Journal of Marine Science and Technology

Volume 19 | Issue 1 Article 13

COMPUTING MULTISCALE ENTROPY WITH ORTHOGONAL RANGE SEARCH

Yu-Hsiang Pan
Department of Environmental Biology and Fisheries Science, National Taiwan Ocean University, Keelung, Taiwan, R.O.C., D95310001@mail.ntou.edu.tw

Wei-Yen Lin
Department of Mechanical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

Yung-Hung Wang
Yung-Hung Wang, Society of Streams, R.O.C., Taiwan, R.O.C

Kuo-Tien Lee
Department of Environmental Biology and Fisheries Science, National Taiwan Ocean University, Keelung, Taiwan, R.O.C.

COMPUTING MULTISCALE ENTROPY WITH ORTHOGONAL RANGE SEARCH

Acknowledgements

The authors would like to thank professor Sheng-Fu Liang in National Cheng Kung University for providing the EEG data. This work was funded in part by the Industrial Development Bureau Ministry of Economic Affairs, Taiwan (ROC).

COMPUTING MULTISCALE ENTROPY WITH ORTHOGONAL RANGE SEARCH

Yu-Hsiang Pan*, Wei-Yen Lin**, Yung-Hung Wang***, and Kuo-Tien Lee*

Key words: computational geometry, *k*-d tree, multi-scale entropy, signal processing.

ABSTRACT

Multi-scale entropy (MSE) is a measurement of a system's complexity. It has received a great deal of attention in recent years, and its effectiveness has been verified, and applied in a number of different fields. However, the algorithms proposed in past studies required $O(N^2)$, which represented a degree of execution time insufficient for on-line applications, or for applications with long-term correlations. In this study, we showed that the probability function in the entropy term could be transformed into an orthogonal range search in the field of computational geometry. We then developed an efficient new algorithm for computing multi-scale entropy. The execution time in the results of our experiments with electrocardiogram (ECG), electroencephalography (EEG), interbeat interval (RR), and mechanical and ecological signals showed a significant improvement from 10 to 70 times over that of conventional methods for N = 80,000. Because the execution time has been significant reduced, the new algorithm could be applied to online diagnosis, in the computation of MSE for long-term correlation of signal.

I. INTRODUCTION

The longstanding problem of deriving useful measurements of time series complexity is important for the analysis of physiology [2], biology [7], geosciences [9], and mechanical signals [20]. Recently Costa *et al*. [6-8] introduced multi-scale entropy (MSE) analysis to measure the complexity of finite length time series. MSE measures complexity by taking into account multiple time scales. This computational methodology can be quite effective for quantifying the complexity of a time series. However, the algorithms for computing the complexity proposed in past studies required $O(N^2)$, which is unrealistic for applications with long data sets, or for online monitoring. To find the statistical meaning of signals, a large amount of data [15] and many parameters typically had to be collected.

For observing long period data, it is useful to analyze local data using rolling windows to reduce execution time [9]. Here, data is first partitioned into number of windows, at which MSE is computed separately for each window. This reduces the execution time, but for signals with long-term correlations, the size of the window has to be carefully selected to cover the largest time scale in the signal. For example, the time scales in an ecosystem may vary from seconds to millennia [10]. One window contains at least 10^{10} data points. Another example measures the complexity of a computer program [7]. An executable computer program exhibits long-term correlations, and the largest time scale may be as large as the size of the computer program. In both situations, it is impossible to partition the data sets into small windows [12].

Current computation speed is insufficient for online monitoring [2, 19], especially in cases for which a signal generates more data points due to a higher sampling rate. For example, EEG (electroencephalography) signal may be sampled at 1 kHz. Mechanical data may be sampled at 10 kHz or higher.

The work of Manis [14] reduced the execution time for computing approximate entropy. He used bucket-assisted techniques similar to bucket-sort to early exclude impossible matches of similarities. That improved the execution time, but it was still an $O(N^2)$ algorithm. To make online monitoring possible, Sugisaki [19] derived a recursive sample entropy algorithm for the situation when rolling windows overlapped. The algorithm had two drawbacks: Firstly, the computed sample entropy was only an approximation. Secondly, computational efficiency decayed with a decrease in overlap length, and the improvement reduced to zero when overlap length is zero.

This motivated the authors to develop a more efficient way of computing MSE. First, we viewed the computation of MSE from another perspective and showed that the probability function was an orthogonal range search problem, in the field of computational geometry. We then developed a new algorithm to reduce the execution time.

