Advances in Adaptive Data Analysis
Vol. 3, Nos. 1 & 2 (2011) 167–186

DOI: 10.1142/S1793536911000775

World Scientific
www.worldscientific.com

A FAST ALGORITHM FOR COMPUTING SAMPLE ENTROPY*

YING JIANG[†,¶], DONG MAO[‡,‖] and YUESHENG XU[§,**]

[†]*Guangdong Province Key Lab of Computational Science,*
Sun Yat-Sen University,
Guangzhou 510275, P. R. China

[‡]*Department of Mathematics,*
Michigan State University,
East Lansing, MI 48824, USA

[§]*Department of Mathematics,*
Syracuse University,
Syracuse, NY 13244, USA
[¶]*yjiang80@gmail.com*
[‖]*dmao@math.msu.edu*
[**]*yxu06@syr.edu*

Sample entropy is a widely used tool for quantifying complexity of a biological system. Computing sample entropy directly using its definition requires large computational costs. We propose a fast algorithm based on a k-d tree data structure for computing sample entropy. We prove that the time complexity of the proposed algorithm is $O(N^{2-\frac{1}{m+1}})$ and its space complexity is $O(N \log N)$, where N is the length of the input time series and m is the length of its pattern templates. We present a numerical experiment that demonstrates significant improvement of the proposed algorithm in computing time.

Keywords: Fast algorithm; sample entropy; k-d tree.

1. Introduction

Sample entropy is often applied to quantify the complexity of a physiologic system. Physiologic systems are usually regulated by interacting mechanisms. It is believed [Costa *et al.* (2005)] that the complexity of time series derived from physiologic systems can provide information of the underlying interacting mechanisms. Recently, entropy was used to quantify complexity. Classical entropies such as the Shannon entropy and the Kolmogorov–Sinai entropy of a dynamic system characterize the rate of generating new information. However, these entropies are not practical in applications since they require an infinite data series with

*This research is partially supported by Guangdong Provincial Government of China through "the Computational Science Innovative Research Team" program.
**Corresponding author.

infinitely accurate precision and resolution. In this regard, approximate entropy [Pincus (1991)] was instead used to quantify complexity, while sample entropy as an improvement of approximate entropy, which reduces the bias of approximate entropy caused by including self-matches, was proposed in [Richman and Moorman (2000)]. In terms of sample entropy, multiscale sample entropy was proposed in [Costa *et al.* (2002a,b); Costa and Healey (2003)] to quantify complexity of a time series on different scales. This family of entropies is also used in studying electroencephalograph [Abasolo *et al.* (2005)], renal sympathetic nerve activity [Zhang *et al.* (2007)], and many others [Kozuch and Wang (2001); Pincus and Kalman (2004)].

Calculating sample entropy of a given time series by directly using its definition (we call it the direct algorithm in this paper) involves counting the number of its matched pairs. It requires execution time proportional to the square of the length of the input time series. Under the circumstance that an instantaneous result of sample entropy of a long time series is required, a fast algorithm for computing sample entropy is highly desirable. Along this direction, a bucket-assisted algorithm was proposed in [Manis (2008)] intending to speed up the calculation of approximate entropy. However, the algorithm still requires $O(N^2)$ execution time where N denotes the length of a time series.

The purpose of this paper is to present a fast algorithm for computing sample entropy by exploring a special data structure. Specifically, we employ a k-d tree to accelerate counting the number of matched pairs of the pattern emplates in a times series, which leads to a fast algorithm for computing sample entropy. We prove that the fast algorithm uses $O(N^{2-\frac{1}{m+1}})$ execution time where m is the number of pattern templates of the time series. While applying the proposed fast algorithm to heart-beat interval time series, numerical experiments exhibit a significant improvement on the execution time, which is $O(N^{1.18})$ on average. When $N = 64,000$, numerical experiments show that the execution time for the fast algorithm is only 2.6% of that for the direct algorithm.

The structure of this paper is as follows. In Sec. 2, we review the concept of sample entropy and the direct algorithm for calculating sample entropy. We also estimate the complexity of the direct algorithm. We recall in Sec. 3 the concept of k-d trees and propose a fast algorithm for calculating sample entropy based on k-d trees. We provide in Sec. 4, the complexity analysis for the fast algorithm proposed. In Sec. 5, we apply the fast algorithm to heart-beat interval time series and compare its numerical performance with that of the direct algorithm. Conclusive remarks are presented in Sec. 6. In the appendix, we include several lemmas and a proposition necessary for the development of the fast algorithm.

2. The Direct Algorithm

We first review the definition of sample entropy (SampEn) and the direct algorithm for calculating it. We begin with recalling the definition of sample entropy. Let $\mathbb{N}$ denote the set of all natural numbers. For $N \in \mathbb{N}$, we let $\mathbb{Z}_N := \{1, 2, \ldots, N\}$ and

we use $\mathbf{x} := [x_i : i \in \mathbb{Z}_N] \in \mathbb{R}^N$ to denote a time series. For all $m \in \mathbb{Z}_{N-1}$ and $i \in \mathbb{Z}_{N-m+1}$, we define

$$\mathbf{x}_{m,i} := [x_{i+j-1} : j \in \mathbb{Z}_m]. \tag{1}$$

It is clear that for each $i \in \mathbb{Z}_{N-m+1}$, $\mathbf{x}_{m,i}$ represents a subsequence in $\mathbf{x}$ of m consecutive terms. We call $\mathbf{x}_{m,i}$ a pattern template of the time series and m the length of the pattern template.

For all $m \in \mathbb{N}$ and $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$, the distance of two vectors $\mathbf{u} := [u_i : i \in \mathbb{Z}_m]$ and $\mathbf{v} := [v_i : i \in \mathbb{Z}_m]$ is defined by

$$d(\mathbf{u}, \mathbf{v}) := \max\{|u_i - v_i| : i \in \mathbb{Z}_m\}. \tag{2}$$

For all $\mathbf{x} \in \mathbb{R}^N$, $m \in \mathbb{Z}_{N-1}$, and all $i, j \in \mathbb{Z}_{N-m+1}$, we call $\{\mathbf{x}_{m,i}, \mathbf{x}_{m,j}\}$ a matched pair if

$$d(\mathbf{x}_{m,i}, \mathbf{x}_{m,j}) \le r\sigma(\mathbf{x}), \quad \text{for a positive real number } \ r,$$

where $\sigma(\mathbf{x})$ represents the sample standard deviation of $\mathbf{x}$. The number r is called the *tolerance for accepting matches*. For all $\mathbf{x} \in \mathbb{R}^N$, $m \in \mathbb{Z}_{N-1}$, and $i \in \mathbb{Z}_{N-m}$, we define

$$\begin{aligned} A_r(\mathbf{x}, m, i) := {} & \text{the number of } \mathbf{x}_{m,j} \text{ such that } \{\mathbf{x}_{m,i}, \mathbf{x}_{m,j}\} \text{ are matched pairs,} \\ & \text{for all } j \in \mathbb{Z}_{N-m} \backslash \mathbb{Z}_i, \end{aligned} \tag{3}$$

and

$$\begin{aligned} B_r(\mathbf{x}, m, i) := {} & \text{the number of } \mathbf{x}_{m+1,j} \text{ such that } \{\mathbf{x}_{m+1,i}, \mathbf{x}_{m+1,j}\} \\ & \text{are matched pairs,} \quad \text{for all } j \in \mathbb{Z}_{N-m} \backslash \mathbb{Z}_i. \end{aligned} \tag{4}$$

For all $\mathbf{x} \in \mathbb{R}^N$, $m \in \mathbb{Z}_{N-1}$, let

$$A_r(\mathbf{x}, m) := \sum_{i \in \mathbb{Z}_{N-m}} A_r(\mathbf{x}, m, i), \tag{5}$$

$$B_r(\mathbf{x}, m) := \sum_{i \in \mathbb{Z}_{N-m}} B_r(\mathbf{x}, m, i). \tag{6}$$

With these notations, sample entropy of a time series $\mathbf{x} \in \mathbb{R}^{\mathbf{N}}$ for all $m \in \mathbb{Z}_{N-2}$ and $r > 0$ is defined by

$$\mathrm{SampEn}(\mathbf{x}, m, r) := \begin{cases} \log((N-m)(N-m-1)), & \text{if } A_r(\mathbf{x}, m) B_r(\mathbf{x}, m) = 0, \\ -\log \dfrac{B_r(\mathbf{x}, m)}{A_r(\mathbf{x}, m)}, & \text{otherwise.} \end{cases} \tag{7}$$

We next describe the direct algorithm based on definition (7) for calculating SampEn.

