RAPID COMMUNICATIONS

PHYSICAL REVIEW E VOLUME 48, NUMBER 5 NOVEMBER 1993

Fast $O(N)$ box-counting algorithm for estimating dimensions

T. C. A. Molteno
*Physics Department, Otago University, Dunedin, New Zealand**
(Received 21 July 1993)

A successive-partitioning algorithm that performs box counting from a time series with computation time and storage both of order N is presented. This enables fast evaluation of generalized dimensions on small computers.

PACS number(s): 02.70.−c, 05.45.+b

I. INTRODUCTION

One method of characterizing the complex structure of a chaotic trajectory is to estimate the dimension of the corresponding strange attractor [1]. Algorithms for estimating dimension abound; the algorithm of Grassberger for estimating the correlation dimension [2] is computationally very efficient, requiring storage of $O(N)$ and computation time $O(N)+O(D)$ for N data points and D computed distances. Box-counting estimation of dimension is popular, conceptually very simple [3], and allows the spectrum of generalized dimensions D_q [4] to be estimated from a single computation. To date the best box-counting algorithms for a time series of N data points require an execution time of $O(N\log(N))$ and storage of $O(N)$. I present in this paper a successive-partitioning algorithm which performs box counting from a time series with computation time and storage both of $O(N)$.

Generalized dimensions of a set approximated by N points embedded in d-dimensional space may be estimated by box counting from [5]

$$D_q = \lim_{N\to\infty}\left\{\frac{1}{1-q}\lim_{\epsilon\to 0}\frac{\log\sum_{i=1}^{N(\epsilon)}P_i^q}{\ln\frac{1}{\epsilon}}\right\}. \qquad (1)$$

The purpose of the box-counting algorithm is to calculate $N(\epsilon)$, the number of boxes of edge length ϵ needed to cover the set, and $P_i = \frac{n_i(\epsilon)}{N}$, the occupation probability of the ith box where $n_i(\epsilon)$ is the population of the ith box. D_0 is called the capacity dimension, D_1 the information dimension, and D_2 the correlation dimension. The capacity dimension D_0, for example, is calculated by plotting $\log N(\epsilon)$ versus $\log(\epsilon)$ and determining the slope. This involves evaluating $N(\epsilon)$ for many different values of ϵ, preferably spaced logarithmically ranging from the size of attractor to the smallest useful box size.

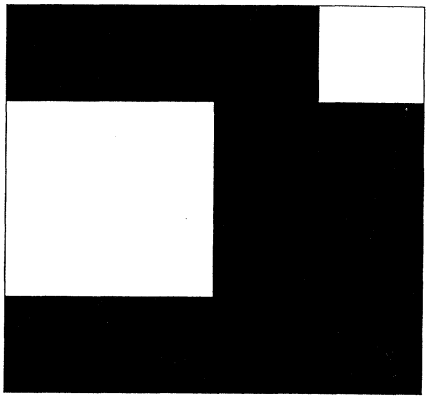
The most straightforward box-counting algorithm for N d-dimensional data points declares a large d-dimensional array and scans the time series, incrementing the appropriate element of the array. This is executed in $O(N)$ time, but sadly the storage required for the array is proportional to ϵ^{-d} and this rapidly becomes unmanageable for accurate dimension estimation (small ϵ) and $d \geq 2$. Several recent papers [6–9] explore methods of circumventing this storage problem using fast $O(N\log(N))$ sorting algorithms [10].

II. SORTING-BASED ALGORITHMS

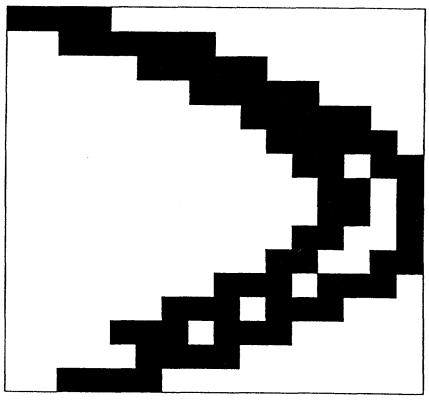
The evaluation of generalized dimensions has been facilitated enormously by the use of fast sorting box-counting algorithms [6,8,7]. The key idea is to take the coordinates of each point of a set embedded in d-dimensional space, rescale them to the interval $[0,2^k-1]$, and express them in binary form. The set is to be covered by a grid of d-dimensional boxes with edge length 2^m $(0 \leq m < k)$. The box to which each point belongs can be found by checking the most significant m bits of each coordinate of that point. To find the number of boxes needed to cover the attractor, the least significant $k-m$ bits of each coordinate are masked and the points sorted in lexicographical order. Then one looks through the sorted list to find the number of distinct values of masked points. This algorithm takes order $N\log(N)$ time using fast sorting procedures (Quicksort or Heapsort [11]). Hou *et al.* [6] improve this algorithm by introducing a new ordering which eliminates resorting in order to count each box size. The memory allocation required for these algorithms is $O(dN)$ and so they can be used with very fine grids.