The remainder of the paper is organized as follows: Section II provides a review of multi-scale entropy. In Section III, the

Paper submitted 01/19/10; revised 06/03/10; accepted 07/20/10. Author for correspondence: Yu-Hsiang Pan (e-mail: D95310001@mail.ntou.edu.tw).
**Department of Environmental Biology and Fisheries Science, National Taiwan Ocean University, Keelung, Taiwan, R.O.C.*
***Department of Mechanical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.*
****Yung-Hung Wang, Society of Streams, R.O.C., Taiwan, R.O.C.*

probability function in the entropy term is transformed into an orthogonal range search problem. In Section IV, the *k*-d tree algorithm is applied to compute the multi-scale entropy. In Section V, the range tree algorithm is applied to compute the multi-scale entropy. In Section VI, we present numerical examples to illustrate the effectiveness of our new algorithm, and we conclude the paper in Section VII.

II. REVIEW OF MULTI-SCALE ENTROPY

The description of the MSE analysis [7] is given a one-dimensional discrete time series; $\{x_1 \ldots x_i \ldots x_N\}$ construct the consecutive coarse-grained time series, $\{y^\tau\}$ determined by the scale factor, τ, according to the equation:

$$y_j^\tau = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i \tag{1}$$

Where τ represents the scale factor and $1 \le j \le \frac{N}{\tau}$. The length of each coarse-grained time series is $\frac{N}{\tau}$. For a scale of one, the coarse-grained time series is simply the original time series. Next, the authors calculate the sample entropy for each scale using the following method. Let

$$\{F\} = \{ f_1 \ldots f_i \ldots f_N\} \tag{2}$$

be a time series of length N, and $u_m(i) = \{f_i, f_{i+1}, \ldots f_{i+m-1}\}$ be vectors of length m.

For finite length N, the approximation entropy (A_E) [16] is:

$$A_E(m,r,N) = \frac{1}{N-m} \sum_{i=1}^{N-m} \frac{\ln(n_i^m)}{\ln(n_i^{m+1})} \tag{3}$$

In addition, the sample entropy (S_E) [18] is:

$$S_E(m,r,N) = \ln\left(\frac{\sum_{i=1}^{N-m} n_i^m}{\sum_{i=1}^{N-m} n_i^{m+1}}\right) = \ln\left(\frac{n_n}{n_d}\right) \tag{4}$$

In Eq. (4), n_i^m stands for the number of vectors that satisfy $d[u_m(i), u_m(j)] \le r$, where d is the Euclidean distance.

$$d[u_m(i), u_m(j)] = \max\{|f(i+k) - f(j+k)|: 0 \le k \le m-1\}$$

Where j ranges from 1 to $(N - m)$ and $i \ne j$ to exclude self-matches. S_E requires much shorter data sets than A_E [18]. As far as computation is concerned, sample entropy and approximate entropy are very similar. The authors focus on computing S_E, the algorithms can be easily modified to compute A_E too.

The algorithm proposed in [7] is repeated here, and the authors refer to the algorithm as Algorithm 1. From Algorithm 1, the time complexity of the original MSE algorithm is $O(N^2)$ for each scale since two loops (i, j) is required, and the total execution times for all scales is:

$$\sum_s n_s^2 = \sum_s \left(\frac{n}{s}\right)^2 = O(N^2) \tag{5}$$

Unless otherwise specified, the values of the parameters used to calculate S_E are $m = 2$, and $r = 0.15 \times SD$ (SD is the standard deviation of the original time series).

Algorithm 1: Brute force method

```
For scale=1: max Scale {
 Find coarse scale f, which is the geometric mean of the input signal.
 for i=1:N {
  for j=i+1:N {
       if (| f_i – f_j) < ε and | f_{i+1} – f_{j+1}| < ε)
    n_n = n_n + 1 /* compute numerator in (4) */
         if (| f_{i+2} – f_{j+2}| < ε) {
             n_d = n_d + 1; /* compute denominator in (4) */
         } // if
        } // if
     } // j
   } // i
   Compute entropy [scale] = -log (n_n / n_d); // from (3)
} // s
```

III. MATHEMATICAL TRANSFORMATIONS

To compute the sample entropy $S_E(m, r, N)$ in Eq. (4), it is necessary to compute the distance vectors (probability) n_n and n_d for each scale.

