proposition gives a lower bound on the search efficiency of an index-based algorithm.

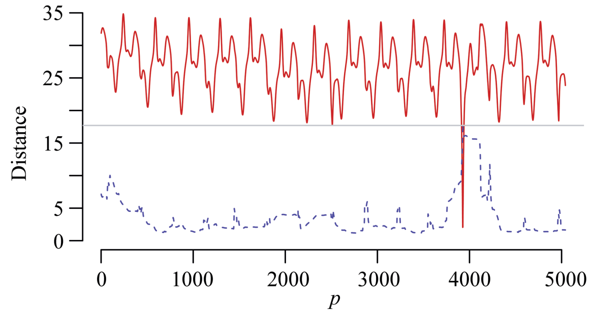

Fig.6. Illustration of the narrow gap between $\max_{p \neq p^*} d[p]$ and $\min_{p:p \neq p^*} D[p^*, p]$ for the time series in Fig.1. The solid line shows $D[p^*, p]$ and the dashed line shows $\boldsymbol{d}$. The horizontal grey line shows $\max_{p \neq p^*} d[p]$.

Proposition 1. *Let σ be the ordering in the outer loop for sequence $(1, 2, \ldots, M)$ such that location $\sigma(i)$ will be evaluated before $\sigma(i + 1)$. An index-based algorithm without caching of distance queries requires at least*

$$M + k \times (M - 2n) \tag{2}$$

distance queries to locate a discord of length n, where

$$k = |\{d[\sigma(i)] > \max_{j<i} d[\sigma(j)] : 1 \leqslant i \leqslant M\}|.$$

Proof. Suppose $d[\sigma(i)] > \max_{j<i} d[\sigma(j)]$. Before the execution of the outer loop on i, we have $d^* = \max_{j<i} d[\sigma(j)]$. As $d[\sigma(i)] = \min_{|\sigma(i)-q|>n} D[\sigma(i), q]$, line 8 in Fig.5 will not run and the inner loop will be completed with all distance queries, and there are at least $M - 2n + 1$ such queries. Since there are k such i, the algorithm requires at least $k \times (M - 2n + 1)$ distance queries. For each of the remaining $M - k$ runs of the outer loop, at least one distance query is needed for line 6. This entails additional $M - k$ distance queries. Therefore at least $M + k \times (M - 2n)$ distance queries are needed in total. □

Assuming that $\{p : D[p^*, p] < \Delta \text{ and } p \neq p^*\}$ is a set smaller than $\{q : D[p, q] < \Delta \text{ and } p \neq q\}$ for each $p \neq p^*$ and positions in the same bin are indistinguishable, we see that $E[k]$ (the expected value of k) equals

$$\frac{1}{2} \times |\{p : D[p^*, p] < \Delta \text{ and } p \neq p^*\}|, \tag{3}$$

half of the number of subsequences overlapped with the discord. For example in Fig.6, letting Δ be 17.75 would result in a $T[p^*]$ that does not share an index key with a nonoverlapping sequence $T[p]$, but $T[p^*]$ still shares an index key with 26 overlapping sequences. Therefore $E[k] = 13$ by (3). As the time series has $M = 5\,041$ subsequences of length $n = 360$, on average 61 000 or more distance queries are needed by (2) in Proposition 1.

In practice, a good index is realized by tuning parameters so that (1) is maximally realized. But few guidelines exist for selecting the tuning parameters.

2.3 Direct Discord Search

In direct discord search exemplified by [19], no index is built and hence no index parameters are needed. Instead, a distance matrix $\boldsymbol{D}$ (as shown in Fig.2) is directly sampled to generate a sparse estimate $\hat{\boldsymbol{D}}$, where each element $\widehat{D}[p,q]$ equals either $D[p,q]$ or `NA` (meaning unevaluated). Consequently an estimate $\hat{\boldsymbol{d}}$ for $\boldsymbol{d}$ can be derived by $\hat{d}[p] \triangleq \min_{q:|p-q|\geqslant n \text{ and } \hat{D}[p,q]!=\texttt{NA}} D[p,q]$; we call such an estimate $\hat{\boldsymbol{d}}$ a natural estimate of $\boldsymbol{d}$. A natural estimate has a property useful for discord search.

Definition 2 (Over Estimate). *An estimate $\hat{\boldsymbol{d}}$ is an over estimate of $\boldsymbol{d}$, denoted by $\hat{\boldsymbol{d}} \succeq \boldsymbol{d}$, if $\hat{d}[p] \geqslant d[p]$ for each p. An estimate $\hat{\boldsymbol{d}}$ is a proper over estimate of $\boldsymbol{d}$, denoted by $\hat{\boldsymbol{d}} \succ \boldsymbol{d}$, if $\hat{\boldsymbol{d}} \succeq d$ and $\hat{\boldsymbol{d}} \neq \boldsymbol{d}$.*

Lemma 1. *A natural estimate $\hat{\boldsymbol{d}}$ of $\boldsymbol{d}$ is also an over estimate of $\boldsymbol{d}$.*

Proof. This is because for each p $\min_{q:|p-q|\geqslant n \text{ and } \hat{D}[p,q]!=\texttt{NA}} D[p,q]$ is greater than or equal to $\min_{q:|p-q|\geqslant n} D[p,q]$. $\square$

Therefore, an estimated distance vector resulting from a partial evaluation of $\boldsymbol{D}$ is always above the true distance vector (see Fig.7). The following proposition gives a sufficient condition for discord to be identified from $\hat{\boldsymbol{d}}$.