Algorithm 2.1. Direct algorithm for calculating SampEn.
Input: *A time series* $\mathbf{x}$ *of length* N, *a template length* $m \in \mathbb{Z}_{N-2}$, *and a tolerance* $r > 0$.
Output: *Sample entropy* $\mathrm{SampEn}(\mathbf{x}, m, r)$.

Step 1. *Compute the sample standard deviation* $\sigma(\mathbf{x})$ *of* $\mathbf{x}$.
Step 2. *For each* $i \in \mathbb{Z}_{N-m}$, *compute distances* $d(\mathbf{x}_{m,i}, \mathbf{x}_{m,j})$ *for all* $j \in \mathbb{Z}_{N-m}\backslash\mathbb{Z}_i$, *and count* $A_r(\mathbf{x}, m, i)$ *using Eq.* (3).
Step 3. *For each* $i \in \mathbb{Z}_{N-m}$, *compute distances* $d(\mathbf{x}_{m+1,i}, \mathbf{x}_{m+1,j})$ *for all* $j \in \mathbb{Z}_{N-m}\backslash\mathbb{Z}_i$, *and count* $B_r(\mathbf{x}, m, i)$ *using Eq.* (4).
Step 4. *Compute* $A_r(\mathbf{x}, m)$ *and* $B_r(\mathbf{x}, m)$ *using Eqs.* (5) *and* (6), *respectively.*
Step 5. *Compute* $\mathrm{SampEn}(\mathbf{x}, m, r)$ *using Eq.* (7).

We next prove that Algorithm 2.1 requires $O(N^2)$ time complexity to compute SampEn for a time series $\mathbf{x}$ of length N. We assume that m is fixed and it does not increase as N increases. In practical applications, the template length m usually takes a small integer such as $m = 2$ or $m = 3$, refer to [Al-Angari and Sahakian (2007); Costa *et al.* (2002a,b,2005); Lake *et al.* (2002); Richman and Moorman (2000)].

Proposition 2.2. *If* $\mathbf{x}$ *is a time series of length* N, *then the time complexity and the space complexity of Algorithm* 2.1 *are* $O(N^2)$ *and* $O(N)$, *respectively.*

Proof. We observe that in step 1 of Algorithm 2.1, for a time series $\mathbf{x}$ of length N, computing the sample standard deviation $\sigma(\mathbf{x})$ requires $O(N)$ multiplications and additions.

In step 2, for each i with $i \in \mathbb{Z}_{N-m}$, and each j such that $j \in \mathbb{Z}_{N-m}\backslash\mathbb{Z}_i$, Algorithm 2.1 requires m subtractions to compute $d(\mathbf{x}_{m,i}, \mathbf{x}_{m,j})$ and m comparisons. Hence, by the definition of $A_r(\mathbf{x}, m, i)$, we see that Algorithm 2.1 executes $N-m-i$ basic operations such as subtraction and comparison to compute $A_r(\mathbf{x}, m, i)$. Noting that

$$\sum_{i\in\mathbb{Z}_{N-m}} (N-m-i) = \frac{(N-m)(N-m-1)}{2},$$

the total number of subtractions and comparisons for computing all $A_r(\mathbf{x}, m, i)$, $i \in \mathbb{Z}_{N-m}$, is $O(N^2)$.

Likewise, in step 3, computing $B_r(\mathbf{x}, m, i)$ defined by Eq. (4) for all $i \in \mathbb{Z}_{N-m}$ requires $O(N^2)$ operations. By the definitions Eq.(5) and Eq. (6), computing $A_r(\mathbf{x}, m)$ and $B_r(\mathbf{x}, m)$ in step 4 needs $O(N)$ operations. Note that operations needed in step 5 is $O(1)$.

Summing the number of operations needed in all steps, we obtain that the time complexity of Algorithm 2.1 is $O(N^2)$. The space complexity of Algorithm 2.1 is clearly $O(N)$. □

3. A Fast Algorithm

The purpose of this section is to develop a fast algorithm for calculating SampEn, which improves the time complexity of the direct algorithm. To accelerate the computation, we speed up the counting of the number of matched pairs of the input time

series by treating the subsequences derived from the input time series as the points in $\mathbb{R}^m$ and organizing them in a k-d tree data structure, which is widely used in computer science and computational geometry, see [Agarwal and Erickson (1999); Bentley (1980)].

A critical step in computing sample entropy for a given time series $\mathbf{x} \in \mathbb{R}^N$ is to calculate $A_r(\mathbf{x}, m, i)$ and $B_r(\mathbf{x}, m, i)$, $i \in \mathbb{Z}_{N-m}$. The proposed fast algorithm is based on an observation that finding the number of $\mathbf{x}_{m,j}$ for all $j \in \mathbb{Z}_{N-m} \backslash \mathbb{Z}_i$ matched with a given $\mathbf{x}_{m,i}$ is equivalent to counting the number of the points representing $\mathbf{x}_{m,j}$ falling into a cube in $\mathbb{R}^m$ centered at the point $\mathbf{x}_{m,i}$ with the edge length $2r\sigma(\mathbf{x})$. Specifically, computing $A_r(\mathbf{x}, m, i)$ is equivalent to calculating the number of the points that fall into a hypercube, in $\mathbb{R}^m$, centered at $\mathbf{x}_{m,i}$ with edge length being $2r\sigma(\mathbf{x})$, and computing $B_r(\mathbf{x}, m, i)$ is equivalent to calculating the number of the points that fall into a hypercube, in $\mathbb{R}^{m+1}$, centered at $\mathbf{x}_{m+1,i}$ with edge length being $2r\sigma(\mathbf{x})$. We call this counting process the range counting.

These points may be organized in a k-d tree, with which we may assign an "order" to all subsequences derived from the input time series. Such an "order" helps us determine whether or not a point is inside a hyperrectangle in $\mathbb{R}^m$. It accelerates various tasks, such as range counting, range reporting, and finding the nearest point. This order exists naturally in the 1-dimensional (1D) space $\mathbb{R}$. For example, for arbitrary real numbers a, b, and c with $a < b$ and $b < c$, it must be $a < c$. In higher-dimensional spaces, the k-d tree structure gives us a similar order which reduces the number of comparisons.

3.1. *A construction of k-d trees*

We describe in this section a construction of a k-d tree for a given *multiset* of points in a Euclidian space. The concept of multisets is a generalization of that of sets. Unlike a set, a multiset allows *non-distinct* members. The cardinality of a multiset is the number of its all members, with a repeated member counting the number of multiplicity. We refer to [Blizard (1989)] for more about the notion of multiset. Suppose that k and N are positive integers with $k \ll N$, and for $n \in \mathbb{N}$ a point multiset

$$S := \{\mathbf{x}_p := [x_{p,i} : i \in \mathbb{Z}_k] : p \in \mathbb{Z}_n\} \subset \mathbb{R}^k$$

is given. We construct a k-d tree for the multiset S. First of all, we assign the *level index* $\mu = 1$ to the multiset S. The ingredients of the construction include (1) specification of a node associated with the multiset and (2) a definition of rules that will be used to split the multiset.

We now consider the first ingredient — the node associated with the multiset S. For the given multiset S, we let $\mathcal{C} := \mathfrak{C}(S)$ denotes the cardinality of S, and let

$$u_{S,i} := \min\{x_{p,i} : p \in \mathbb{Z}_n\} \quad \text{and} \quad v_{S,i} := \max\{x_{p,i} : p \in \mathbb{Z}_n\}, \quad i \in \mathbb{Z}_k.$$

Note that when $S = \emptyset$, we set $u_{S,i} := 0$ and $v_{S,i} := 1$. We then define $\mathbf{u}_S := [u_{S,i} : i \in \mathbb{Z}_k]$ and $\mathbf{v}_S := [v_{S,i} : i \in \mathbb{Z}_k]$. Associated with the given multiset S, there is a unique node which is the triple $\mathbf{n}_S := \{\mathcal{C}, \mathbf{u}_S, \mathbf{v}_S\}$.

We turn to describing the second ingredient — the splitting rule. We employ the *midpoint* splitting rule. For given point multiset $S \subset \mathbb{R}^k$ with the level index $\mu \in \mathbb{Z}_k$

- find the median $m_{S,\mu}$ of the values of the μth coordinate of all points in S and
- split S into two disjoint submultisets S' and S'' such that the values of the μth coordinates of the points in S' are less than $m_{S,\mu}$, and those of the μth coordinates of the points in S'' are not less than $m_{S,\mu}$. Assign to S' and S'' the level index $(\mu \bmod k)+1$.

With the above ingredients, we now construct the k-d tree of the finite point multiset $S \subset \mathbb{R}^k$. We assign to S the level index $\mu = 1$ and the k-d tree is constructed recursively. The root node of the k-d tree of S is the node associated with the multiset S. By using the splitting rules described above, we split multiset S into two disjoint submultisets S' and S'', both having the level index $\mu = 2$. The two child nodes (of the root node) in the k-d tree are the nodes associated with the multisets S' and S'', respectively. This process is repeated until the submultisets cannot be further split. We summarize in the following algorithm the construction of the k-d tree of a given finite point multiset.