The time series in Eq. (2), $\{F\} = \{f_1 \ldots f_i \ldots f_N\}$, can be transformed into d (where $d = m + 1$) dimensional point set by setting:

$$x_i = f_i, \; y_i = f_{i+1}, \; z_i = f_{i+2} \tag{6}$$

The term n_n in Eq. (3) is equivalent to the number of points that satisfy the following constraint:

$$f_i - \varepsilon < f_j < f_i + \varepsilon; \; f_{i+1} - \varepsilon < f_{j+1} < f_{i+1} + \varepsilon;$$
$$f_{i+2} - \varepsilon < f_{j+2} < f_{i+2} + \varepsilon \tag{7}$$

Define

$$x_1 = f_j - \varepsilon \;\; x_2 = f_j + \varepsilon; \; y_1 = f_{j+1} - \varepsilon \;\; y_2 = f_{j+1} + \varepsilon;$$
$$z_1 = f_{j+2} - \varepsilon \;\; z_2 = f_{j+2} + \varepsilon \tag{8}$$

From Eq. (6) and Eq. (8), n_n is equivalent to the number of

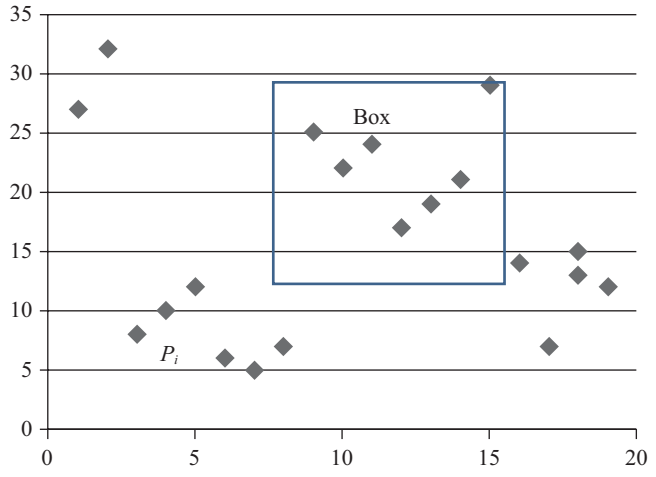

Fig. 1. A demonstration of two-dimensional orthogonal range search. Given the point set {*P*}, query the number of points inside the query Box B.

points that satisfy the following constraint:

$$x_{i1} < x_j < x_{i2};\ y_{i1} < y_j < y_{i2};\ z_{i1} < z_j < z_{i2} \tag{9}$$

In other words, for each point i, $P_i = (X_i, Y_i, Z_i)$ $1 \le i \le N$, find the number of points inside the bounding box

$$[x1{:}x2] \times [y1{:}y2] \times [z1{:}z2] \tag{10}$$

This is an orthogonal range search problem in the field of computational geometry. The computations of n_n are equivalent to the $m-1$ dimensional orthogonal range counting problem and the computations of n_d are equivalent to the m dimensional orthogonal range counting problem. Once n_n and n_d are computed, S_E can be calculated directly. The computation of MSE is equivalent to a d and $d-1$ dimensional orthogonal range search problem ($d = m + 1$). For computing MSE, there is no need to report points in the rectangle range, only point counting is required.

Figurg 1 is the geometric view of Eq. (10) for $m = 1$ (two-dimension). In this view, Algorithm 1 is summarized as follows: given a query box B, identify whether each point in the domain is inside the box or not. One needs to query N times for each point (box), and there are N query boxes. It then requires $N^2 = O(N^2)$ time to finish all queries. In a sense, Algorithm 1 may be interpreted as a brute force method.

Orthogonal range search is used to design a data structure to store the point sets for rapid query times. The *k*-d tree [3] is the earliest and probably the simplest one.

As far as query time is concerned, the best-known data structure is the range tree [4] combined with fractional cascading; e.g. algorithms [13, 20] can be applied to orthogonal range search problems. Brief reviews are found in [1, 5], and the results are summarized in Table 1.

From Table 1, there is tradeoff between time and strategic complexity. Though a range tree is faster, it requires more storage.

Table 1. Comparison of different (d > 2) orthogonal range search algorithms.

	k-d-tree	range tree + fractional cascading
Construct Time	$O(N \log(N))$	$O(N \log N)^{d-1}$
Search Time	$O(N^{1-\frac{1}{d}})$	$O((\log N)^{d-1}$
Memory Cost	$O(N)$	$O(N (\log N)^{d-1})$

Both *k*-d tree and range tree are well-known orthogonal range search algorithms. The pseudo code for applying *k*-d tree and range tree to compute Eq. (4) are described in Sec IV and Sec. V.

IV. *k*-d TREE ALGORITHM

The *k*-d tree, proposed by Bentley in 1975, is a binary tree, each of whose nodes v is associated with a rectangle Bv. If Bv does not contain any point in its interior, v will be a leaf. Otherwise, Bv is partitioned into two rectangles by drawing a horizontal or vertical line such that each rectangle contains, at most, half of the points; the splitting lines are alternately horizontal and vertical. A *k*-d tree can be extended to higher dimensions in an obvious manner.