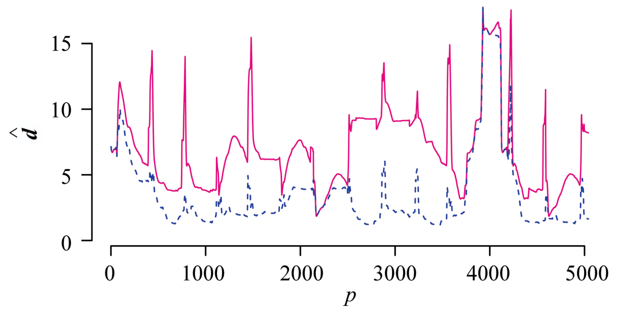

Fig.7. Illustration of the sufficient condition for discord identification. The solid line represents a natural estimate $\hat{\boldsymbol{d}}$ for $\boldsymbol{d}$. The dashed line represents the distance vector $\boldsymbol{d}$ in Fig.3. Although $\hat{\boldsymbol{d}}$ is a rough estimate of $\boldsymbol{d}$, the two vectors have the same maximum at the same position. Hence $\hat{\boldsymbol{d}}$ is sufficient for identifying the discord location.

Proposition 2. *Let $\hat{\boldsymbol{d}}$ be a natural estimate of $\boldsymbol{d}$. If there exists a location p^* such that $d[p^*] \geqslant \hat{d}[p]$ for each location p, then $d[p^*] \geqslant d[p]$ is also true for each p (i.e., p^* is a discord location).*

Proof. Since $\hat{\boldsymbol{d}}$ is a natural estimate of $\boldsymbol{d}$, by Lemma 1 $\hat{d}[p] \geqslant d[p]$ for each p. Thus $d[p^*] \geqslant \max_p \hat{d}[p]$ implies $d[p^*] \geqslant \max_p d[p]$. $\square$

Proposition 2 shows that a natural estimate $\hat{\boldsymbol{d}}$ does not have to be close to $\boldsymbol{d}$ at every position p to reveal the discord location p^*; it suffices that $\hat{d}[p] \leqslant \max(\boldsymbol{d})$ for every p. This is illustrated in Fig.7.

In direct search, two sampling strategies are used in combination to generate a good estimate $\hat{\boldsymbol{d}}$ with a small number of distance queries to $\boldsymbol{D}$.

1) *Differentiating* aims to obtain a small $\hat{d}[p]$ with minimum number of queries to the distance profile $\boldsymbol{D}[p,\cdot]$. This is similar to the inner loop of the index-based search (see Fig.5). To minimize the number of distance queries, a reference function is used. Define a reference function $f : \{1,2,\ldots,M\} \mapsto Y$ where $f(p)$ depends on the subsequence $T[p]$ and $D[p,q]$ is likely to be small when $f(p) \simeq f(q)$. The idea is that for a p, it is easier to find a q with $f(p) \simeq f(q)$ than to find a q with $T[p] \simeq T[q]$, and the gap between $f(p)$ and $f(q)$ serves as an estimate for $D[p,q]$. When $Y = \mathbb{R}$, distances in $\{D[p,q] : q \in (1,2,\ldots,M)\}$ can be queried in the order of increasing $|f(p) - f(q)|$ value (see Subsection 3.2). Differentiating is not limited to only direct discord search. For example in index-based search, an index $I : \mathbb{R}^n \mapsto K$ can be regarded as a reference function f such that $f(p) = I(T[p])$. Differentiating then means querying those $D[p,q]$ with $I(T[p]) = I(T[q])$.

2) *Traversing* aims to exploit the diagonal lines in a recurrence plot to improve a segment of $\hat{\boldsymbol{d}}$. In the phase space of a time series, a segment of trajectory may run parallel to another segment (see Fig.4), thus creating diagonal line segment $(D[p+i,q+i] : 1 \leqslant i \leqslant s) \simeq \boldsymbol{0}$ (see Fig.8 and also [23, Subsection 3.2.3]). Therefore if $\hat{d}[p]$ is small, then a sequence of small $\hat{d}[p']$ values could be found around p. Let $\hat{\boldsymbol{d}}$ be the existing estimate for $\boldsymbol{d}$ and suppose $\hat{d}[p] = D[p,q]$ is small. Then we should query $D[p-1,q-1]$ (Traversing left) and $D[p+1,q+1]$