Algorithm 3.1. Algorithm for generating the k-d tree.
Input: *A finite point multiset $S \subset \mathbb{R}^k$ with the level index $\mu = 1$.*
Output: *The k-d tree T of S.*

Step 1. *Generate node $\mathfrak{N}$ from S.*
Step 2. *If $\mathbf{u}_S \neq \mathbf{v}_S$, then split S into two disjoint submultisets S' and S'' by the midpoint split rule. Otherwise, go to step* 4.
Step 3. *Generate nodes $\mathfrak{N}'$ and $\mathfrak{N}''$ from S' and S'', respectively, and assign them as two child nodes of $\mathfrak{N}$. Let $S := S'$, S'', respectively, assign them the level index $\mu := (\mu \bmod k) + 1$, and go to step* 1.
Step 4. *Stop and output the k-d tree of S.*

We now illustrate the algorithm by an example with $k = 2$. Suppose that S_1 is a multiset of n points in $\mathbb{R}^2$. We assign S_1 the level index $\mu = 1$. We count the cardinality of S_1, and generate vectors $\mathbf{u}_{S_1}$ and $\mathbf{v}_{S_1}$. In this way, we have constructed the node $\mathfrak{N}_1$ associated with S_1. It serves as the root node of the k-d tree. If $\mathbf{u}_{S_1} = \mathbf{v}_{S_1}$, then we do not split multiset S_1 and the k-d tree of S_1 has only one node $\mathfrak{N}_1$. If $\mathbf{u}_{S_1} \neq \mathbf{v}_{S_1}$, we then split the multiset S_1 into two submultisets S_2 and S_3 by the midpoint splitting rule, and assign both of them the level index $\mu = 2$. We count the cardinality of S_2 and S_3 and generate vectors $\mathbf{u}_{S_2}$, $\mathbf{v}_{S_2}$, $\mathbf{u}_{S_3}$, and $\mathbf{v}_{S_3}$ so that the nodes $\mathfrak{N}_2$ and $\mathfrak{N}_3$ associated with S_2 and S_3, respectively, are constructed. Nodes $\mathfrak{N}_2$ and $\mathfrak{N}_3$ are the two child nodes of $\mathfrak{N}_1$. If $\mathbf{u}_{S_2} = \mathbf{v}_{S_2}$, we do not split

S_2 further and otherwise, we split the multiset S_2 into two submultisets by the midpoint split rule. Likewise, if $\mathbf{u}_{S_3} = \mathbf{v}_{S_3}$, we do not split S_3 and otherwise, we split the multiset S_3 into two submultisets by the midpoint split rule. We repeat this process until it terminates. The nodes associated with all resulting submultisets form the k-d tree of S_1. Figure 1 illustrates the procedure of splitting for the point multiset $S_1 := \{a, b, c, d, e, f, g\}$ as shown in part (a).

3.2. *A counting algorithm*

For a given time series, the k-d tree of the multiset associated with it allows us to speed up the range-counting process, which is crucial for computing its sample entropy. Suppose that S is a multiset of finite points in $\mathbb{R}^k$ and T is its k-d tree. For a given cube $R \subseteq \mathbb{R}^k$, we need to count the cardinality of the multiset $S \cap R$. This can be efficiently done by using the k-d tree of S. We present in this section an algorithm based on the k-d tree T of S to count this number.

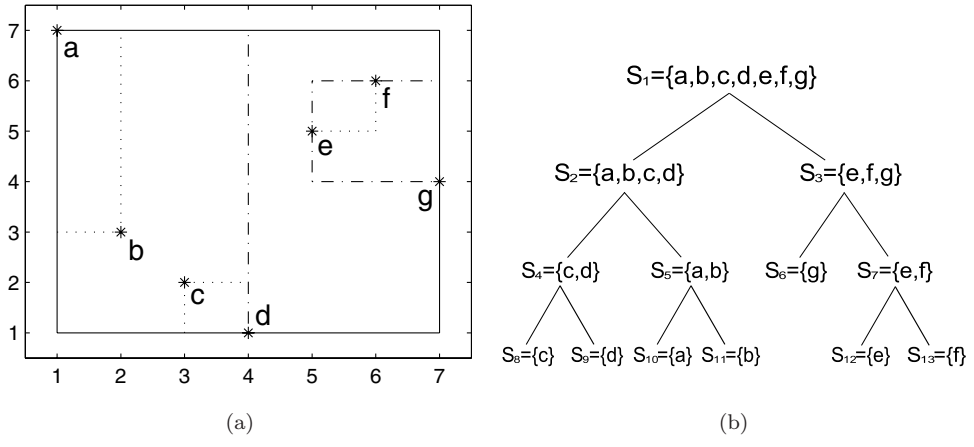
We first define a counting function. For each node $\mathfrak{N}$ of T, we define the rectangle

$$\mathcal{B}_{\mathfrak{N}} := \{\mathbf{x} := [x_i : i \in \mathbb{Z}_k] \in \mathbb{R}^k : u_i \le x_i \le v_i, i \in \mathbb{Z}_k\},$$

where vectors $\mathbf{u} := [u_i : i \in \mathbb{Z}_k]$ and $\mathbf{v} := [v_i : i \in \mathbb{Z}_k]$ are the vectors in node $\mathfrak{N}$. We then define the *counting* function for a cube $R \subset \mathbb{R}^k$ and a node $\mathfrak{N}$ of T by

$$F(\mathfrak{N}, R) := \begin{cases} 0, & \text{if } \mathcal{B}_{\mathfrak{N}} \cap R = \emptyset, \\ \mathcal{C}, & \text{if } \mathcal{B}_{\mathfrak{N}} \cap R = \mathcal{B}_{\mathfrak{N}}, \\ F(\mathfrak{N}', R) + F(\mathfrak{N}'', R), & \text{otherwise}, \end{cases} \tag{8}$$

where $\mathfrak{N}'$ and $\mathfrak{N}''$ are two child nodes of $\mathfrak{N}$, and $\mathcal{C}$ is the nonnegative integer in $\mathfrak{N}$. Note that the counting function is defined recursively. The fact that it is well



(a) (b)

Fig. 1. Part (a) illustrates the partition of a multiset S_1. Part (b) shows all submultisets generated during the procedure of constructing the k-d tree of S_1.

defined is proved in Corollary A.4. It is proved in Proposition A.6 that

$$F(\mathfrak{N}, R) = \mathfrak{C}(S \cap R).$$

The range counting for S with the k-d tree is a recursive process, which runs the counting function. The order introduced by the k-d tree speeds up the range-counting process. We next present the algorithm with which we obtain the counting number.

Algorithm 3.2. Range counting.
Input: *A k-d tree T of S with root node $\mathfrak{N}_r$ and a cube $R \in \mathbb{R}^k$.*
Output: *The counting number C.*

- *Compute $F(\mathfrak{N}_r, R)$.*
- *Output $C := F(\mathfrak{N}_r, R)$.*

Algorithm 3.2 allows us to design a fast algorithm for computing sample entropy.

3.3. *A fast algorithm based on k-d trees*

Based on Algorithm 3.2, we develop in this section a fast algorithm for calculating $A_r(\mathbf{x}, m)$ and $B_r(\mathbf{x}, m)$.