The steps of applying *k*-d tree algorithm to compute S_E are as follows:

(a) For each scale, find the coarse scale of the discrete time series by Eq. (1).
(b) Transform the discrete time series into discrete space point sets by Eq. (6).
(c) $k = m - 1$, Build the *k*-d tree using the space points.
(d) For each space point, calculate the query box by Eq. (8).
(e) Query the number of points inside this box (n_i^m) by the *k*-d tree query algorithm.
(f) Compute n_n by Eq. (4)
(g) Set $k = m$, repeat step (b) through (f) to compute n_d
(h) Compute S_E by Eq. (4)

To apply an orthogonal range search (counting) algorithm to compute S_E, one needs two computer functions (libraries): a building tree function, and a query function. The other steps are straightforward. The pseudo-code is summarized in Algorithm 2.

Algorithm 2: *k*-d tree algorithm

```
Given time signal from Eq. (2) and specify r
For scale=1: maxScale {
  Find coarse scale f, which is the geometric mean of the input signal.
  Point Array = Transform time signal to space points set using Eq. (6)
```

```
  for(k=m-1:m)
    d = m+1;
    Build up d dimensional k-d tree = build KD Tree (point
Array)
    for i=1:N-m {
    count = count + kd Search(point Array[i],tree);
   } // i
   if (k==m-1) {
    n_n = count;
   }
   else if (k==m) {
     n_d = count
   }
 } // k
 Compute sample entropy from Eq. (4):
S_E(m, r, N_s) = ln( n_n / n_d )
} // scale
```

Time and Memory Complexity

In Eq. (5), the computation of scale one was the bottleneck, and the authors focused on scale one in the analysis. Also, it was necessary to perform a d dimensional k-d tree search and a $d-1$ dimensional k-d tree search. The cost for the latter could be overlooked, since it was much smaller than the first.

Step 1. Transform the original discrete time series to a spaced point set from Eq. (2): The time cost is $O(N)$ and the memory cost is $O(N)$.

Step 2. Construct the k-d tree: The d dimensional k-d tree is constructed by using all N-m points. The time cost is $O(N \log N)$ and the memory cost is $O(N)$.

Step 3. Range query: For the d dimensional k-d search, the time cost is $N \cdot O(N^{1-\frac{1}{d}})$ for N queries, and the memory cost is $O(N)$.

To summarize, from the description, the preparation time is $O(N \log N)$ (Steps 1 and 2), and the total query time is $N \cdot O(N^{1-\frac{1}{d}})$ for N queries. The total time of the three steps is $O(N) + O(N \log N) + N \cdot O(N^{1-\frac{1}{d}}) = O(N \cdot N^{1-\frac{1}{d}})$. Typically, $m = 2$, then the time cost is reduced from $O(N^2)$ to $O(N^{5/3})$, and the memory cost is $O(N)$.

V. RANGE TREE ALGORITHM

A two-dimensional range tree for a set point P is an augmented tree. The main tree is a leaf-oriented balanced binary search tree on the x-coordinates of the points in P. The data structure associated with a node v is a leaf-oriented balanced binary search tree on the y-coordinates of the canonical subset of v. A range tree can be extended to higher dimensions in an obvious manner.

A range tree is often combined with fractional cascading technique to further reduce the query time by $O(\log N)$, as seen from Table 1.

The range tree has a superior query time over that of k-d tree, but requires more storage. From Table 1, the building tree time is $O(N(\log N)^{d-1})$, total query time is $O(N(\log N)^{d-1}$, and the memory requirement is $O(N(\log N)^{d-1})$ where $d = m + 1$. Notice that the range tree requires a shorter execution time from order analysis, but the memory requirement is not linear. Applying range tree algorithms to compute multi-scale entropy is straightforward. One needs to replace buildKDTree (point array) and k-dSearch (point array, tree) in Algorithm 2 with buildRangeTree and rangeSearch respectively. Range tree is a mature technology, which can be found in many computational geometry textbooks [5], and computer programs can be found in public domain.

The authors found that the execution time for computing MSE by range search was longer than that of k-d tree for data of interested length (within a hundred thousand). The rationale is as follows: Firstly, $O((\log N)^{d-1}) < O(N^{1-\frac{1}{d}})$ is true only when N is greater than a given threshold. Order analysis only holds when N approaches infinity. In practice, the data length is finite. Secondly, building range trees takes much longer than building k-d trees, and the memory consumption for range trees is not linear. Therefore, it is not practical to apply range search trees to compute S_E for data with a practical length.

VI. EXPERIMENTS

Although range trees require a shorter execution time, the memory requirement is not linear, and is impractical due to DRAM limitations in personal computers and hardware devices when handling large-scale data. Therefore, a k-d tree algorithm was used to demonstrate the effectiveness of the new algorithm in this section. All tests were applied with r = 0.15, match Point (m) = 2, and scale = 1 to 20. The numerical experiment was performed on a 1.5 G Hz Intel CPU with 1.5 G RAM. The code is written in C language.