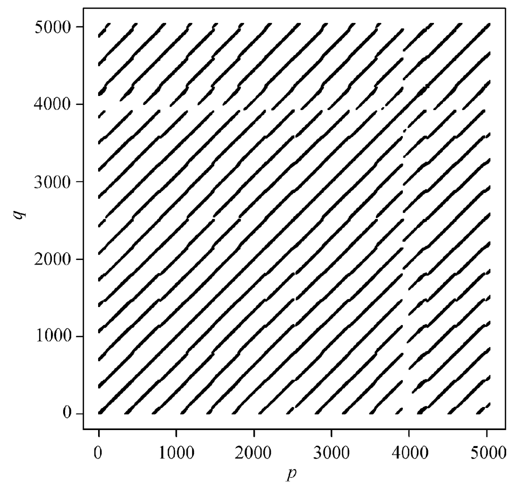

Fig.8. Recurrence plot for the ECG time series shown in Fig.1. The threshold distance ϵ is set to 17.75. That is only points (p,q) with $D[p,q] < 17.75$ are shown.

(Traversing right) as these distances are likely to be small as well (see Fig.2). Traversing can be continued to the left (or the right) as long as $D[p-i, q-i] < \hat{d}[p]$ (or $D[p+i, q+i] < \hat{d}[p]$ respectively).

With the two sampling strategies Differentiating and Traversing, direct discord search iteratively refines $\hat{\boldsymbol{D}}$ (and hence $\hat{\boldsymbol{d}}$) until the sufficient condition in Proposition 2 is satisfied. Fig.9 shows the structure of direct discord search.

```
1:  Initialize $\hat{\boldsymbol{D}}$ with Differentiating and Traversing. {Initialization}
2:  while true do
3:      Let $p^* \leftarrow \arg\max_p\{\hat{d}[p]\}$ and $Q \leftarrow \{q : \hat{D}[p^*, q] = \mathtt{NA}$ and $|p^* - q| \geqslant n\}$.
4:      while $Q \neq \varnothing$ do {Verification}
5:          Let $q \leftarrow$ a Differentiating sample from $Q$ and update $Q \leftarrow Q \setminus \{q\}$.
6:          if $D[p^*, q] < \hat{d}[p^*]$ (i.e., the current $\hat{d}[p^*]$ is not small enough) then
7:              Break to find another $p^*$.
8:          end if
9:      end while
10:     if $Q = \varnothing$ (i.e., $\hat{d}[p^*] = d[p^*]$) then
11:         return $p^*$ as the discord location.
12:     end if
13:     Refine $\hat{\boldsymbol{D}}$ with Traversing. {Refinement}
14: end while
```

Fig.9. Structure of direct discord search.

Direct discord search will always stop because $\boldsymbol{D}$ has only finite elements. When it stops, it returns a true discord location by Proposition 2. Also note that direct discord search contains no intrinsic random components. If we fix the starting point and the way to update Q, two runs of direct search would sample the same positions in the distance matrix.

In direct discord search, the search efficiency (measured by the number of distance queries) is determined by how the initialization, verification, and refinement steps in Fig.9 are implemented, which in turn depends on the Differentiating operation. In [19], the Differentiating operation is designed to exploit quasi-periodicity present in many time series.

2.3.1 Periodicity-Based Direct Search

When a time series is quasi-periodic, the periodicity entails information redundancy in the distance matrix $\boldsymbol{D}$. Hence direct search can be designed to take advantage of the periodicity in time series. One such algorithm was previously proposed in [19]; it will be called Periodicity-Based Direct Search (PBDS) in this paper.

Suppose T is a time series with a period l. Then $D[p, q] \simeq 0$ whenever p and q are multiple periods away ($p - q = k \times l$, where $k \in \mathbb{Z}$). Although the PBDS algorithm does not have an explicit reference function, we could consider that f is defined as $f(p) = (p \bmod l)$. Hence $f(p) = f(q)$ if p and q are multiple periods apart. In the initialization step of PBDS, a distance $D[p, p + k \times l]$ is queried for every p to generate an initial estimate $\hat{\boldsymbol{d}}$. The verification step of PBDS is essentially brute-force computation of $\boldsymbol{D}[p, \cdot]$. Fig.10 shows the positions sampled by the PBDS algorithm. First, the vertical line on the left shows the exhaustive query on $\boldsymbol{D}[1, \cdot]$ to estimate the period of the time series at the initialization step. Second, the scattered points multiple periods off the center diagonal line show the trace of Differentiating sampling at the verification step. Next, the diagonal segments result from the Traversing sampling at the refinement step. Finally, the vertical line at the discord location 3 927 shows a complete run of the while loop to verify the discord (lines 4~9 in Fig.9).

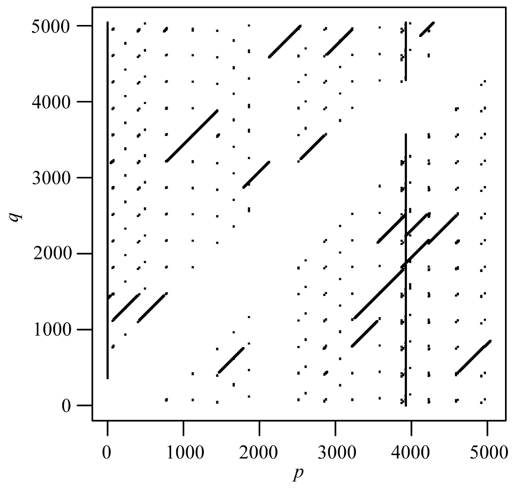

Fig.10. Positions of the distance matrix in Fig.2 sampled by the PBDS algorithm. In total 23 183 positions have been sampled.