We rewrite $A_r(\mathbf{x}, m)$ and $B_r(\mathbf{x}, m)$ in convenient alternative forms. Recall that m is the template length and r is the tolerance. For a given time series $\mathbf{x} := [x_i : i \in \mathbb{Z}_N]$, we define

$$S_{m,\mathbf{x}} := \{\mathbf{x}_{m,i} : i \in \mathbb{Z}_{N-m}\} \quad \text{and} \quad S^+_{m,\mathbf{x}} := \{\mathbf{x}_{m+1,i} : i \in \mathbb{Z}_{N-m}\},$$

where $\mathbf{x}_{m,i}$ and $\mathbf{x}_{m+1,i}$ are defined by Eq. (1). For each $i \in \mathbb{Z}_{N-m}$, we define two sets

$$R_{m,i} := \{\mathbf{y} \in \mathbb{R}^m : d(\mathbf{y}, \mathbf{x}_{m,i}) \le r\sigma(\mathbf{x})\}$$

and

$$R^+_{m,i} := \{\mathbf{y} \in \mathbb{R}^{m+1} : d(\mathbf{y}, \mathbf{x}_{m+1,i}) \le r\sigma(\mathbf{x})\}.$$

For each $\mathbf{x} \in \mathbb{R}^N$, $m \in \mathbb{Z}_{N-2}$ and $i \in \mathbb{Z}_{N-m}$, we let

$$C_r(\mathbf{x}, m, i) := \mathfrak{C}(S_{m,\mathbf{x}} \cap R_{m,i})$$

and

$$C^+_r(\mathbf{x}, m, i) := \mathfrak{C}(S^+_{m,\mathbf{x}} \cap R^+_{m,i}).$$

It is clear that $C_r(\mathbf{x}, m, i)$ is the number of points in $S_{m,\mathbf{x}}$ such that $d(\mathbf{x}_{m,i}, \mathbf{x}_{m,j}) \le r\sigma(\mathbf{x})$, and $C^+_r(\mathbf{x}, m, i)$ is the number of points in $S^+_{m,\mathbf{x}}$ such that

$d(\mathbf{x}_{m+1,i}, \mathbf{x}_{m+1,j}) \le r\sigma(\mathbf{x})$. We also define for all $\mathbf{x} \in \mathbb{R}^N$, $m \in \mathbb{Z}_{N-2}$

$$C_r(\mathbf{x}, m) := \sum_{i \in \mathbb{Z}_{N-m}} C_r(\mathbf{x}, m, i), \tag{9}$$

$$C_r^+(\mathbf{x}, m) := \sum_{i \in \mathbb{Z}_{N-m}} C_r^+(\mathbf{x}, m, i). \tag{10}$$

If $\{\mathbf{x}_{m,i}, \mathbf{x}_{m,j}\}$ is a matched pair with $i \neq j$, i.e., $\mathbf{x}_{m,i}$ and $\mathbf{x}_{m,j}$ are within the distance $r\sigma(\mathbf{x})$, then this pair is counted twice when calculating $C_r(\mathbf{x}, m)$. On the other hand, the self-matched pair $\{\mathbf{x}_{m,i}, \mathbf{x}_{m,i}\}$ is counted once in $C_r(\mathbf{x}, m)$. By this observation and by Eq. (5), we reexpress $A_r(\mathbf{x}, m)$ in terms of $C_r(\mathbf{x}, m)$ as

$$A_r(\mathbf{x}, m) = \frac{1}{2}\left(C_r(\mathbf{x}, m) - N + m\right). \tag{11}$$

Likewise, by Eqs. (6) and (10), we reexpress $B_r(\mathbf{x}, m)$ in terms of $C_r^+(\mathbf{x}, m)$ as

$$B_r(\mathbf{x}, m) = \frac{1}{2}\left(C_r^+(\mathbf{x}, m) - N + m\right). \tag{12}$$

Formulas (11) and (12) provide alternative ways to compute sample entropy. They will lead to a fast algorithm for computing sample entropy.

According to Eqs. (11) and (12), we may use Algorithm 3.2 to count the number of matched pairs among pattern templates of a given time series. The next algorithm is designed for this purpose.

Algorithm 3.3. Counting matched pairs.
Input: *Positive integers n and k, a point multiset $S = \{\mathbf{x}_i := [x_{i,l} : l \in \mathbb{Z}_k] : i \in \mathbb{Z}_n\} \subset \mathbb{R}^k$ and a tolerance d.*
Output: *The number C of matched pairs.*

Step 1. *Construct the k-d tree T of S.*
Step 2. *For each $i \in \mathbb{Z}_n$, construct the rectangle*

$$R_i := \prod_{l=1}^{k} [x_{i,l} - d, x_{i,l} + d].$$

Step 3. *For each $i \in \mathbb{Z}_n$, run Algorithm 3.2 with the input T and R_i and obtain the output C_i.*
Step 4. *Compute*

$$C := \frac{1}{2}\left(\sum_{i \in \mathbb{Z}_n} C_i - n\right).$$

Algorithm 3.3 is now used to compute $A_r(\mathbf{x}, m)$ and $B_r(\mathbf{x}, m)$, and then sample entropy of $\mathbf{x}$. Specifically, we compute $A_r(\mathbf{x}, m)$ by Algorithm 3.3 with inputs $n := N - m$, $k := m$, $S := S_{m,\mathbf{x}}$ and $d := r\sigma(\mathbf{x})$. Likewise, we compute $B_r(\mathbf{x}, m)$ by Algorithm 3.3 with inputs $n := N - m$, $k := m + 1$, $S := S_{m,\mathbf{x}}^+$ and $d := r\sigma(\mathbf{x})$. The

fast algorithm for calculating sample entropy based on the k-d tree is now described below.

Algorithm 3.4. Fast algorithm for calculating SampEn.
Input: *A time series* $\mathbf{x} = [x_i : i \in \mathbb{Z}_N]$, *a template length* $m \in \mathbb{Z}_{N-2}$ *and a tolerance* r.
Output: *The sample entropy* $\mathrm{SampEn}(\mathbf{x}, m, r)$.

Step 1. *Generate* $S_{m,\mathbf{x}} = \{\mathbf{x}_{m,i} : i \in \mathbb{Z}_{N-m}\}$ *and* $S^+_{m,\mathbf{x}} := \{\mathbf{x}_{m+1,i} : i \in \mathbb{Z}_{N-m}\}$ *where* $\mathbf{x}_{m,i}$ *and* $\mathbf{x}_{m+1,i}$ *are defined by Eq.* (1). *Compute* $\sigma(\mathbf{x})$.
Step 2. *Compute* $A_r(\mathbf{x}, m)$ *by Algorithm* 3.3 *with inputs* $n := N - m$. $k := m$, $S := S_{m,\mathbf{x}}$ *and* $d := r\sigma(\mathbf{x})$.
Step 3. *Compute* $B_r(\mathbf{x}, m)$ *by Algorithm* 3.3 *with inputs* $n := N - m$, $k := m + 1$, $S := S^+_{m,\mathbf{x}}$ *and* $d := r\sigma(\mathbf{x})$.
Step 4. *Compute* $\mathrm{SampEn}(\mathbf{x}, m, r)$ *by Eq.* (7).

Algorithm 3.4 is a fast algorithm for computing sample entropy of a time series. In the next section, we provide complexity analysis for the algorithm.

4. Complexity Analysis

This section is devoted to a complexity analysis of Algorithm 3.4 for computing sample entropy of a time series.

The algorithm complexity considered here consists of the time complexity and the space complexity. The time complexity of an algorithm is measured by the total number of all elementary operations involved in running the algorithm, where elementary operations include addition/subtraction, multiplication/division, and comparison of two numbers. For simplicity, we assume in this paper that all elementary operations take the same amount of time. The space complexity of an algorithm is measured by the number of elementary objects that the algorithm needs to store during its execution. We assume in this paper that all elementary objects need the same amount of space.

The complexity analysis of Algorithm 3.4 requires a complexity estimate for constructing a k-d tree and for range counting. The complexity for constructing the k-d tree of a point multiset was studied in [Cormen *et al.* (2001)]. For convenience of citation, we state below a result from [Cormen *et al.* (2001)].

Proposition 4.1. *Let* m *and* N *be integers with* $m \ll N$. *If* $S \subseteq \mathbb{R}^m$ *is a multiset consisting of* N *points, the time complexity for constructing the k-d tree of* S *is* $O(N \log N)$ *and the space complexity is* $O(N \log N)$.

The time complexity of Algorithm 3.2, which we state below was analyzed in [Bentley (1980); Lee and Wong (1977)].

Proposition 4.2. *Let* m *and* N *be integers with* $m \ll N$. *Suppose that* S *is a multiset consisting of* N *points in* $\mathbb{R}^m$, T *is the k-d tree of* S, *and* R *is a cube in*

$\mathbb{R}^m$. *If* $m = 1$, *then the time complexity of Algorithm* 3.2 *is* $O(\log N)$. *If* $m \geq 2$, *then the time complexity is* $O(mN^{1-\frac{1}{m}})$.

We are now ready to present a complexity estimation for Algorithm 3.4.

Theorem 4.3. *If* $\mathbf{x} = [x_i : i \in \mathbb{Z}_N]$ *is a time series in* $\mathbb{R}^N$, m *is the template length, and* r *is the tolerance, then Algorithm* 3.4 *requires time complexity* $O(N^{2-\frac{1}{m+1}})$ *and the space complexity* $O(N \log N)$.

Proof. We prove this theorem by counting complexity for each step in Algorithm 3.4.

In step 1, computing the sample standard deviation $\sigma(\mathbf{x})$ requires $O(N)$ multiplications and additions and $O(1)$ storage units.