III. SUCCESSIVE-PARTITIONING ALGORITHM

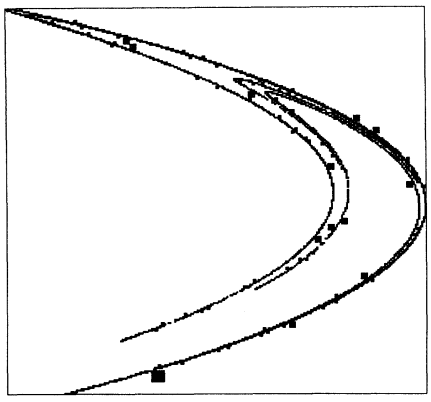
The present work uses a different approach. It involves the repeated application of a simple partitioning procedure to the set, eventually dividing it into a large number of subsets, each covered by a small box. Take a box (in a d-dimensional space) containing all the points in the set. If this box is divided evenly into 2^d boxes of equal size, then these smaller boxes each have a set of points contained within them. This process is then applied recursively; each of the 2^d boxes is subdivided as the original box, creating another 2^{2d} box, each with its attendant set of points. Some of these boxes are not important to this calculation because they are empty, no points of the set fall inside them [see Fig. 1(a)]; these empty boxes are not partitioned any further. If this procedure is repeated m times, the original box will be divided into a grid of 2^{md} boxes [see Figs. 1(b) and 1(c)]. If at each stage of this process the empty boxes are removed from the calculation, then no more than 2^d-1 empty boxes will be stored at any stage. Furthermore if a record is kept



(a)



(b)



(c)

FIG. 1. The successive partitioning of a 10^4-point approximation to the Hénon attractor. (a) After two partitions ($\epsilon = 2^{-2}$), the attractor is approximated by 11 nonempty boxes (black squares) which will be passed on to the successive stages. (b) $\epsilon = 2^{-4}$, $N(\epsilon) = 74$. (c) $\epsilon = 2^{-8}$, $N(\epsilon) = 2211$. Note that some of the boxes are larger than others; these contained one point at a previous stage and were passed over in successive stages.

of the number of boxes, as this process continues, then this will give $N(\epsilon)$ for logarithmically spaced values of ϵ, $\{N(2^{-1}), N(2^{-2}), N(2^{-3}), \ldots, N(2^{-m})\}$, as required to estimate the limit in Eq. (1).

Implementation of the successive-partitioning algorithm revolves around the procedure for partitioning a single box. This procedure is applied recursively to the nonempty sub-boxes until a halting criterion is satisfied (see Sec. IV). Boxes are stored as records, each containing the corner closest to the origin, the edge size, a pointer to the data held in the box, the population, and an array of pointers pointing to the sub-boxes which are created during the partitioning process. The procedure for partitioning a single box divides naturally into three steps. The first is to check the population of the box and calculate the edge size of the new sub-boxes. If the box under consideration only has one point within it, then further partitioning is of no use and the procedure halts. Otherwise the next step is to scan the time series associated with the box to be partitioned and allocate points to the corresponding sub-boxes. This second stage is where the most significant optimizations are found, as this is the section of the code executed most often—once for each element of the time series. The time series is stored as integers and the binary representation of these integers is used to generate an index into the array of sub-boxes. This index is generated by bit shifting and masking instructions which execute very quickly. The contents of the sub-boxes are specified as lists of pointers into the time series using the same system as Grassberger [2]. The final step is to place a null pointer at the end of the contents list of each sub-box.

This single box partitioning procedure takes $O(M)$ time where M is the population of the box. Since the sum of the box populations at any stage is N, any stage in the partitioning process takes a total time of $O(N)$. This means that the whole process of box counting to a grid of say 2^{10} by 2^{10} is $O(N)$, as it requires the application of ten $O(N)$ processes.