Empirical evaluation in [19] shows that for a quasi-periodic time series, PBDS is more efficient than index-based algorithms. The PBDS algorithm, however, assumes the existence of a period l in the time series. When a time series is not quasi-periodic, the efficiency of PBDS will be compromised. In the following section, we propose a more general discord search algorithm. With a more flexible reference function, the algorithm achieves better efficiency for all periodic and non-periodic time series.

3 General Direct Search of Discord

In this section, we introduce a direct discord search algorithm that eliminates the quasi-periodic assumption on the time series. We call the algorithm General Direct Search (GDS). At the center of GDS is a new reference function.

3.1 New Reference Function for Direct Search

Consider the discord search problem with the subsequence length n setting to 1. In other words, each subsequence $T[p]$ has one value and $D[p,q] = |T[p] - T[q]|$. Let π be a sorting permutation such that $T[\pi(1)] \leqslant T[\pi(2)] \leqslant \cdots \leqslant T[\pi(i)] \leqslant \cdots \leqslant T[\pi(M)]$. Then a discord is a point with the greatest distance to its left and right neighbors in the sorted list. For each p, the value $d[p]$ can be computed in constant time as

$$\min_{i \in \{1,-1\}} (T[p] - T[\pi(\pi^{-1}(p) + i)]), \tag{4}$$

where $\pi^{-1}(\cdot)$ is defined to be the inverse permutation for π. Therefore by sorting subsequences before the search, the length-1 discord can be found in time $O(M)$ instead of $O(M^2)$. In GDS, we realise a similar reduction in distance queries by sorting on a one-dimensional transformation of subsequences.

We define a profile reference function f to be the distance profile $\boldsymbol{D}[p_0, \cdot]$ for some fixed p_0. Fig.11 shows a profile reference function when $p_0 = 1$. A profile reference function f has the following property: $D[p,q]$ is small only if $f(p) \simeq f(q)$.

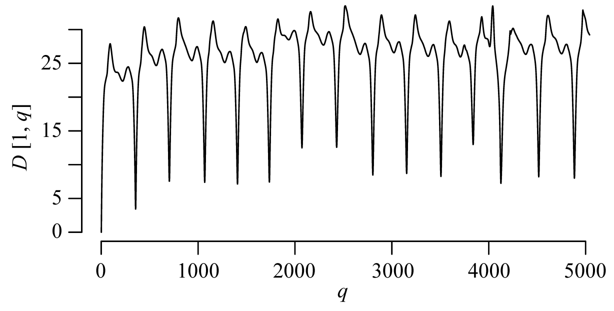

Fig.11. Distance profile $\boldsymbol{D}[1,\cdot]$ for the time series in Fig.1.

Lemma 2. *Let f be a profile reference function for GDS; that is, $f(p) = D[p_0, p]$ for each p. Then $|f(p) - f(q)| \leqslant D[p,q] \leqslant f(p) + f(q)$ for every pair of positions p and q. In other words, $D[p,q]$ has an lower bound $|f(p) - f(q)|$ and an upper bound $f(p) + f(q)$.*

Proof. It follows directly from the triangle inequality and the reverse triangle inequality of a normed vector space. □

3.2 Differentiating with the New Reference Function

A profile reference function f is real-valued; that is f maps $(1, 2, \ldots, M)$ to $\mathbb{R}$. Hence given a position p, a gap function g_p can be defined such that $g_p(q) = |f(p) - f(q)|$. As $g_p(q)$ provides a lower bound for $D[p,q]$, it can be sorted in increasing order to guide the queries of the profile $\boldsymbol{D}[p,\cdot]$. This will likely generate a tight $\hat{d}[p]$ with a small number of queries to $\boldsymbol{D}[p,\cdot]$. Suppose we have a sorting permutation π such that $g_p(\pi(1)) \leqslant g_p(\pi(2)) \leqslant \ldots \leqslant g_p(\pi(i)) \leqslant g_p(\pi(i+1)) \leqslant \ldots \leqslant g_p(\pi(M))$. If $\hat{d}[p] < g_p(\pi(i))$, then $\hat{d}[p] < g_p(\pi(j)) < D[p, \pi(j)]$ for every $j > i$. Then $\hat{d}[p] = d[p]$ can be verified with only i distance queries.