We now estimate the complexity for step 2. Since in step 2 we execute Algorithm 3.3, we need to consider the time complexity and space complexity of Algorithm 3.3 with inputs S and $r\sigma(\mathbf{x})$. Recall that T is the k-d tree of $S = \{\mathbf{x}_{m,i} : i \in \mathbb{Z}_{N-m}\} \subset \mathbb{R}^m$, where $\mathbf{x}_{m,i}$ are defined by Eq. (1). From Proposition 4.1, the numbers of operators and space required for constructing the k-d tree T are both $O(N \log N)$. In Algorithm 3.3, for each point in S Algorithm 3.2 is executed once. By Proposition 4.2, the number of operations required in Algorithm 3.2 is $O(N^{1-\frac{1}{m}})$. Since there are $N - m$ points in S, the total number of operations required in Algorithm 3.3 is $O(N \times N^{1-\frac{1}{m}}) = O(N^{2-\frac{1}{m}})$.

The same complexity estimates of the time complexity and the space complexity for step 3 are similarly obtained. The number of operations needed in step 4 is $\mathcal{O}(1)$.

Therefore, by summing the estimated numbers of operations and space needed in all steps of Algorithm 3.4, we obtain that its time complexity is $O(N^{2-\frac{1}{m+1}})$ and its space complexity is $O(N \log N)$. □

It is possible that some subsequences derived from a time series $\mathbf{x}$ coincide with each other. These coincident points contribute to reducing the time complexity in calculating sample entropy of the time series. All coincident points can be identified during the construction of the k-d tree. Note that the estimation of time complexity for Algorithm 3.4 in the proof of Theorem 4.3 depends on the number of points derived from $\mathbf{x}$ and a point may represent multiple subsequences of $\mathbf{x}$. As a result, Theorem 4.3 can be restated as follows.

Theorem 4.4. *Suppose that* $\mathbf{x} = [x_i : i \in \mathbb{Z}_N]$ *is a time series in* $\mathbb{R}^N$, m *is the template length, and* r *is the tolerance. If* $\mathbf{x}$ *generates* n *(with* $n < N$*) different points that represent its pattern templates, then the time complexity of Algorithm* 3.4 *is* $O(n^{2-\frac{1}{m+1}})$.

We remark that in applications, m in Theorem 4.4 is significantly smaller than N. This indicates that the proposed algorithm is indeed a fast algorithm. Examples that we present in the next section support this point.

5. A Numerical Experiment

We demonstrate in this section the efficiency of the proposed fast algorithm for computing sample entropy of a time series by a numerical experiment. The purpose of the experiment is to compare computational performance of Algorithm 3.4 with that of the direct algorithm. The numerical experiment is conducted on a computer with dual core processors (each having 2.80 GHz), with 1GB RAM memory and Windows XP Sp2 operating system.

In this experiment, we consider different time series, which include the heart interbeat interval time series derived from healthy subjects, subjects with congestive heart failure (CHF), and subjects with atrial fibrillation (AF). The heart-beat data were obtained from 24-hour continuous heart-beat recording of 72 healthy subjects, 43 CHF subjects, and 9 AF subjects. We also examine 17 surrogate data, which include stationary signals with different correlations, signals with linear, sinusoidal and power-law trends, and signals with different types of nonstationarity. For more information about surrogate data, see [Chen *et al.* (2002); Hu *et al.* (2001)].

For comparison purpose, we also study time series with different distributions. These time series include 30 time series of the uniform distribution, 30 time series of the Gaussian (standard normal) distribution and 30 time series of the $1/f$ distribution. For a positive integer N, we let $\mathbf{x} = [x_i : i \in \mathbb{Z}_N]$ denote a time series. A time series $\mathbf{x}$ is said to have a uniform (resp. Gaussian) distribution if all x_i are events of independent random variables with the uniform (resp. Gaussian) distribution. In some circumstances, a time series of the Gaussian distribution is also called a white noise. The $1/f$ distribution is generated as follows: start with a time series of the Gaussian distribution, calculate the Fourier transform of the time series, then impose a $1/f$ weight on the power spectrum of its Fourier transform where f stands for the frequency, finally we obtain a $1/f$ distribution by calculating the inverse Fourier transform. A $1/f$ distribution is believed to be a good approximation to a type of noise [Costa *et al.* (2005)] occurred in the physical community. These time series are illustrated in Fig. 2.

We apply both the direct algorithm and Algorithm 3.4 to these time series. Table 1 lists the average execution times (measured in seconds) of the direct algorithm and the fast algorithm applied to the healthy subjects, CHF, AF, Gaussian, $1/f$, uniform, and surrogate time series when the template length $m = 2$ and tolerance $r = 0.1$. In Table 1, D and F denote the direct algorithm and the fast algorithm, respectively. For all time series in the case of the data length being 64,000, the average execution time of the direct algorithm 2.1 is 100.8 seconds with the standard deviation 3.41 seconds and that of Algorithm 3.4 is 2.6 seconds with the standard deviation 0.79 seconds. Figure 3 illustrates the comparison of the two algorithms. Both Table 1 and Fig. 3 show that Algorithm 3.4 is a fast algorithm.

In Fig. 4, we verify the time complexities of the direct algorithm and Algorithm 3.4, which are given by Proposition 2.2 and Theorem 4.3, respectively. We recall from these results that the time complexities of the direct algorithm and

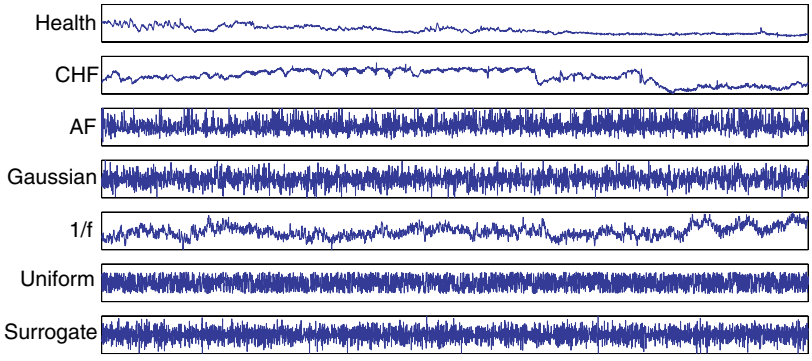


Fig. 2. Representation of healthy subjects, CHF, AF, Gaussian, $1/f$, uniform, and surrogate time series.

Table 1. Average execution times (in seconds) of the direct algorithm and the fast algorithm applied to healthy, CHF, AF, Gaussian, $1/f$, uniform, and surrogate time series when the template length $m = 2$ and tolerance $r = 0.1$.

Data length	Healthy		CHF		AF		Gaussian		$1/f$		Uniform		Surrogate	
	D	F	D	F	D	F	D	F	D	F	D	F	D	F
1,000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2,000	0.1	0.0	0.1	0.1	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0	0.1	0.0
4,000	0.4	0.1	0.4	0.1	0.4	0.1	0.4	0.1	0.4	0.1	0.4	0.1	0.4	0.1
8,000	1.6	0.2	1.7	0.2	1.6	0.2	1.6	0.2	1.6	0.2	1.7	0.2	1.8	0.3
16,000	6.4	0.4	6.5	0.5	6.3	0.4	6.1	0.4	6.2	0.4	6.6	0.5	7.0	0.8
32,000	25.2	1.0	25.6	1.2	24.8	0.8	24.2	0.7	24.6	0.8	26.0	1.3	27.6	2.0
64,000	99.1	2.4	103.1	2.7	99.9	1.9	96.1	1.5	98.1	1.9	103.7	3.2	109.9	5.5

Algorithm 3.4 are, respectively, $O(N^2)$ and $O(N^{2-\frac{1}{m+1}})$. Taking the logarithm (with base 2) of the execution times, we can observe the actual powers of the time complexities for these two algorithms from the slopes of straight lines illustrated in Fig. 4. The time complexity of the direct algorithm is about $O(N^2)$, which confirms the theoretical estimate given by Proposition 2.2. The computed time complexities of Algorithm 3.4 are far less than the theoretical estimate given by Theorem 4.3. The powers range from 1.15 to 1.23 for different value of m and have an average 1.18. We can see that the time complexity in the examples is less than the theoretical one shown in Theorem 4.3. This is because that the theoretical complexity presents the time complexity of the worst cases. Specifically, computing the sample entropy of a time series with length N by Algorithm 3.4 leads to call Algorithm 3.2 $O(N)$ times. Further, in most cases, Algorithm 3.2 does not request $O(N^{1-\frac{1}{m+1}})$ elementary operations which happens in the worst case. Hence, in practice, the elementary operations used in Algorithm 3.4 are less than $O(N^{2-\frac{1}{m+1}})$ which is the theoretical one shown in Theorem 4.3.