IV. HALTING CONSIDERATIONS

When applying the successive-partitioning algorithm to a strange set it is important to stop computation before the box size becomes too small; otherwise $N(\epsilon) \to N$ and the dimension approaches zero, i.e., the partitioning algorithm ought to be halted before the boxes cease to be a good approximation to the covering of the attractor. The point at which this happens is indicated by a deviation from the expected scaling behavior for very small box sizes. For the Hénon attractor, this deviation appears to be related to the average box population $k = \frac{N(\epsilon)}{N}$ (see Fig. 2). The average box population, k, is a very convenient measure of the quality of the approximation because the successive-partitioning procedure can be automatically halted when k falls below a chosen value k_0; a good choice appears to be $k_0 = 10$.

If the partitioning algorithm is continued until the average population falls below some specific value k_0, then the computation time for a set embedded into d dimen-

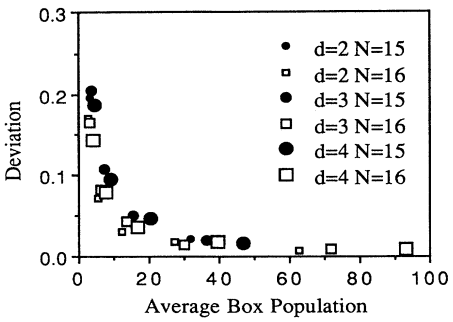


FIG. 2. The fractional deviation from expected scaling behavior caused by approximating the Hénon attractor with a finite set of points is inversely related to the average box population. These data are from 2^{15} and 2^{16} point approximations embedded into 2, 3, and 4 dimensions. The fractional deviation is approximated by $1 - \frac{N(\epsilon)}{\tilde{N}(\epsilon)}$ where $\tilde{N}(\epsilon)$ is the number of boxes in a 2^{20} point approximation to the covering.

sions will no longer be simply $O(dN)$ but will be $O(ldN)$, where l is the number of partitionings that have taken place. If the probability measure is uniform over the set, then the population of all boxes at each stage is uniform. The number of boxes then scales as

$$N(\epsilon) \approx \epsilon^{-D_0} = (2^{-l})^{-D_0} = 2^{lD_0}.$$

If $N(\epsilon)$ is greater than $\frac{N}{k_0}$ (our halting criterion) then the number of partitionings, l, is the first integer greater than $\frac{1}{D_0}\log_2\frac{N}{k_0}$ and is bounded above by

$$l < 1 + \frac{1}{D_0}\log_2\frac{N}{k_0}.$$

Thus the algorithm takes computation time bounded above by $O(dN(1 + \frac{1}{D_0}\log_2(N/k_0)))$. Table I shows a benchmark for the estimation of dimension from a set of randomly generated points using the successive-partitioning scheme. If $l \gg 1$ then the time taken is approximately $O(\frac{d}{D_0}N\log_2(N/k_0))$; for random vectors where $D_0 = d$ for all embedding dimensions the time is independent of embedding dimension. The time per partitioning shows a dependance on the embedding dimension as expected.

TABLE I. Benchmark for the behavior of the successive-partitioning algorithm with different embedding dimensions on a DecStation 3100 computer. The times are for the successive partitioning of 2^{18} points generated at random and embedded into d dimensions. l is the number of partitionings required for the average box population to fall below 10—at which point the procedure halted.

d	t (s)	l	$\frac{t}{l}$
1	15.9	15	1.06
2	12.9	8	1.61
3	10.1	5	2.02
4	10.4	4	2.60
5	9.0	3	3.00
6	14.0	3	4.67

If the probability measure is not uniform, then the computation time is shorter, as not every partitioning will involve all N data points—some will lie in boxes containing a single point and are not to be partitioned further. This is still an improvement over the $O(N\log N)$ Heapsort-based sorting algorithm especially for higher dimensional sets [Heapsort is used, as the worst case performance of Quicksort is $O(N^2)$].

