International Journal of Modern Physics C, Vol. 4, No. 3 (1993) 515–523

ON EFFICIENT BOX COUNTING ALGORITHMS

PETER GRASSBERGER
Physics Department, University of Wuppertal, D-5600 Wuppertal 1, Germany

Received 4 September 1992

We present two variants of a fast and storage efficient algorithm for box counting of fractals and fractal measures. In contrast to recently proposed algorithms, no sorting of the data is done. With comparable storage demands, CPU times are between 1 and 2 orders of magnitude lower than those needed with the latter algorithms.

1. Recently, several papers appeared[1–4] which present new box counting algorithms. In these papers it is claimed that previous algorithms for box counting are less efficient. This seems somewhat strange as box counting is usually not considered as a difficult problem, and previous analyses[9–14] had used already rather large statistics (up to 4×10^9 points in ca. 9×10^5 non-empty boxes on a grid of size 28762×28762). Indeed, the authors of Ref. 14 (who did the latter analysis) considered the method straightforward enough so that they gave no details. On the other hand, when comparing the CPU times and storage requirements quoted in Refs. 1–4, it seems clear that it would have been virtually impossible to collect this large statistics with any of these algorithms.[a]

In view of this situation, and at the risk of beating the obvious, it is the purpose of the present note to give some details of a rather simple class of algorithms which seem to be much more efficient than those presented in Refs. 1–4. The basic idea is that described already in Ref. 9. While I do not claim that this gives the fastest algorithm ever possible, it seems to provide a very good compromise between speed, storage requirements, and ease of use.

The main problem in box counting algorithms is that it is trivial to write a very fast algorithm (one just builds a large histogram and scans it afterwards), but this trivial algorithm is in general very inefficient in the use of fast memory. Consider a fractal of dimension D embedded in a Euclidean space of dimension d. If one wants to do box counting with a grid of size L^d, we have to provide an array of $N = L^d$ entries, out of which only $n \approx L^D$ entries contain non-trivial information.

[a]Notice that box counting can have advantages over the more popular neighbor-counting algorithms[5,6] (also called sand-box method by some authors) if one needs very high statistics. Even optimized neighbor-counting algorithms[7,8] need CPU times $\propto$ (data size $\times$ neighborhood size), while box counting can in ideal cases be linear in the data size and independent of the box size.

Thus most of the allocated storage is never used. Even with the virtual memory facilities provided by modern computers this is a serious problem, since the actually needed array elements are scattered all over the array. What we obviously need is an algorithm which allocates only $\mathcal{O}(n)$ storage elements, without too much overhead in addressing these elements and preprocessing the data.

If only the capacity dimension is needed, the above problem is less grave since only one bit (occupied/empty) is needed per lattice site. This can be made use of by "multispin" coding techniques also known from lattice statistical dynamics.[9] The problem is most severe if one is interested in estimating the full spectrum of generalized dimensions D_q[6,15] or the equivalent $f(\alpha)$ function.[16–18]

In this connection we should point out that box counting problems for fractals come essentially in two variants:

(a) In the first case, all points of the fractal are known. This is the case, e.g., for percolation or DLA clusters.[19] Eventually, such a fractal is already defined on a lattice. In this case, each lattice site is occupied by at most one point, and the finest grid used for box counting is just this lattice. Even if the cluster is grown off-lattice, the natural choice of the finest grid is such that non-empty boxes contain $\mathcal{O}(1)$ points.

(b) This is very different from the situation where only a random sample of points on the fractal is known. This is typically the case in strange attractors and repellers (including, e.g., Julia sets), and in any measure concentrated on a fractal support. An example of the latter is the electrostatic charge distribution on a fractal surface as estimated from the hitting frequencies of random walkers.[19] In this case, one needs very many points per non-empty box.

It is mainly the second case which we are interested in. Our algorithms can be used also for case (a), but the advantage over the algorithms of Refs. 1–4 is largest for case (b).

In Refs. 1, 3 the finest grid was chosen such that each non-empty box contained on the average $\langle m \rangle = 5$ points. For the attractor of the Hénon map

$$x_{n+1} = a - y_n - x_n^2 , \quad y_{n+1} = x_n \tag{1}$$

studied in these papers, and for the accuracy ± 0.01 quoted in Ref. 1, this is certainly not enough. If all boxes had the same weight p_i, then the distribution of the number of points per box would be Poissonian (in the limit of many non-empty boxes), and $\langle m \rangle = 5$ would be enough for estimating the capacity dimension D_0. But the weights of boxes are unequal for two reasons: because of multifractality and because it might happen that a non-empty box intersects with the fractal only very little.[10] Both effects together lead to a slow power-like convergence of the number of non-empty boxes with the number of points,[9–11] instead of the exponential convergence expected for equal weights.