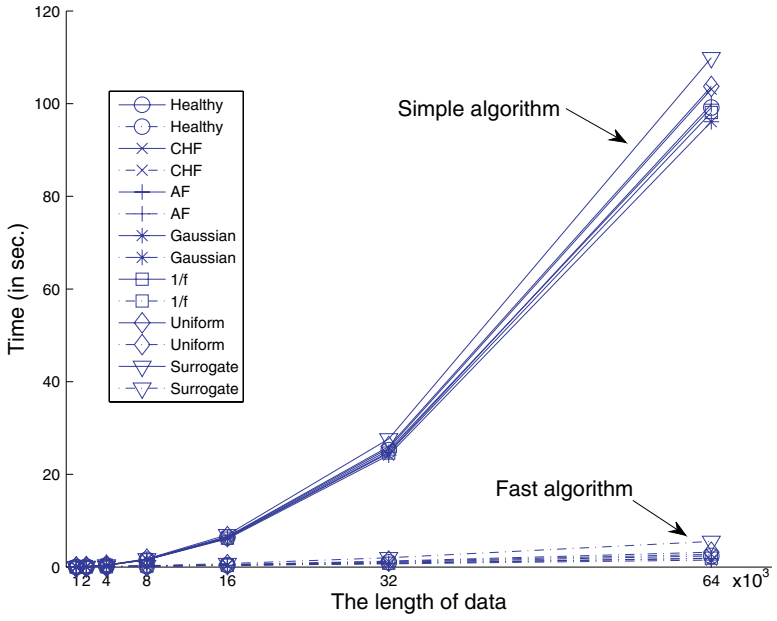


Fig. 3. A comparison of execution times of the direct algorithm and Algorithm 3.4, when $m = 2$ and $r = 0.1$.

When the data length is small, computational performance of the direct algorithm and Algorithm 3.4 is comparable. It is seen that Algorithm 3.4 performs much better than the direct algorithm when the data length is large. When the data length is 64,000, Algorithm 3.4 needs only 2.6% of the execution time required by the direct algorithm to calculate sample entropy.

6. Conclusion

Extracting the underlying information from time series is important not only for physiologic systems but also for other dynamic systems. The complexity of a physiologic system can provide information of interacting mechanisms, which drive the system. Approximate entropy, sample entropy, and multiscale sample entropy of a time series play important roles in measuring the complexity of its underlying physiologic system. Computation of these entropies of a time series requires to count the number of matched pairs in its pattern templates.

The direct algorithm for computing sample entropy of a time series requires time complexity $O(N^2)$, where N is its data length. Using the k-d tree to organize pattern templates of the time series, we design a fast algorithm which speeds up the calculation of sample entropy. The theoretical time complexity estimate is reduced to $O(N^{2-\frac{1}{m+1}})$, where m is the template length. The numerical results indicate that the execution time needed for the fast algorithm is much less than that given by the theoretical estimate. The proposed fast algorithm makes it possible to deal

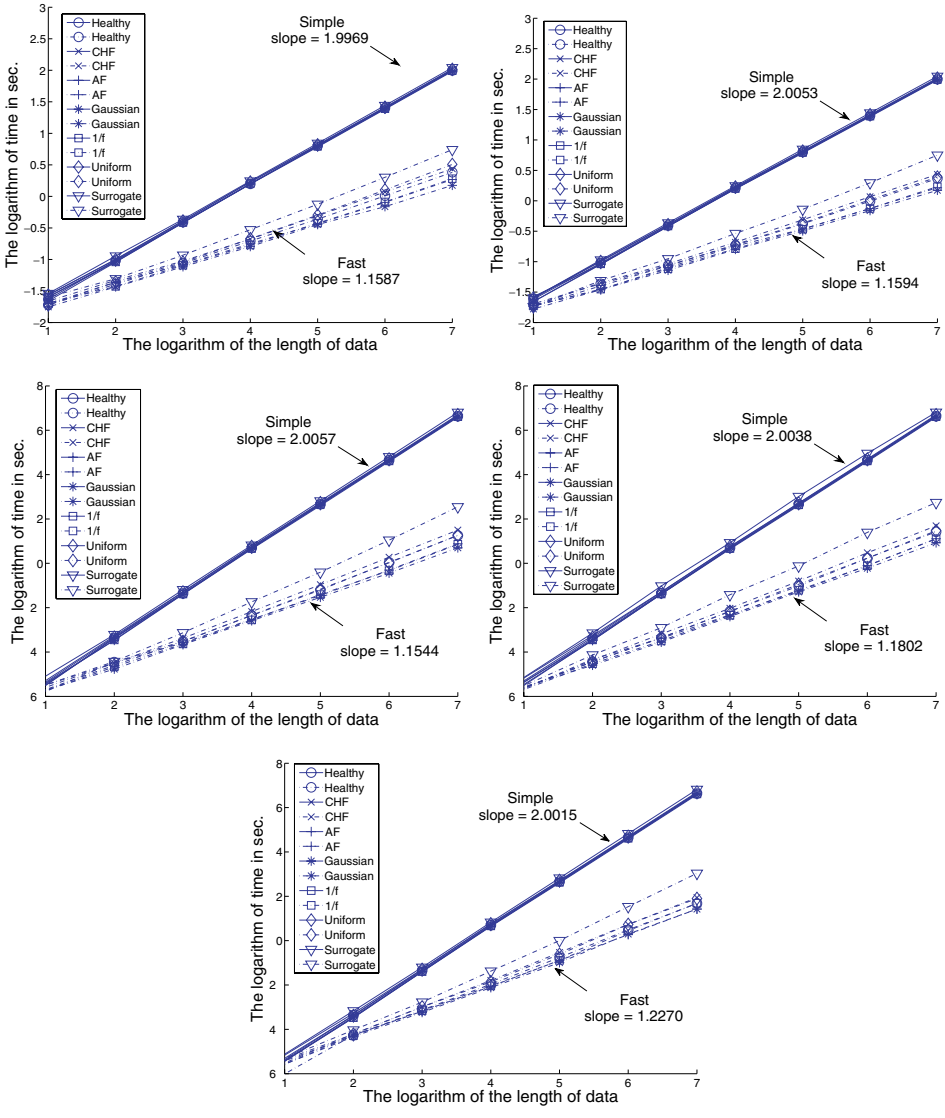


Fig. 4. The logarithm of the execution time with different m for the direct algorithm and Algorithm 3.4.

with physiologic time series of large size in a fixed time period. Physiologic time series of large size would allow us to extract more accurate information for clinical purposes.

Acknowledgments

We thank Drs. Madalena Costa, Ary L. Goldberger and C.-K. Peng of Harvard Medical School for providing biological data used in this paper. This research was

supported in part by US Air Force Office of Scientific Research under grant FA9550-09-1-0511, by the US National Science Foundation under grant DMS-0712827, by the Natural Science Foundation of China under grant 11071286.

Appendix

Here we present lemmas and propositions that lead to Algorithm 3.2.

Lemma A.1. *Let $S \subseteq \mathbb{R}^k$ be a finite multiset and T be the k-d tree of S. Suppose that $\mathfrak{N}$ is a node of T. If $\mathbf{u}_{\mathcal{B}_{\mathfrak{N}} \cap S} \neq \mathbf{v}_{\mathcal{B}_{\mathfrak{N}} \cap S}$, and $\mathcal{B}_{\mathfrak{N}} \cap S$ is split into two multisets S' and S'' by the midpoint splitting rule, then*

$$S' = \mathcal{B}_{\mathfrak{N}'} \cap S \tag{A.1}$$

and

$$S'' = \mathcal{B}_{\mathfrak{N}''} \cap S, \tag{A.2}$$

where $\mathfrak{N}'$ and $\mathfrak{N}''$ are associated with S' and S'', respectively.