Sorting all gap functions entails $O(M^2 \log(M))$ computational complexity, and hence has to be avoided. In GDS, we sort only the values for $f(p)$ and use two pointers for each p to find the next $f(q)$ closest to $f(p)$. Let σ be a permutation for $(1, 2, \ldots, M)$ such that $f(\sigma(i)) < f(\sigma(i+1))$ for every $i < M$. Let $F = (f(\sigma(1)), f(\sigma(2)), \ldots, f(\sigma(M)))$ be the sorted sequence. Then $f(p)$ will be in the $\sigma^{-1}(p)$-th place in F. For each p, a left pointer bl_p and a right pointer br_p are defined; they initially point to $f(\sigma(\sigma^{-1}(p) - 1))$ and $f(\sigma(\sigma^{-1}(p) + 1))$ (see Fig.12). To simulate the order of the gap function g_p, the two pointers bl_p and br_p move to left and right respectively until $bl_p \leqslant 1$ or $br_p \geqslant M$. With the pointers, Differentiating for all p locations can be done with a sorted list shared by the locations.

$f(\sigma(\sigma^{-1}(p) - 1))$	$f(p)$	$f(\sigma(\sigma^{-1}(p) + 1))$	$f(\sigma(\sigma^{-1}(p) + 2))$
$\sigma^{-1}(p) - 1$	$\sigma^{-1}(p)$	$\sigma^{-1}(p) + 1$	$\sigma^{-1}(p) + 2$
$\uparrow$			$\uparrow$
bl_p			br_p

Fig.12. Illustration of pointers used in Differentiating. The function f is sorted in increasing order. Here because $|f(\sigma(\sigma^{-1}(p) + 1)) - f(\sigma(\sigma^{-1}(p)))| \leqslant |f(\sigma(\sigma^{-1}(p) - 1)) - f(\sigma(\sigma^{-1}(p)))|$, the right pointer has moved to the right and the left pointer stays in its initial position.

3.3 Initialization and Refinement in GDS

In GDS, the reference function f can be used as an initial estimation $\hat{\boldsymbol{d}}$, by defining

$$\hat{d}[p] = \begin{cases} f(p), & \text{if } |p - p_0| \geqslant n, \\ \text{infinity}, & \text{otherwise}. \end{cases} \tag{5}$$

Our experiments show that the choice of p_0 rarely matters in the search. In this paper, we assume that $p_0 = 1$, the first location of the time series.

The refinement step can be implemented using Traversing. Fig.13 shows how the Traversing operation can be repeatedly applied to improve $\hat{\boldsymbol{d}}$. Note that the location $\arg\min_q \hat{D}[p,q]$ is retrieved from cache, not being computed every time.

For the time series in Fig.1, Fig.14 shows the results of repeated Traversing on the initial $\hat{\boldsymbol{d}} = \boldsymbol{D}[p_0, \cdot]$. As shown in the figure, repeated Traversing is very effective in exploiting a small number of small $\hat{d}[p]$ values to generate continuous blocks of relatively tight distance estimates.

```
1:  For each $p$, let to.traverse$[p] \leftarrow$ true.
2:  while $P \leftarrow \{p$ : to.traverse$[p]$ = true$\}$ is nonempty do
3:      $p \leftarrow \arg\min_{p\in P} \hat{d}[p]$.
4:      Traverse to right from location $(p, \arg\min_q \hat{D}[p,q])$
        until $D[p+i, \arg\min_q \hat{D}[p,q]+i] > \hat{d}[p+i]$ at step $i$.
5:      Traverse to left from location $(p, \arg\min_q \hat{D}[p,q])$ until $D[p-i, \arg\min_q \hat{D}[p,q]-i] > \hat{d}[p-i]$ at step $i$.
6:      Let to.traverse$[p'] \leftarrow$ false for all $p'$ traversed.
7:  end while
```

Fig.13. Repeated Traversing in GDS.

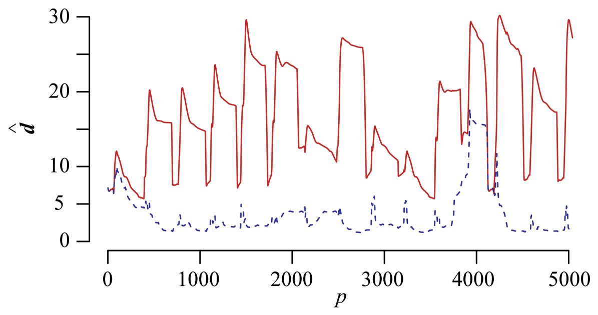

Fig.14. Estimate $\hat{\boldsymbol{d}}$ resulted from the initialization step. The dashed line shows the true distance vector $\boldsymbol{d}$.

Fig.15 shows the positions sampled by the GDS algorithm. In contrast to Fig.10, the figure shows that Traversing is more heavily used than Differentiating in GDS.

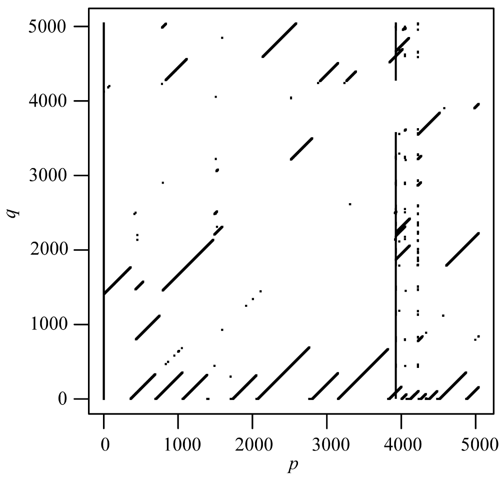

Fig.15. Positions of the distance matrix in Fig.2 sampled by the GDS algorithm. In total 20 531 positions have been sampled.