Test 1: To test the performance of k-d tree compared with algorithms proposed in other studies

The authors compare the performance of the new algorithm to the recent bucket-assisted algorithm [14]. This measures the factor of execution time improvement for both algorithms over the brute force algorithm. In [14], the author computes A_E instead of S_E. Computing S_E and A_E consumes approximately the same computational resources as seen from Eq. (3) and Eq. (4). RR (vardiac interbeat interval) time series (Fig. 2), ECG signal (Fig. 3), and EEG signal (Fig. 4) are tested in this experiment. ECG and EEG data are both measured from hardware with 256 resolutions (8 bits). RR time series are extracted from the ECG signal. The parameters

Test 1.1 RR time series

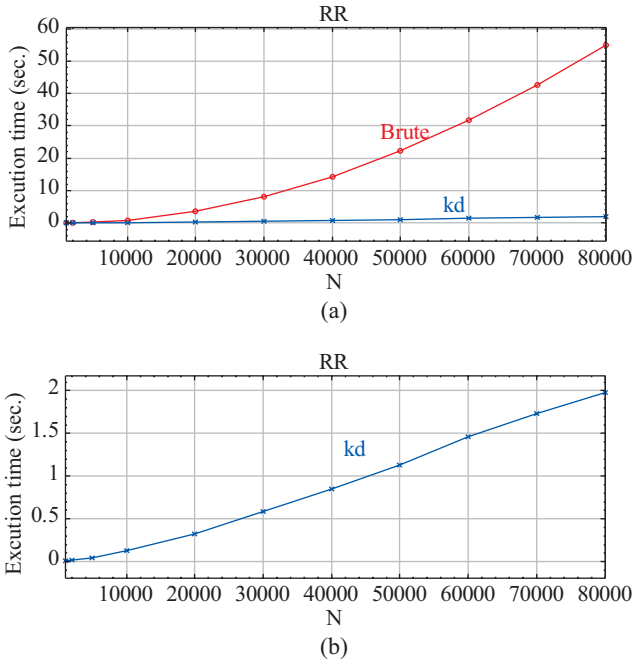

Fig. 2. (a) Execution times versus *N* for RR interval. Red circle line represents execution times of the brute force algorithm, blue cross line represents the corresponding values for the *k*-d tree algorithm. (b) Execution time versus *N* for RR interval for the *k*-d tree algorithm.

Test 1.2 ECG signal

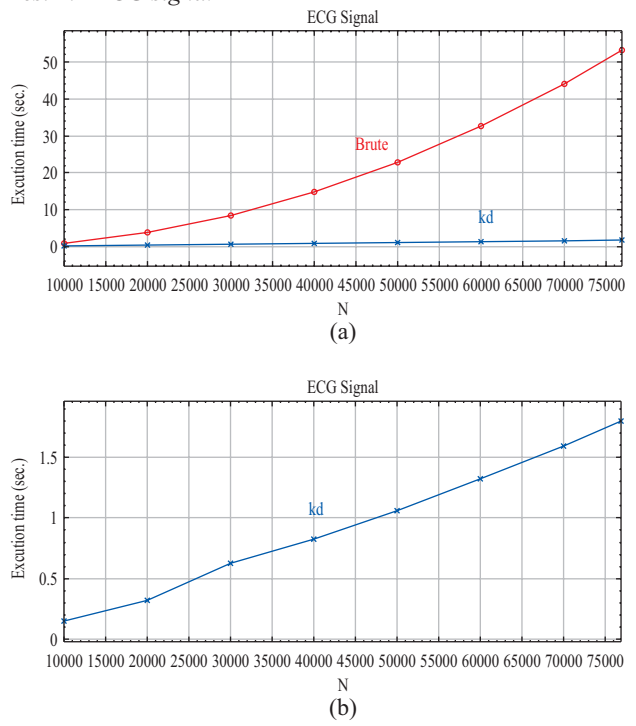

Fig. 3. (a) Execution times versus *N* for ECG signal. Red circle line represents the execution time of the brute force algorithm, blue cross line represents the corresponding values for the *k*-d tree algorithm. (b) Execution time versus *N* for ECG signal for the *k*-d tree algorithm.