Proof. Since $\mathcal{B}_{\mathfrak{N}} \cap S$ is split into two multisets S' and S'' by the midpoint splitting rule, we have that

$$\mathcal{B}_{\mathfrak{N}} \cap S = S' \cup S''. \tag{A.3}$$

Without loss of generality, we assume that $\mathcal{B}_{\mathfrak{N}} \cap S$ is split into two multisets S' and S'' by the midpoint split rule with $i = l$, for some $l \in \mathbb{Z}_k$. We also assume for all $\mathbf{x} := [x_i : i \in \mathbb{Z}_k] \in S'$ and $\mathbf{y} := [y_i : i \in \mathbb{Z}_k] \in S''$ that

$$x_l < y_l. \tag{A.4}$$

We now prove Lemma A.1. Note that $S' \subseteq S$ and $S'' \subseteq S$. From the definitions of $\mathcal{B}_{\mathfrak{N}'}$ and $\mathcal{B}_{\mathfrak{N}''}$, we know that

$$S' \subseteq \mathcal{B}_{\mathfrak{N}'} \cap S \tag{A.5}$$

and

$$S'' \subseteq \mathcal{B}_{\mathfrak{N}''} \cap S. \tag{A.6}$$

It suffices to prove that

$$S' \supseteq \mathcal{B}_{\mathfrak{N}'} \cap S. \tag{A.7}$$

Assume to the contrary that there exits $\mathbf{x}^* := [x_i^* : i \in \mathbb{Z}_k] \in \mathcal{B}_{\mathfrak{N}'} \cap S$ such that

$$\mathbf{x}^* \notin S'. \tag{A.8}$$

From Eq. (A.4) and the definitions of $\mathcal{B}_{\mathfrak{N}'}$ and $\mathcal{B}_{\mathfrak{N}''}$ we know for all $\mathbf{y} := [y_i : i \in \mathbb{Z}_k] \in \mathcal{B}_{\mathfrak{N}''}$ that $x_l^* < y_l$. This implies that

$$\mathbf{x}^* \notin \mathcal{B}_{\mathfrak{N}''} \cap S. \tag{A.9}$$

Combining Eqs. (A.6) and (A.9) yields that

$$\mathbf{x}^* \notin S''. \tag{A.10}$$

From Eqs. (A.3), (A.8) and (A.10) we obtain that

$$\mathbf{x}^* \notin \mathcal{B}_{\mathfrak{N}} \cap S. \tag{A.11}$$

On the other hand, since $S' \subseteq \mathcal{B}_{\mathfrak{N}} \cap S$, from the definition of $\mathcal{B}_{\mathfrak{N}'}$ we know that $\mathcal{B}_{\mathfrak{N}'} \subseteq \mathcal{B}_{\mathfrak{N}}$. Thus, noting that $\mathbf{x}^* \in \mathcal{B}_{\mathfrak{N}'} \cap S$, we obtain that $\mathbf{x}^* \in \mathcal{B}_{\mathfrak{N}} \cap S$, which contradicts Eq. (A.11). The contradiction ensures that Eq. (A.7) must be satisfied, and as a result, Eq. (A.1) holds.

Equation (A.2) is proved similarly with S', S'', $\mathfrak{N}'$, and $\mathfrak{N}''$ being replaced by S'', S', $\mathfrak{N}''$, and $\mathfrak{N}'$, respectively. □

A node $\mathfrak{N}$ of T is said to be uniquely associated with a submultiset $A \subseteq S$ if for all sub-multisets $B \subseteq S$, with which $\mathfrak{N}$ is associated, we have that $A = B$. In the next lemma, we show that each node $\mathfrak{N}$ of T is uniquely associated with the multiset $\mathcal{B}_{\mathfrak{N}} \cap S$.

Lemma A.2. *Let $S \subseteq \mathbb{R}^k$ be a finite multiset and T be the k-d tree of S. Then, $\mathfrak{N}$ is uniquely associated with the multiset $\mathcal{B}_{\mathfrak{N}} \cap S$.*

Proof. It can be shown by induction that $\mathfrak{N}$ is a node of T associated with the multiset $\mathcal{B}_{\mathfrak{N}} \cap S$, beginning with assuming that $\mathfrak{N}$ is the root node of T.

It remains to prove the uniqueness. Suppose that $\mathfrak{N}$ is associated with a multiset $\tilde{S} \subseteq S$. It suffices to prove that

$$\tilde{S} = \mathcal{B}_{\mathfrak{N}} \cap S. \tag{A.12}$$

From the definition of $\mathcal{B}_{\mathfrak{N}}$, we conclude that $\tilde{S} \subseteq \mathcal{B}_{\mathfrak{N}}$, from which we have that

$$\tilde{S} \subseteq \mathcal{B}_{\mathfrak{N}} \cap S \tag{A.13}$$

because $\tilde{S} \subseteq S$. Since $\mathfrak{N}$ is associated with both multisets $\mathcal{B}_{\mathfrak{N}} \cap S$ and $\tilde{S}$, by the definition of $\mathfrak{N}$, we observe that

$$\mathfrak{C}(\mathcal{B}_{\mathfrak{N}} \cap S) = \mathfrak{C}(\tilde{S}). \tag{A.14}$$

Combining relations (A.13) and (A.14) yields Eq. (A.12). □

We now turn to proving lemmas that will be employed to show that function F in Eq. (8) is well defined.

Lemma A.3. *Let $S \subseteq \mathbb{R}^k$ be a finite multiset and T be the k-d tree of S. If $\mathfrak{N}$ is a node of T, then either $\mathfrak{N}$ has two child nodes, or $\mathfrak{N}$ is a leaf node. Moreover, if $\mathfrak{N}$ is a leaf node of T, then for all cubes $R \subset \mathbb{R}^k$*

$$\mathcal{B}_{\mathfrak{N}} \cap R = \mathcal{B}_{\mathfrak{N}}, \quad \text{or} \quad \mathcal{B}_{\mathfrak{N}} \cap R = \emptyset. \tag{A.15}$$

Proof. Lemma A.2 ensures that $\mathfrak{N}$ is uniquely associated with $\mathcal{B}_{\mathfrak{N}} \cap S$. If node $\mathfrak{N}$ is not a leaf node of T, then, from step 2 of Algorithm 3.1 we have that

$\mathbf{u}_{\mathcal{B}_{\mathfrak{N}}\cap S} \neq \mathbf{v}_{\mathcal{B}_{\mathfrak{N}}\cap S}$. Thus, from Algorithm 3.1, we know that $\mathfrak{N}$ has two child nodes. This proves the first statement.

To prove the second statement, we let $\mathfrak{N}$ be a leaf node of T. Then, $\mathbf{u}_{\mathcal{B}_{\mathfrak{N}}\cap S} = \mathbf{v}_{\mathcal{B}_{\mathfrak{N}}\cap S}$, which implies that either $\mathcal{B}_{\mathfrak{N}} \cap S$ consists of a single point, or all points in $\mathcal{B}_{\mathfrak{N}} \cap S$ are coincident, or $\mathcal{B}_{\mathfrak{N}} \cap S$ is empty. When $\mathcal{B}_{\mathfrak{N}} \cap S$ consists of a single point, or all points in $\mathcal{B}_{\mathfrak{N}} \cap S$ are coincident, we conclude that Eq. (A.15) holds for all cubes $R \subset \mathbb{R}^k$. When $\mathcal{B}_{\mathfrak{N}} \cap S$ is empty, we see for all cubes $R \subset \mathbb{R}^k$ that $\mathcal{B}_{\mathfrak{N}} \cap R = \emptyset$, proving Eq. (A.15). □

The following corollary is an immediate consequence of Lemma A.3.

Corollary A.4. *Let $S \subseteq \mathbb{R}^k$ be a finite multiset and T be the k-d tree of S. For a given node $\mathfrak{N}$ of T, if there exists a cube $R \subset \mathbb{R}^k$ such that*

$$\mathcal{B}_{\mathfrak{N}} \cap R \neq \mathcal{B}_{\mathfrak{N}}, \quad \text{and} \quad \mathcal{B}_{\mathfrak{N}} \cap R \neq \emptyset, \tag{A.16}$$

then $\mathfrak{N}$ has two child nodes.

Lemma A.5. *Let $S \subseteq \mathbb{R}^k$ be a finite multiset, T be the k-d tree of S, and $\mathfrak{N}$ be a node of T. Suppose that R is a cube in $\mathbb{R}^k$. If $\mathcal{B}_{\mathfrak{N}} \cap R = \emptyset$ or $\mathcal{B}_{\mathfrak{N}} \cap R = \mathcal{B}_{\mathfrak{N}}$, then*

$$F(\mathfrak{N}, R) = \mathfrak{C}(\mathcal{B}_{\mathfrak{N}} \cap S \cap R). \tag{A.17}$$

Proof. If $\mathcal{B}_{\mathfrak{N}} \cap R = \emptyset$, we see $\mathcal{B}_{\mathfrak{N}} \cap S \cap R = \emptyset$. Thus, by the definition of function F, we have that $F(\mathfrak{N}, R) = 0$. This leads to Eq. (A.17).

If $\mathcal{B}_{\mathfrak{N}} \cap R = \mathcal{B}_{\mathfrak{N}}$, then $\mathcal{B}_{\mathfrak{N}} \cap S \cap R = \mathcal{B}_{\mathfrak{N}} \cap S$. Thus, by Lemma A.2 we conclude that $\mathfrak{N}$ is uniquely associated with $\mathcal{B}_{\mathfrak{N}} \cap S \cap R$. Hence, Eq. (A.17) holds. □

The following proposition ensures that the output of Algorithm 3.2 is equal to $\mathfrak{C}(S \cap R)$.