3.4 Performance Analysis

The initialization step of GDS needs M distance queries for generating the distance profile at p_0. The verification step needs M additional distance queries to compute the exact value of $d[p^*]$ (unless p_0 is a discord location). Therefore GDS needs at least $2M$ distance queries. In the best case, p_0 is a discord location and M further queries in the refinement step verify that $\max_p \hat{d}[p] = d[p_0]$. In this case, the discord location p_0 can be found with only $2M$ distance queries.

Clearly $\binom{M}{2}$ is an upper bound on the number of distance queries for GDS. As shown in the next section, in practice GDS often needs fewer than $10M$ distance queries.

4 Empirical Evaluation

In this section, a collection of time series is used to compare the efficiency of two direct discord search methods: the PBDS algorithm in [19] and the GDS algorithm in this paper. Following the convention in discord search, the number of distance queries is used to measure the efficiency of the algorithms. Two sets of experiments are performed. In the first set, the discord length n is fixed and the time series length m varies. In the second set, a set of time series is used with varying discord length n. Fig.16 shows a sample of the time series used in the two sets of experiments.

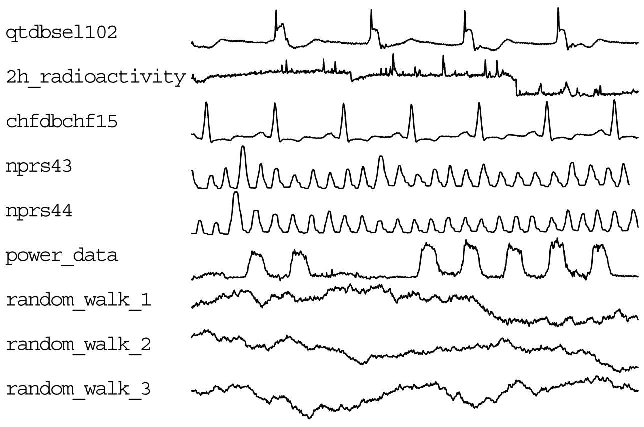

Fig.16. Time series used in empirical evaluation. Only the first 1 000 values of each time series are plotted.

In the first set of experiments, time series of varying lengths are randomly generated from a long time series `qtdbsel102` (as in [26]). For each time series length m, 100 random excerpts of `qtdbsel102` are created. The discord length n is fixed to 128. The same time series are analyzed using both PBGS and GDS.

As shown in Table 1, GDS uses fewer distance queries than PBGS and the end-to-end running time is shorter. Moreover GDS scales better with increasing m. Finally, GDS has stable performance over random excerpts of time series; compared to PBDS, GDS has a smaller variance in the number of distance queries and running time.

For the complete time series `qtdbsel102`, the resulting discord of length 128 is shown in Fig.17. Note that

Table 1. Comparison Results for Discord Search on Random Excerpts of Different Length m from Time Series `qtdbsel102`

Time Series Length	Number of Distance Queries (Standard Error)*		End-to-End Running Time in Seconds (Standard Error)*,**	
	PBDS	GDS	PBDS	GDS
1 000	4 020 (1 441)	3 311 (531)	0.5 (0.2)	0.4 (0.1)
2 000	11 159 (4 641)	8 071 (1 178)	1.7 (0.9)	1.2 (0.2)
4 000	30 938 (12 473)	18 293 (3 707)	7.5 (6.1)	3.4 (0.7)
8 000	77 381 (33 064)	38 900 (7 270)	37.4 (39.1)	9.9 (2.2)
16 000	168 277 (70 071)	79 086 (14 634)	198.9 (108.2)	36.5 (6.6)
32 000	365 900 (184 540)	159 486 (32 532)	556.0 (157.6)	131.7 (62.1)

Note: *The standard error of a sample is defined to be $\frac{s}{\sqrt{n}}$, where s is the sample standard deviation and n is the sample size.
**Experiments ran on a desktop PC with 3.40 GHz CPU and the program was implemented with the interpreted language `R`.

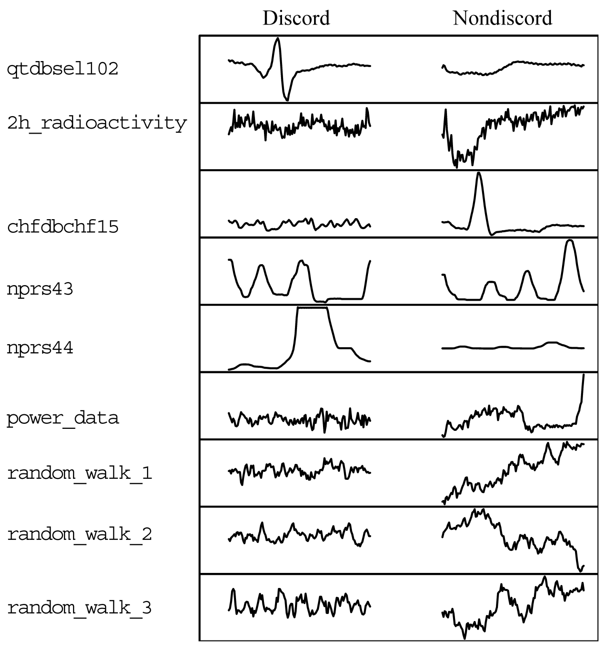

Fig.17. Length-128 discords of time series in Fig.16. For comparison, the initial segment of length 128 for each complete time series is also displayed.

the correctness of the algorithm is guaranteed by Proposition 2.