Similar systematic errors arise also for the other generalized dimensions if the number of random points is not sufficient.[10,20] For the information dimension, the leading error is, e.g., simply $\delta D_1 = -n/2N$, where n is the number of non-empty boxes and N is the number of points.[10] In the numbers shown below (Fig. 1), these errors are already corrected.

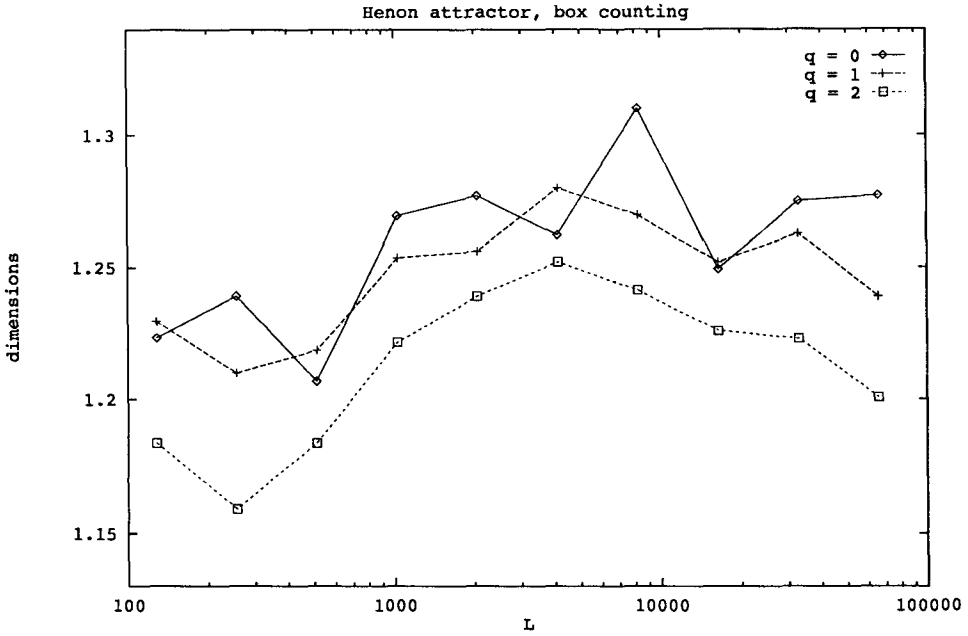

Fig. 1. Estimates of the generalized dimensions D_q with $q = 0, 1, 2$ for the Hénon attractor ($a = 1.4$, $b = .3$), plotted semi-logarithmically against the grid size L.

2. A complete FORTRAN routine for measuring the dimensions D_0, D_1 and D_2 of the Hénon attractor is given in listing 1. The output produced by this routine are the Renyi entropies

$$H_q = \frac{1}{1-q} \log \sum_i p_i^q \,. \tag{2}$$

The sum runs over all non-empty boxes. The generalized dimensions are obtained from it as $D_q = \lim_{L\to\infty} H_q / \log L$. For $q = 1$, Eq. (2) is replaced by its limit for $q \to 1$. The finite-N corrections mentioned above are already applied for $q = 1, 2$. To apply these corrections also to D_0, one has to perform part 3 of the listing not only at the end of the calculation but also at intermediate times.

The basic trick for avoiding unnecessary storage allocations is to use a coarse grid in addition to the fine grid used for measuring the weights p_i. To each box of the coarse grid is associated an element of an array (called "COARSE" in listing

1). The elements of this array are set to zero at the beginning. At the end of the computation they are still zero if no point had fallen into the corresponding box. Otherwise, they contain pointers to elements of another array (called "FINE" in listing 1) which then contains further information about the occupancy of that part of the fine grid which is inside the coarse box. In previous test runs, the rough number of non-empty coarse boxes have to be determined (so that an upper limit to the size of FINE can be fixed beforehand), but the construction of the pointers is done 'on the fly'. Notice that we have taken the size of the array COARSE somewhat larger than strictly needed, to allow for points which fall slightly outside the nominal bounds due to round-off errors.