Proposition A.6. *Suppose that $S \subset \mathbb{R}^k$ is a finite multiset and $R \subseteq \mathbb{R}^k$ is a cube. If T is the k-d tree of S with the root node $\mathfrak{N}_r$, then $F(\mathfrak{N}_r, R)$ is equal to $\mathfrak{C}(S \cap R)$.*

Proof. It suffices to prove that for each node $\mathfrak{N}$ of the k-d tree T

$$F(\mathfrak{N}, R) = \mathfrak{C}(\mathcal{B}_{\mathfrak{N}} \cap S \cap R). \tag{A.18}$$

We prove Eq. A.18 by induction. We assume that $\mathfrak{N}$ is a leaf node of T. By Lemma A.3, we know that $\mathcal{B}_{\mathfrak{N}} \cap R = \mathcal{B}_{\mathfrak{N}}$ or $\mathcal{B}_{\mathfrak{N}} \cap R = \emptyset$. Thus, from Lemma A.5, we conclude that Eq. (A.18) holds.

We assume that $\mathfrak{N}$ is a node of T but not a leaf node. Lemma A.3 ensures that $\mathfrak{N}$ has two child nodes $\mathfrak{N}'$ and $\mathfrak{N}''$. We assume that

$$F(\mathfrak{N}', R) = \mathfrak{C}(\mathcal{B}_{\mathfrak{N}'} \cap S \cap R) \quad \text{and} \quad F(\mathfrak{N}'', R) = \mathfrak{C}(\mathcal{B}_{\mathfrak{N}''} \cap S \cap R), \tag{A.19}$$

and prove that Eq. (A.18) holds for the particular node $\mathfrak{N}$.

If $\mathcal{B}_{\mathfrak{N}} \cap R = \mathcal{B}_{\mathfrak{N}}$ or $\mathcal{B}_{\mathfrak{N}} \cap R = \emptyset$, Lemma A.5 yields Eq. (A.18). Otherwise, the definition of F ensures that

$$F(\mathfrak{N}, R) = F(\mathfrak{N}', R) + F(\mathfrak{N}'', R). \tag{A.20}$$

By Lemma A.2, $\mathfrak{N}$ is uniquely associated with $\mathcal{B}_{\mathfrak{N}} \cap S$. Since $\mathfrak{N}$ has two child nodes $\mathfrak{N}'$ and $\mathfrak{N}''$, from Algorithm 3.1 we know that $\mathcal{B}_{\mathfrak{N}} \cap S$ is split into two submultisets S' and S'' having the property

$$\mathcal{B}_{\mathfrak{N}} \cap S = S' \cup S'' \tag{A.21}$$

and

$$S' \cap S'' = \emptyset, \tag{A.22}$$

and that $\mathfrak{N}'$ and $\mathfrak{N}''$ are associated with S' and S'', respectively. By Lemma A.1, we have that

$$S' = \mathcal{B}_{\mathfrak{N}'} \cap S \quad \text{and} \quad S'' = \mathcal{B}_{\mathfrak{N}''} \cap S. \tag{A.23}$$

Substituting Eq. (A.23) into Eqs. (A.21) and (A.22), and then taking the intersection of R with both sides of the resulting equation, we have that

$$\mathcal{B}_{\mathfrak{N}} \cap S \cap R = (\mathcal{B}_{\mathfrak{N}'} \cap S \cap R) \cup (\mathcal{B}_{\mathfrak{N}''} \cap S \cap R) \tag{A.24}$$

and

$$(\mathcal{B}_{\mathfrak{N}'} \cap S \cap R) \cap (\mathcal{B}_{\mathfrak{N}''} \cap S \cap R) = \emptyset. \tag{A.25}$$

Equations (A.24) and (A.25) yield that

$$\mathfrak{C}(\mathcal{B}_{\mathfrak{N}} \cap S \cap R) = \mathfrak{C}(\mathcal{B}_{\mathfrak{N}'} \cap S \cap R) + \mathfrak{C}(\mathcal{B}_{\mathfrak{N}''} \cap S \cap R). \tag{A.26}$$

Substituting Eqs. (A.19) into (A.26), we obtain that

$$\mathfrak{C}(\mathcal{B}_{\mathfrak{N}} \cap S \cap R) = F(\mathfrak{N}', R) + F(\mathfrak{N}'', R). \tag{A.27}$$

Combining Eqs. (A.20) and (A.27) leads to Eq. (A.18). Therefore, by the induction principle, we have completed the proof of this lemma. □

References

Abasolo, D., Hornero, R., Espino, P., Poza, J., Sanchez, C. I. and Rosa, R. (2005). Analysis of regularity in the EEG background activity of Alzheimer's disease patients with approximate entropy. *Clin. Neurophysiol.*, **116**: 1826–1834.

Agarwal, P. K. and Erickson, J. (1999). Geometric range searching and its relatives, *Advances in Discrete and Computational Geometry*, eds. Chazelle, B., Goodman, J. E. and Pollack, R. in Contemporary Mathematics, Vol. 233 (American Mathematical Society, Providence, RI), pp. 1–56.

Al-Angari, H. M. and Sahakian, A. V. (2007). Use of sample entropy approach to study heart rate variability in obstructive sleep apnea syndrome, *IEEE Trans. Biomed. Eng.*, **54**: 1900–1904.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching, *Commun. ACM*, **18**: 509–517.

Bentley, J. L. (1980). Multidimensional divide-and-conquer, *Commun. ACM*, **23**: 214–229.

Blizard, W. D. (1989). Multiset theory, *N. D. J. F. L.*, **30**: 36C66.

Chen, Z. Ivanov, P., Ch. Hu, K. and Stanley, H. E. (2002). Effects of nonstationarities on detrended fluctuation analysis, *Phys. Rev. E.*, **65**: 041107.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001). *Introduction to Algorithms* (2nd edn.) MIT Press and McGraw-Hill.

Costa, M., Goldberger, A. L. and Peng, C.-K. (2002a). Multiscale entropy analysis of complex physiologic time series, *Phys. Rev. Lett.*, **89**: 068102.

Costa, M., Goldberger, A. L. and Peng, C.-K. (2002b). Multiscale entropy to distinguish physiologic and synthetic RR time series, *Comput. Cardiol.*, **29**: 137–140.

Costa, M. and Healey, J. A. (2003). Multiscale entropy analysis of complex heart rate dynamics: discrimination of age and heart failure effects, *Comput. Cardio.*, **30**: 705–708.

Costa, M. Goldberger, A. L. and Peng, C.-K. (2005). Multiscale entropy analysis of biological signals, *Phys. Rev. E.*, **71**: 021906.

Cover, T. and Thomas, J. (1991). *Elements of Information Theory*, Wiley, New York.

Eckmann, J. P. and Ruelle, D. (1985). Addendum: Ergodic theory of chaos and strange attractors, *Rev. Mod. Phys.*, **57**: 617–656.

Goldberger, A. L., Amaral, L. Glass, L., Hausdorff, J. M., Ivanov, P., Ch. Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K. and Stanley, H. E. (2000). Physiobank, Physiotoolkit, and Physionet: Components of a new research resource for complex physiologic signals, *Circulation*, **101**: 215–220.

Grassberger, P. (1990). Information and Complexity Measures in Dynamical Systems, *Information Dynamics,* eds. Atmanspacher, H. and Scheingraber, H., Plenum press, New York, pp. 15–33.

Hu, K., Ivanov, P., Ch. Chen, Z., Carpena, P. and Stanley, H. E. (2001). Effects of trends on detrended fluctuation analysis, *Phys. Rev. E.*, **64**: 011114.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information, *Problems Inf. Theory*, **1**: 1–7.

Kozuch, M. and Wang, L. (2001). Approximate entropy as a measure of irregularity in earthquake sequences, *American Geophysical Union*, Fall Meeting, San Francisco, California, USA.

Lake, D. E., Richman, J. S., Griffin, M. P. and Moorman, J. R. (2002). Sample entropy analysis of neonatal heart rate variability, *Am. J. Physiol.*, **283**: 789–792.

Lee, D. T. and Wong, C.-K. (1977). Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, *Acta Inf.*, **9**: 23–29.

Manis, G. (2008). Fast computation of approximate entropy, *Comput. Meth. Prog. Biomed.*, **91**: 48–54.

Pincus, S. M. (1991). Approximate entropy as a measure of system complexity, *Proc. Natl. Acad. Sci.*, **88**: 2297–2301.

Pincus, S. M. and Kalman, R. (2004). Irregularity, volatility, risk, and financial market time series, *Proc. Natl. Acad. Sci.*, **101**: 709C714.

Richman, J. S. and Moorman, J. R. (2000). Physiological time series analysis using approximate entropy and sample entropy, *Am. J. Physiol. Heart. Circ. Physiol.*, **278**: 2039–2049.

Zhang, T., Yang, Z. and Coote, J. H. (2007). Cross-sample entropy statistic as a measure of complexity and regularity of renal sympathetic nerve activity in the rat, *Exp. Physiol.*, **92**: 659–669.