V. DYNAMIC RANGE OF BOX COUNTING

For generalized dimensions estimated using box-counting methods, the maximum possible dimension $d_{\max}$, is given by the expression

$$d_{\max} = \frac{\log N}{-\log\epsilon}.$$

If we add the constraint that the average population is not to fall below some value k, then this expression depends on the probability measure over the attractor. If the probability measure is uniform, i.e., $P_i = \frac{k}{N}$, then the maximum generalized dimension is given by

$$d_{\max} = \frac{\log\left(\frac{N}{k}\right)}{-\log\epsilon}.$$

This should be compared with other dimension estimation techniques, notably the Grassberger algorithm [2], which takes $O(N)$ time (for $q \geq 0$) and has twice the dynamic range [12]. The algorithm of Grassberger becomes less efficient when $q \ll 0$ as the number of distances computed approaches N^2, the computation time scales as $O(N^2)$, and the dynamic range per unit computation time is similar to that of an $O(N)$ box-counting algorithm.

VI. CONCLUSIONS

The fact that this successive partitioning algorithm takes a computation time of $O(N)$ does not make it a better choice than an $O(N\log(N))$ algorithm. It does

TABLE II. Benchmark for box counting of an N point approximation to the Hénon attractor. T_{10} is the time taken to partition to a $2^{10} \times 2^{10}$ grid while T_{tot} is the time taken to partition until the average box population is less than 10; the Grid column is the smallest grid to which the successive partitioning algorithm is counted. The column T_{Hou} is the time taken for the sorting based algorithm of Hou [6]. Both algorithms were implemented in C, compiled and executed on a DecStation 3100 computer. All times are in seconds and do not include the time required to read in and scale the data.

N	T_{10}	T_{tot}	Grid	T_{Hou}
2^{16}	2.6	2.6	1024	19.1
2^{17}	4.9	6.0	2048	40.9
2^{18}	9.6	11.3	4096	86.6
2^{19}	18.9	25.2	8192	192.0
2^{20}	37.7	61.5	16384	451.1

mean that there will be some N, above which successive partitioning is faster. To determine this value, the successive partitioning algorithm is compared with the sorting based algorithm of Hou [6] (see Table II) on the Hénon attractor. The successive-partitioning algorithm is 7–10 times faster than sorting approaches to box counting on small data sets ($N \approx 2^{16}$), and for larger data sets the relative improvement is more dramatic as the computation time is $O(N)$, as opposed to $O(N \log(N))$. Furthermore, the process is automatically halted when the average box population falls below a critical value; the dimension can then be estimated automatically using a least squares linear fit, although convergence is still not guaranteed [13]. Further performance improvements are achievable by implementing this algorithm in parallel. This is easily done because the problem of partitioning a single box creates several smaller versions of the original problem, each of which can be passed onto another processor. Source codes (in C or PASCAL) for the algorithm described in this article are available from the author.

ACKNOWLEDGMENTS

I am indebted to Dr. Xinjun Hou for providing me with a copy of the source code for his algorithm and to Dr. Nick Tufillaro for many useful comments.

---

* Internet: tim@newton.otago.ac.nz

[1] T. Halsey *et al.*, Phys. Rev. A **33**, 1141 (1986).

[2] P. Grassberger, Phys. Lett. A **148**, 63 (1990).

[3] E. Ott, *Chaos in Dynamical Systems* (Cambridge University Press, Cambridge, 1993).

[4] P. Grassberger, Phys. Lett. A **97**, 227 (1983).

[5] P. Grassberger, Phys. Lett. A **97**, 224 (1983).

[6] X.-J. Hou, R. Gilmore, G. Mindlin, and H. Solari, Phys. Lett. A **151**, 43 (1990).

[7] L. Liebovitch and T. Toth, Phys. Lett. A **141**, 386 (1989).

[8] A. Block, W. von Bloh, and H. Schnellnhuber, Phys. Rev. A **42**, 1869 (1990).

[9] F. Ling and G. Schmidt, J. Comput. Phys. **99**, 196 (1992).

[10] L. Meisel, M. Johnson, and P. Cote, Phys. Rev. A **45**, 6989 (1992); **45**, 6996 (1992), present an algorithm with computation time essentially independent of N and storage $O(\epsilon^{-d})$. Their algorithm is designed for data from imaging devices and requires the points to be prepartitioned into boxes of size ϵ [an $O(N)$ process if the data are initially in a time series].

[11] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes: the Art of Scientific Computing* (Cambridge University Press, Cambridge, 1986).

[12] J.-P. Eckmann and D. Ruelle, Phys. D **56**, 185 (1992).

[13] H. Greenside, A. Wolf, J. Swift, and T. Pignataro, Phys. Rev. A **25**, 3543 (1982).