In the second set of experiments, a collection of time series from both [26] and [27] is used: `2h_radioactivity`, `chfdchf15`, `nprs43`, `nprs44`, `power_data`. (For `nprs43` and `nprs44`, the leading and trailing segments display rapid changes most likely introduced during data collection. Therefore, we remove 20 points at the beginning and the end from both time series.) For large m, we also generate three random walk time series `random_walk_1`, `random_walk_2`, and `random_walk_3` using the model $T[p] = \sum_{i=1}^{p} Z_i$, where each Z_i is an independent random variable with the standard normal distribution (with mean 0 and variance 1). For each time series, discords of lengths $n = 64$, 128, and 256 are searched for using both PBDS and GDS. The resulting discords for length 128 are displayed in Fig.17.

As shown in Table 2, PBDS's efficiency degrades as the discord length n is reduced. This problem is most severe when the time series is non-periodic, as for the three random walk series. In contrast, GDS's efficiency is not affected by the change in n — GDS displays consistent performance advantage over PBDS. With random walk time series and a high m/n ratio, GDS often makes only 5% of the distance queries needed by PBDS.

Table 2. Comparison on the Number of Distance Queries for Varying Discord Length n

Time Series	Length	Number of Distance Queries (GDS/PBDS) $n = 64$	$n = 128$	$n = 256$
`2h_radioactivity`	4 370	32 425 /226 729	34 326 /149 098	46 702 /87 560
`chfdbchf15`	15 000	54 166 /87 890	58 546 /83 318	63 922 /68 338
`nprs43`	18 012	80 937 /301 084	86 966 /309 147	92 581 /235 725
`nprs44`	24 085	108 152 /475 035	152 097 /599 344	129 892 /269 660
`power_data`	35 000	218 388 /2 407 453	226 500 /1 074 393	176 907 /239 122
`random_walk_1`	64 000	281 388 /13 077 385	436 323 /8 500 259	1 000 761 /1 318 399
`random_walk_2`	128 000	910 864 /61 675 743	868 648 /24 527 477	1 021 691 /10 949 911
`random_walk_3`	256 000	2 088 612 /40 867 969	1 269 231 /66 813 625	2 030 416 /44 377 293

5 Conclusions

We have proposed a general discord search (GDS) algorithm that is free of parameters. Empirical evaluation shows that the GDS algorithm finds the same discords with considerably fewer distance queries than the existing direct search algorithm PBDS. As shown in [19], direct discord search is faster than index-based discord search. The GDS algorithm demonstrates even greater search efficiency. Finally, the GDS algorithm has stable performance across a wide range of time series lengths and discord lengths. This robustness is another benefit of eliminating tuning parameters.

We believe that the simplicity and speed advantage of the GDS algorithm will further the adoption of time series anomaly mining in a broader range of applications, as they allow quick search of anomalous subsequences and evaluation of their relevance. For example in many pattern recognition applications involving time series (such as seizure detection with epilepsy data[14]), anomalous subsequences are important features for the prediction systems. As the GDS algorithm provides a robust way to quickly extract discords of different lengths, it can be a new feature extraction tool for pattern recognition practitioners.

The structure of the GDS algorithm suggests a connection between discord search and other dimensionality reduction techniques such as PCA. In particular, the reference function can potentially be replaced by other low dimensional representations of subsequences (e.g., the first principal scores). We are currently examining the connection and its use in discord-search algorithm design.

References

[1] Shoeb A, Edwards H, Connolly J *et al.* Patient-specific seizure onset detection. *Epilepsy & Behavior*, 2004, 5(4): 483-498.

[2] Febrero M, Galeano P, González-Manteiga W. A functional analysis of NOx levels: Location and scale estimation and outlier detection. *Computational Statistics*, 2007, 22(3): 411-427.

[3] van Wijk J, Van Selow E. Cluster and calendar based visualization of time series data. In *Proc. the 1999 IEEE Symposium on Information Visualization*, Oct. 1999, pp.4-9.

[4] Rebbapragada U, Protopapas P, Brodley C, Alcock C. Finding anomalous periodic time series. *Machine learning*, 2009, 74(3): 281-313.

[5] Rousseeuw P, Leroy A. Robust Regression and Outlier Detection. Wiley Online Library, 1987.

[6] Dasgupta D, Forrest S. Novelty detection in time series data using ideas from immunology. In *Proc. the International Conference on Intelligent Systems*, Dec. 1996, pp.82-87.