Test 1.3 EEG signal

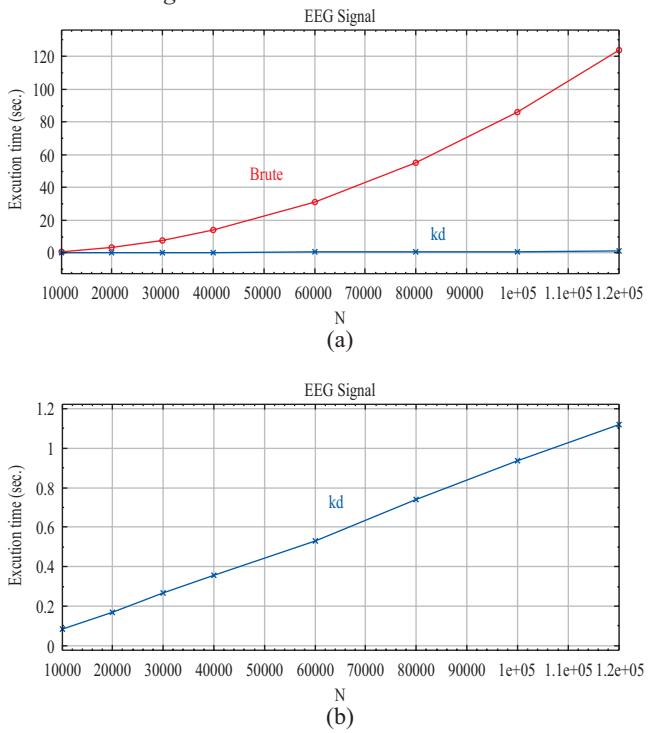

Fig. 4. (a) Execution time versus *N* for EEG signal. The red curved line represents the execution time of the brute force algorithm, and the blue cross line represents the corresponding values for the *k*-d tree algorithm. (b) Execution time versus *N* for EEG signal for the *k*-d tree algorithm.

used in this experiment were $m = 2$ and scale = 1. The execution time versus N is plotted in Fig. 2 for RR interval. It is obvious from Fig. 2(a), the execution time of *k*-d tree was much better than that of brute force algorithm. To focus on the execution time of *k*-d tree, the execution times of *k*-d tree algorithms were enlarged in Fig. 2(b). As seen from [14], the bucket-assisted algorithm improved the brute force algorithm by a factor of less than 5 times, while *k*-d tree improved the brute force algorithm by a factor of 20 times for N = 80,000. Figure 2(b) shows that the *k*-d tree algorithm performed much better than $O(N^{5/3})$, and it performed like $O(N)$. This was probably because the execution time depended on the nature of the signal, and time complexity analysis only predicted the worst case. Similar results were obtained for ECG signal and EEG signal. The *k*-d tree algorithm improved the brute force algorithm by a factor of 30 times for ECG signal for N = 80,000, and a factor of 70 times for EEG signal for N = 80,000 as shown in Fig. 3(a) and Fig. 4(a).

Test 2: Pink Noise (1/f)

To study the long-term correlations in environmental time series [10, 13, 18] and other applications, 1/*f* noise was tested in this example. Data lengths ranging from 100 to 1.6 million were used to test S_E versus N. Figure 5 shows S_E oscillating

Table 2. Time cost list for pink noise.

Number of Points	Brute Force (s)	3D *k*-d Tree (s)
1,000	0.020	0.05
2,000	0.060	0.05
4,000	0.290	0.13
8,000	0.921	0.29
20,000	5.390	1.01
50,000	32.400	4.18
10^5	128.400	15.16
2×10^5	501.000	51.30
4×10^5	2063.000	176.70
8×10^5	8716.000	655.00
16×10^5	too long	2263.00

Table 3. Comparison of brute force and *k*-d tree algorithm used to test an overnight EEG signal.

Case	Length	Brute Force (s)	3D *k*-d Tree (s)
1	7.8×10^6	~10 days	14529.7

(Courtesy of Computer Science and Information Engineering, National Cheng Kung University, Taiwan)

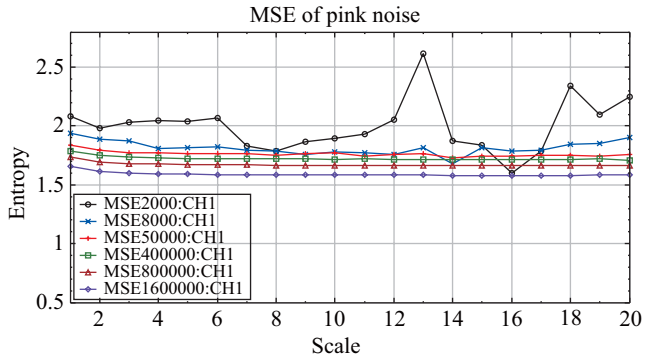

Fig. 5. S_E vs. scale for different N.

between different scales for small N. The computational result converged with the analytical results as N increased, as described in [7]. The execution time for *k*-d tree and brute force algorithm are listed in Table 2.

Test 3: EEG signal

The purpose of this experiment was to test the performance of the *k*-d tree algorithm in handling large N. MSE for scale from 1 to 20, and $m = 2$, was applied to overnight EEG signals. The signals were partitioned into windows for practicality sake. The execution time for the two algorithms is shown in Table 3. A significant improvement for *k*-d tree algorithm is clearly seen in Table 3.