The time complexity of the first two parts of the algorithm is obviously $\mathcal{O}(N)$ for N data points. The time needed for the last part (reading the non-empty boxes) is $\mathcal{O}(n \cdot \beta^{d-D_0})$. Here, n, D_0 and d are again the number of non-empty boxes and the capacity resp. embedding dimension, and β is the scale factor between the coarse and fine grid (in listing 1, $d = 2$ and $\beta = 8$). In the limit of interest to us ($N \gg n \gg 1$), the time needed for the second part dominates the overall CPU time.

The routine as listed in listing 1 needed 53 sec on a SUN SPARCserver 670MP (ca. 28.5 MIPS), for 10^7 points in 33369 different boxes. In comparison to this, the fastest sorting-based algorithm published so far[4] needed 417 sec for 8×10^6 points on a transputer system with 8 transputers (in total ca. 80 MIPS). The grid size for this was not quoted, but the algorithm of Ref. 4 should depend little on it. We see thus that our algorithm is faster by a factor of ca. 28. Since the time complexity of the sorting used in Ref. 4 is $\mathcal{O}(N \log N)$, this factor should further increase for larger N.

For most applications of box counting, it is not the CPU time but the storage which is critical. The storage requirements of listing 1 can be estimated for arbitrary L (=grid size), n, D_0, d, β as follows. For the array COARSE we need $(L/\beta)^d$ words, while we need $\beta^d \cdot a(L/\beta)^{D_0}$ words for the array FINE. Here we have assumed that $n \approx aL^{D_0}$, with a constant a of order unity. The total memory ("space complexity") needed is thus

$$S = (L/\beta)^d + a\beta^d(L/\beta)^{D_0}\,. \tag{3}$$

Since the CPU time is essentially independent of β, we can minimize S without penalty by choosing an appropriate β. Setting $\partial S/\partial\beta$ to zero, we obtain

$$\beta_{\text{opt}} = \left[\frac{d}{d-D_0}\right]^{1/(2d-D_0)} L^{(d-D_0)/(2d-D_0)} \tag{4}$$

(for efficient programming, β should of course be the nearest power of 2) and

$$S(\beta_{\text{opt}}) \propto L^{d^2/(2d-D_0)} \propto n^{d/(2d-D_0)}\,. \tag{5}$$

This is not yet the desired scaling $S \propto n$, but it is already much better than the naive histogram method. For the parameters of listing 1, the latter would have

needed $L^2 = 4 \times 10^6$ words, while listing 1 needs less than 2.3×10^5 words.

3. For an algorithm with the optimal behavior $S \propto n$, we have just to repeat the basic trick used in listing 1. Instead of using only one coarse and one fine grid, we use a whole hierarchy of nested grids. The contents of the array COARSE (corresponding to the coarsest grid) act now as pointers to another set of pointers corresponding to the second-coarsest grid. The latter point again to pointers etc. Thus we have a chain of pointers, only the last of which point to the information about the finest grid. The advantage of this is that now the factor β between successive levels can be much less than above.

If we assume that the coarsest grid has $\mathcal{O}(1)$ boxes, the space complexity now becomes

$$S = a\beta^d (L/\beta)^{D_0}(1 + \beta^{-D_0} + \beta^{-2D_0} + \cdots) = \frac{\beta^d}{\beta^{D_0} - 1}\, n\,. \tag{6}$$

Since for each point one needs to follow a chain of $\log L/\log\beta$ pointers, the time complexity now is essentially $\propto N \log L/\log\beta$. So we have found an algorithm with both memory and CPU time $\propto N$. Indeed, CPU time can be reduced considerably without too much increase of S by taking the coarsest grid sizes much smaller than L but $\gg 1$. For the Hénon map with $L \approx 2^{12}$ to 2^{16}, we got good results with $\beta = 2$ and a coarsest grid of size $\approx \sqrt{L}$.

An implementation of this algorithm, again for the Hénon map, is given in listing 2. There, the finest grid has $L = 2048$, and the auxiliary grids have sizes from 64 to 1024. To simplify programming, all information about the auxiliary grids except the coarsest one are written into the same array. On the same SUN workstation mentioned above, the routine needed 156 sec. Compared with listing 1, this is a factor 3 slower, but the storage requirements are cut down from 2.3×10^5 to 1.0×10^5 words.