[7] Hyndman R, Ullah S. Robust forecasting of mortality and fertility rates: A functional data approach. *Computational Statistics & Data Analysis*, 2007, 51(10): 4942-4956.

[8] Febrero M, Galeano P, González-Manteiga W. Outlier detection in functional data by depth measures, with application to identify abnormal NO_X levels. *Environmetrics*, 2008, 19(4): 331-345.

[9] Keogh E, Lin J, Fu A. HOT SAX: Efficiently finding the most unusual time series subsequence. In *Proc. the 5th IEEE International Conference on Data Mining*, Nov. 2005, pp.226-233.

[10] Lin J, Keogh E, Fu A, Van Herle H. Approximations to magic: Finding unusual medical time series. In *Proc. the 18th IEEE Symposium on Computer-Based Medical Systems*, Jun. 2005, pp.329-334.

[11] Bu Y, Leung T, Fu A, Keogh E, Pei J, Meshkin S. WAT: Finding top-K discords in time series database. In *Proc. the 7th SIAM International Conference on Data Mining*, Apr. 2007.

[12] Yankov D, Keogh E, Rebbapragada U. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. *Knowledge and Information Systems*, 2008, 17(2): 241-262.

[13] Goldberger A, Amaral L, Glass L *et al.* Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 2000, 101(23): e215-e220.

[14] Shoeb A, Guttag J. Application of machine learning to epileptic seizure detection. In *Proc. the 27th International Conference on Machine Learning*, Jun. 2010, pp.975-982.

[15] Pham N, Le Q, Dang T. HOT aSAX: A novel adaptive symbolic representation for time series discords discovery. In *Proc. the 2nd Intelligent Information and Database Systems*, Mar. 2010, pp.113-121.

[16] Fu A, Leung O, Keogh E, Lin J. Finding time series discords based on Haar transform. In *Proc. the 2nd Advanced Data Mining and Applications*, Aug. 2006, pp.31-41.

[17] Shieh J, Keogh E. *i*SAX: Indexing and mining terabyte sized time series. In *Proc. the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2008, pp.623-631.

[18] Son M, Anh D. EWAT+: Finding time series discords based on new discord measure functions. In *Proc. IEEE International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future*, Nov. 2010, pp.1-4.

[19] Luo W, Gallagher M. Faster and parameter-free discord search in quasi-periodic time series. In *Proc. PAKDD*, May 2011, pp.135-148.

[20] Eckmann J, Kamphorst S, Ruelle D. Recurrence plots of dynamical systems. *Europhysics Letters*, 1987, 4(9): 973-977.

[21] Iwanski J, Bradley E. Recurrence plots of experimental data: To embed or not to embed? *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 1998, 8(4): 861-871.

[22] Webber Jr C, Zbilut J. Recurrence quantification analysis of nonlinear dynamical systems. *Tutorials in Contemporary Nonlinear Methods for the Behavioral Sciences*, 2005, pp.26-94.

[23] Marwan N, Carmen Romano M C, Thiel M, Kurths J. Recurrence plots for the analysis of complex systems. *Physics Reports*, 2007, 438(5/6): 237-329.

[24] Marwan N. A historical review of recurrence plots. *The European Physical Journal-Special Topics*, 2008, 164(1): 3-12.

[25] Chatfield C. The Analysis of Time Series: An Introduction, (6th edition). Chapman & Hall/CRC, 2004.

[26] Keogh E, Lin J, Fu A. The UCR time series discords. http://www.cs.ucr.edu/∼eamonn/discords/, April 2010.

[27] Hyndman R. Time series data library. http://data.is/TSDLdemo, April 2010.

Wei Luo holds a Ph.D. degree in computing science from Simon Fraser University and is currently a research fellow in Centre for Pattern Recognition and Data Analytics at Deakin University. He has years of experience in health care data mining, having completed a number of projects in patient flow optimization and harm reduction. His research interests include temporal data mining, machine learning, data quality, and visual analytics.

Marcus Gallagher received the B.Comp.Sc. and Grad.Dip.Sc. degrees from the University of New England, Armidale, Australia, in 1994 and 1995, respectively, and the Ph.D. degree in computer science from the University of Queensland, Brisbane, Australia, in 2000. He is a senior lecturer in the Complex and Intelligent Systems Research Group at the School of Information Technology and Electrical Engineering, University of Queensland. His main research interests are metaheuristic optimization and machine learning algorithms, in particular techniques based on statistical modeling. He is also interested in biologically inspired algorithms, methodology for empirical evaluation of algorithms, and the visualization of high-dimensional data.

Janet Wiles holds a Ph.D. degree in computer science from the University of Sydney, and is a professor of complex and intelligent systems at the University of Queensland. She recently completed a five-year project leading the Thinking Systems Project, supervising a cross-disciplinary team studying fundamental issues in how information is transmitted, received, processed and understood in biological and artificial systems. Her research interests include complex systems biology, computational neuroscience, computational modeling methods, artificial intelligence and artificial life, language and cognition.