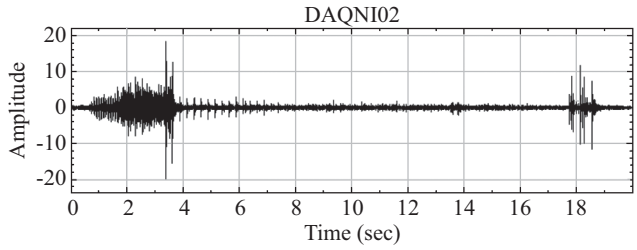

Fig. 6. Cutting process: raw data.

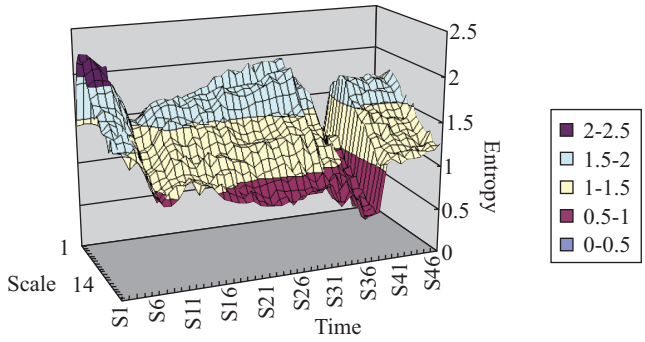

Fig. 7. MSE result of the cutting process.

Test 4: Mechanical Problem

MSE was applied to monitor tool life in machines. Entropy, which was superior to RMS or Kurtosis [21], was used as an indicator for monitoring tool life. In this example, MSE with a rolling window monitored the tool life. The experiment setup was a sampling rate of 10 kHz, the recorded data is 400,000 points. The window size is 0.2 seconds (2,000 points) with 50% overlap, $m = 2$ and scale = 1 to 20 is used in this experiment. The entire machine process was measured by microphone. Raw data is shown in Fig. 6 and the results of MSE with a rolling window during 0~4.6 second are shown in Fig. 7. It shows the machine started at S6 (where S6 stands for $0.1 \times 6 = 0.6$ second), and the tool wore out during S6~S36. The lowest value of MSE appeared at time S36, and the tool broke at time S36.

Online monitor of machining was achieved by applying the *k*-d tree algorithm. From the numerical experiments given above, the *k*-d tree algorithm showed the greatly improved computation time.

VII. SUMMARY

In this section, the execution time and the performance of the newly developed *k*-d tree algorithm was tested. For all testing, the *k*-d tree algorithm improved the execution time from 10 to 70 times faster compared to conventional brute force method for $N = 80{,}000$. Tests 2 and 3 showed that improvement increased with N.

The time complexity of the newly developed *k*-d tree algorithm was $O(N^{5/3})$, which represented an improvement over

the brute force algorithm by $O(N^{1/6})$. From the Fig. 2(b), Fig. 3(b), and Fig. 4(b), it appears that the execution time was proportional to N, which was linear and performed much better than predicted in the order analysis. At present, the authors do not understand the reason. It is possible that order analysis only predicted the worst case.

Long data is often partitioned into windows, as in Test 4, and the execution time was reduced. The length of the window depended on sample frequency, and the nature of the signal. However, the execution time for the conventional brute force algorithm was still too slow for application with real world data such as long-term correlation $1/f$ noise, as in Test 2; or high sample rated mechanical applications, as in Test 4. Since the execution time was significantly reduced by the newly developed k-d tree algorithm, one could compute the MSE with a much longer data set. By collecting additional data and parameters, the statistical meanings of the signal, could be determined.

On-line monitoring of the health of a system has been made possible. Real time computation was achieved in Test 4, where the sample rate was 10 kHz and window size was 2,000 points. For higher sample rates, it would be necessary to limit the size of the window to achieve real time application using the same hardware. This will require further research.

VIII. CONCLUSION

Multi-scale entropy (MSE) is measurement of complexity used to analyze signals in many fields. The time complexity of the algorithms proposed in previous studies required $O(N^2)$, which is too slow for many applications. This research first showed that the probability function in entropy could be transformed into an orthogonal range search problem. A proposed new algorithm was then developed to reduce the computational time to $O(N \log^2 N)$ using $O(N \log^2 N)$ memory, or $O(N^{5/3})$ using linear memory for a typical value, $m = 2$. Experiments using k-d tree algorithm showed significant improvement in execution time from 10 to 70 times faster compared with conventional brute force methods for $N = 80{,}000$. Because the execution time has been significantly reduced, the new algorithm could be applied to online diagnosis, to compute the MSE for long-term correlated signal efficiently. Future research would focus on further improvements in execution time, using linear memory.