In both listings, only the Renyi entropies for the finest length scale is computed. In both cases it would be easy to compute also the entropies on coarser scales, without much additional time. We did not include the corresponding codes since they are fairly straightforward but somewhat lengthy. In listing 1, we have to construct coarsened histograms by lumping together the entries of array FINE. In listing 2, we have to access array FINE in the third part not directly but indirectly through the same chains of nested boxes as in part 2.

The saving in storage is of course even larger for larger lattices. With 32 MB of fast memory, we could run listing 2 with L up to 2^{16}. The number of non-empty boxes after 3.3×10^9 iterations of the Hénon map was in that case $n = 2773664$. The resulting dimensions are shown in Fig. 1. More precisely, the quantities shown in this figure are estimates

$$D_q(L) = \frac{H_q(L) - H_q(L/2)}{\log 2}\,. \tag{7}$$

We can see in Fig. 1 the oscillations (or rather fluctuations) noticed first in Ref. 9 (similar oscillations for the Zaslavski map had previously been seen in Ref. 5, and were discussed there in detail; see also Refs. 21, 13, 20). These oscillations are much bigger than statistical errors which are smaller than the sizes of the symbols for large L.

4. Summarizing we have seen that using auxiliary grids can be much more efficient for box counting than algorithms using comparison based sorts. This might not come as a surprise to some researchers since, contrary to widespread belief, comparison based sorting algorithms like heap sort or quicksort (which have average time complexity $\mathcal{O}(N \log N)$ are *not* the fastest ones. It is well known in principle (but largely neglected by the community) that box- (or "radix"-) assisted methods can be $\mathcal{O}(N)$.[22,23] A very short explicit FORTRAN routine for ranking real numbers which is much faster than quicksort is given in Ref. 24 (see also Ref. 8). Thus it should not be surprising that box-assisted methods are superior in cases where a grid structure is already employed anyhow.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft, SFB 237.

References

1. L. S. Liebovitch and T. Toth, *Phys. Lett.* **A141**, 386 (1989).
2. A. Block, W. von Bloh and H. J. Schellnhuber, *Phys. Rev.* **A42**, 1869 (1990).
3. X.-J. Hou, R. Gilmore, G. B. Mindlin, and H. G. Solari, *Phys. Lett.* **A151**, 43 (1990).
4. S. Goshen and R. Thieberger, *Int. J. Mod. Phys.* **C3**, 267 (1992).
5. P. Grassberger and I. Procaccia, *Physica* **D9**, 189 (1983).
6. P. Grassberger, *Phys. Lett.* **A97**, 227 (1983).
7. P. Grassberger, *Phys. Lett.* **A148**, 63 (1990).
8. T. Schreiber, "Efficient search for nearest neighbors", preprint, 1992.
9. P. Grassberger, *Phys. Lett.* **A97**, 224 (1983).
10. W. E. Caswell and J. A. Yorke, "Invisible errors in dimension calculations: Geometric and systematic effects", in *Dimensions and Entropies in Chaotic Systems*, ed. G. Mayer-Kress (Springer, 1986), p. 123.
11. R. Badii and A. Politi, *Phys. Lett.* **A101**, 182 (1984).
12. A. Giorgilli, D. Casati, L. Sironi, and L. Galgani, *Phys. Lett.* **A115**, 202 (1986).
13. A. Arneodo, G. Grasseau and E. J. Kostelich, *Phys. Lett.* **A124**, 426 (1987).
14. P. Grassberger, R. Badii and A. Politi, *J. Stat. Phys.* **51**, 135 (1988).
15. H. G. E. Hentschel and I. Procaccia, *Physica* **D8**, 435 (1983).
16. G. Parisi, appendix *in* U. Frisch, "Fully developed turbulence and intermittency", in *Proc. of Int. School on Turbulence and Predictability in Geophysical Fluid Dynamics and Climate Dynamics*, ed. M. Ghil (North-Holland, 1984).
17. T. C. Halsey, M. H. Jensen, L. P. Kadanoff, I. Procaccia, and B. I. Shraiman, *Phys. Rev.* **A33**, 1141 (1986).
18. G. Paladin and A. Vulpiani, *Phys. Reports* **156**, 147 (1987).
19. J. Feder, *Fractals* (Plenum, New York, 1988).
20. P. Grassberger, *Phys. Lett.* **A128**, 369 (1988).