ACKNOWLEDGMENTS

The authors would like to thank professor Sheng-Fu Liang in National Cheng Kung University for providing the EEG data. This work was funded in part by the Industrial Development Bureau Ministry of Economic Affairs, Taiwan (ROC).

REFERENCES

1. Agarwal, P. K. and Erickson, J., "Geometric range searching and its relatives," *Advances in Discrete and Computational Geometry*, Vol. 23, pp. 1-56 (1999).
2. Angelini, L., Maestri, R., Marinazzo, D., Nitti, L., Pellicoro, M., Pinna, G., Stramaglia, S., and Tupputi, S. A., "Multiscale analysis of short term heart beat interval, arterial blood pressure, and instantaneous lung volume time series," *Artificial Intelligence in Medicine*, Vol. 41, pp. 237-250 (2007).
3. Bentley, J. L., "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, Vol. 18, No. 9, pp. 509-517 (1975).
4. Bentley, J. L., "Decomposable Searching Problems," *Information Processing Letter*, Vol. 8, pp. 244-251 (1979).
5. Berg, M. de, Krefeld, M. van, Overmars, M., and Schwarzkopf, O., "Orthogonal range searching," *Computational Geometry: Algorithms and Applications*, 3rd ed., Springer-Verlag, Berlin (2008).
6. Costa, M., Goldberger, A. L., and Peng, C. K., "Multiscale entropy analysis of complex physiologic time series," *Physical Review Letters*, Vol. 89, p. 068102 (2002).
7. Costa, M., Goldberger, A. L., and Peng, C. K., "Multiscale entropy analysis of biological signals," *Physical Review E*, Vol. 71, p. 021906 (2005).
8. Costa, M., Peng, C. K., Goldberger, A. L., and Hausdorff, J. M., "Multiscale entropy analysis of human gait dynamics," *Physica A*, Vol. 330, pp. 53-60 (2003).
9. Guzmán-Vargas, L., Ramírez-Rojas, A., and Angulo-Brown, F., "Multiscale entropy analysis of electroseismic time series," *Natural Hazards and Earth System Sciences*, Vol. 8, pp. 855-860 (2008).
10. Halley, J. M., "Ecology, evolution and l/f noise," *Trends in Ecology and Evolution*, Vol. 11, pp. 33-37 (1996).
11. Halley, J. M. and Kunin, W. E., "Extinction risk and the 1/f family of noise models," *Theoretical Population Biology*, Vol. 56, pp. 215-230 (1999).
12. Kang, X., Jia, X., Geocadin, R. G., Thakor, N. V., and Maybhate, A., "Multiscale entropy analysis of EEG for assessment of post-cardiac arrest neurological recovery under hypothermia in rats," *IEEE Transactions on Biomedical Engineer*ing, Vol. 56, No. 4, pp. 1023-1030 (2009).
13. Lueker, G. S., "A data structure for orthogonal range queries," *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, New York, pp. 28-34 (1978).
14. Manis, G., "Fast computation of approximate entropy," *Computer Methods and Programs in Biomedicine*, Vol. 91, pp. 48-54 (2008).
15. Norris, P. R., Anderson, S. M., Jenkins, J. M., Williams, A. E., and Morris, Jr, J. A." Heart rate multiscale entropy at three hours predicts hospital mortality in 3154 trauma patients," *Shock*, Vol. 30, No. 1, pp. 17-22, (2008).
16. Pincus, S. M., "Approximate entropy as a measure of system complexity," *Proceedings of the National Academy of Sciences, USA*, Vol. 88, pp. 2297-2301 (1991).
17. Reed, D. H., O'Grady, J. J., Brook, B. W., Ballou, J. D., and Frankham, R., "Estimates of minimum viable population sizes for vertebrates and factors influencing those estimates," *Biological Conservation*, Vol. 113, pp. 23-34 (2003).
18. Richman, J. S. and Moorman, J. R., "Physiological time series analysis using approximate entropy and sample entropy," *American Journal of Physiology*, Vol. 278, No. 6, pp. H2039-H2049 (2000).
19. Sugisaki, K. and Ohmori, H., "Online estimation of complexity using variable forgetting factor," *SICE Annual Conference*, pp. 1-6 (2007)
20. Willard, D. E., *Predicate-Oriented Database Search Algorithms*, Harvard University Aiken Laboratory, Harvard, TR-20-78 (1978).
21. Yan, R. and Gao, R. X., "Approximate entropy as a diagnostic tool for machine health monitoring," *Mechanical Systems and Signal Processing*, Vol. 21, pp. 824-839 (2007).