21. R. Badii and A. Politi, *Phys. Lett.* **A104**, 303 (1984).
22. D. Knuth, *The Art of Computer Programming*, Vol. 3 (Addison-Wesley, 1973).
23. T. Ottmann and P. Widmayer, *Algorithmen und Datenstrukturen* (BI-Verlag, Mannheim, 1991).
24. P. Grassberger, T. Schreiber and C. Schaffrath, *Int. J. Bifurc. Chaos* **1**, 521 (1991).

Listing 1:

```
      parameter (a=1.4,b=.3, kmax=10**7, isize=2048)
      real*8 x,y,z,x0,y0,xmin,xmax,scale,h0,h1,h2
      integer coarse(-1:256,-1:256),fine(0:63,2500)
      data x,y,xmin,xmax,h1,h2/6*0./
      data coarse,fine,non_empty,nbox/226566*0/

c (1) Get range of x values and scale factor for discretization:

      do 1 k=-100,kmax
         z=a+b*y-x*x
         y=x
         x=z
         if(k.eq.0)  x0=x
         if(k.eq.0)  y0=y
         if ((x.lt.xmin).and.(k.gt.0))  xmin=x
  1      if ((x.gt.xmax).and.(k.gt.0))  xmax=x
      scale=isize/(xmax-xmin)

c (2) Iterate once again, and put points into boxes:

      x=x0
      y=y0
      ix=(x-xmin)*scale
      ixg=rshift(ix,3)
      ixf=ix.and.7
      do 2 k=1,kmax
         iyg=ixg
         iyf=ixf
         z=a+b*y-x*x
         y=x
         x=z
         ix=(x-xmin)*scale
         ixg=rshift(ix,3)
         ixf=ix.and.7
         nb=coarse(ixg,iyg)
         if (nb.eq.0) then
             nbox=nbox+1
             coarse(ixg,iyg)=nbox
             nb=nbox
         endif
         ixyf=ixf+lshift(iyf,3)
  2      fine(ixyf,nb)=fine(ixyf,nb)+1
```

```
c (3) Count points, compute & print Renyi entropies:

      do 3 nb=1,nbox
         do 3 i=0,63
            f=fine(i,nb)
            if (f.gt.0) then
               non_empty=non_empty+1
               h1=h1+f*log(f) -.5
               h2=h2+f*(f-1)
            endif
  3         continue
      h0=log(real(non_empty))
      h1=log(real(kmax))-h1/kmax
      h2=2*log(real(kmax))-log(h2)
      print *,'Renyi entropies for q=0,1,2:',h0,h1,h2
      end
```

Listing 2:

```
      parameter (a=1.4,b=.3, kmax=10**7)
      parameter (icoarse=64,ib=5,isize=icoarse*2**ib)
      real*8 x,y,z,xmin,xmax,scale,h0,h1,h2
      integer coarse(-1:icoarse,-1:icoarse)
      integer fine(0:3,14000),box(0:3,10000)
      data x,y,xmin,xmax,h1,h2/6*0./,nbox,nfine/2*0/
      data coarse,box,fine/100356*0/,non_empty/0/

c (1) Get range of x values, and scale factor for discretization:

      ... (same as listing 1)

c (2) Iterate once again, and put points into boxes:

      ix=(x-xmin)*scale
      ixc=rshift(ix,ib)
      do 2 k=1,kmax
         iy=ix
         iyc=ixc
         z=a+b*y-x*x
         y=x
         x=z
         ix=(x-xmin)*scale
         ixc=rshift(ix,ib)
         nn=coarse(ixc,iyc)
         if (nn.eq.0) then
             nbox=nbox+1
             coarse(ixc,iyc)=nbox
             nn=nbox
         endif
```

```
         do 22 m=ib-1,2,-1
            n=nn
            ii=(rshift(ix,m+1).and.2)+(rshift(iy,m).and.1)
            nn=box(ii,n)
            if (nn.eq.0) then
               nbox=nbox+1
               box(ii,n)=nbox
               nn=nbox
            endif
 22         continue
         ii=(ix.and.2)+(rshift(iy,1).and.1)
         nf=box(ii,nn)
         if (nf.eq.0) then
            nfine=nfine+1
            box(ii,nn)=nfine
            nf=nfine
         endif
         ii=(lshift(ix,1).and.2)+(iy.and.1)
  2      fine(ii,nf)=fine(ii,nf)+1

c (3) Count points, compute & print Renyi entropies:

      do 3 nb=1,nfine
         do 3 i=0,3
            f=fine(i,nb)

         ...  (same as listing 1)

      end
```