International Journal of Bifurcation and Chaos, Vol. 20, No. 12 (2010) 4067–4077

DOI: 10.1142/S0218127410028197

OPTIMAL DYNAMIC BOX-COUNTING ALGORITHM

PANAGIOTIS D. ALEVIZOS* and MICHAEL N. VRAHATIS†
Computational Intelligence Laboratory, Department of Mathematics, University of Patras Artificial Intelligence Research Center, University of Patras, GR-26110 Patras, Greece
**alevizos@math.upatras.gr*
†*vrahatis@math.upatras.gr*

Received April 1, 2008; Revised July 6, 2010

An optimal box-counting algorithm for estimating the fractal dimension of a nonempty set which changes over time is given. This nonstationary environment is characterized by the insertion of new points into the set and in many cases the deletion of some existing points from the set. In this setting, the issue at hand is to update the box-counting result at appropriate time intervals with low computational cost. The proposed algorithm tackles the dynamic box-counting problem by using computational geometry methods. In particular, we use a sequence of compressed Box Quadtrees to store the data points. This storage permits the fast and efficient application of our box-counting approach to compute what we call the "dynamic fractal dimension". For a nonempty set of points in the d-dimensional space $\mathbb{R}^d$ (for constant $d \geq 1$), the time complexity of the proposed algorithm is shown to be $O(n \log n)$ while the space complexity is $O(n)$, where n is the number of considered points. In addition, we show that the time complexity of an insertion, or a deletion is $O(\log n)$, and that the above time and space complexity is optimal. Experimental results of the proposed approach illustrated on the well-known and widely studied Hénon map are presented.

Keywords: Box-counting; fractal dimension; quadtree; Hénon map.

1. Introduction

The box-counting dimension (capacity or "packing" dimension) is widely used for determining the fractal dimension of a nonempty set $\mathcal{V}$ in a metric space. In fractal geometry, this dimension is also known as the Minkowski–Bouligand dimension, or Minkowski dimension. To compute the box-counting dimension for a nonempty set $\mathcal{V}$, overlay the d-dimensional space that embeds $\mathcal{V}$ with an evenly-spaced grid of d-dimensional boxes with sides of length ε, and count the number of boxes $N(\varepsilon)$ that are required to cover $\mathcal{V}$. The box-counting dimension $\dim_{\mathrm{B}}(\mathcal{V})$ for the set $\mathcal{V}$ is then determined by checking how $N(\varepsilon)$ changes as ε becomes smaller. The box-counting dimension is well defined only if the following limit exists:

$$\dim_{\mathrm{B}}(\mathcal{V}) = -\lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log \varepsilon}. \tag{1}$$

Otherwise, the lower limit corresponds to the lower box-counting dimension called the lower Minkowski dimension, while the upper limit corresponds to the upper box-counting dimension called the upper Minkowski dimension (also known as entropy dimension, Kolmogorov dimension, or Kolmogorov capacity).

In [Liebovitch & Toth, 1989] a *static* box-counting algorithm to determine the fractal

dimension of a data set $\mathcal{V}$ in d dimensions has been proposed. The coordinates of each point of $\mathcal{V}$ are rescaled in the interval $[0, 2^k - 1]$ and are expressed in binary form. Thus, the coordinates are considered integer numbers and each number is stored in a word of k bits. The set is covered by a grid of d-dimensional boxes with edge size 2^m $(0 \leq m \leq k-1)$. With the binary representation of coordinates, the box to which a point belongs can be found by checking the most significant m bits (left m bits) of each coordinate of that point. To find the number of boxes needed to cover the set $N(2^m)$, the right $k-m$ bits of each coordinate are masked to 0's and the points are sorted into lexicographical order in $O(n \log n)$ time. However, the lexicographical ordering of points requires a new sorting for each box size, and the time complexity becomes $O(kn \log n)$. The space complexity is $O(n)$. To avoid the lexicographical ordering, an efficient way to order the points in any embedding dimension space has been proposed in [Liebovitch & Toth, 1989] (based on a suggestion by D. Kaplan). This approach allows for a faster implementation of box-counting algorithms, as has been fully described in [Hou *et al.*, 1990] and implemented in [Kruger, 1996].

More specifically, in [Hou *et al.*, 1990] the time complexity of the box counting method is improved by using a new topological ordering of points. Instead of sorting the points into lexicographical order directly, this method intercalates the coordinates of each point into a long bit string of length $d\,k$ and sorts the points according to the value of intercalated bit string in $O(n \log n)$ time. Once the intercalated points are sorted, the most significant d bits determine the box which contains that point. All points in the same box are always in one segment, independent of the box size. In order to count the number of nonempty boxes the algorithm finds the number of distinct values in the ordered list of points. This gives the number of boxes of the smallest possible size 2^0 required to cover the set $N(2^0)$. Then d bits on the right are masked to 0's to create a list of boxes of size 2^1, then d more bits from the right are masked for the next box size 2^2, and so on. Each time d bits are masked, and a list is prepared in $O(n)$ time which counts the number of boxes required of a bigger box size. Thus, in $O(n \log n)$ time the algorithm gets the number of boxes needed to cover the set for box sizes ranging between 2^0 and 2^{k-1}.

Obviously, the box counting method of [Hou *et al.*, 1990] is a static method and works efficiently for a fixed data set of n points. In a dynamic data set however, insertions and deletions of points change the sorted sequence of the long bit strings, and the algorithm must restart the counting procedure, in order to update the number of boxes needed to cover the set for box sizes ranging between 2^0 and 2^{k-1}. Therefore, after an insertion or deletion in the data set, the method in [Hou *et al.*, 1990] updates the fractal dimension in $O(n \log n)$ time.

The purpose of our contribution is to introduce the *dynamic fractal dimension* of a set which changes over time by inserting new points into the set and in many cases by deleting some existing points from the set. To estimate the dynamic fractal dimension we employ computational geometry methods and propose an optimal box-counting algorithm which we call *dynamic box-counting algorithm*. Furthermore, we show that the proposed algorithm possesses an optimal time and space complexity.

The rest of the paper is organized as follows. In Sec. 2, we present the box-counting algorithm along with its complexity issues. In Sec. 3, we give a method to determine the fractal dimension of a nonempty set and we show that the proposed dynamic box-counting algorithm possesses an optimal time and space complexity. In Sec. 4, we present experimental results of our approach illustrated on the well-known and widely studied Hénon map. The paper ends in Sec. 5 with a synopsis.

2. The Box-Counting Algorithm

Next, we present our approach. Let $\mathcal{V} = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in d-dimensional space, for constant $d \geq 1$, with coordinate axes $(O_{x_1}, O_{x_2}, \ldots, O_{x_d})$, and let $p_i = (x_1^i, x_2^i, \ldots, x_d^i)$ be the representation of any point $p_i \in \mathcal{V}$. We assume that the coordinates of each point in $\mathcal{V}$ are real numbers. Each point $p_i = (x_1^i, x_2^i, \ldots, x_d^i)$ is intercalated into a long string with digits of the set $\{0, 1, \ldots, 9\}$. We show this intercalation procedure in $d = 2$ dimensions: let $p = (x, y)$ be a point in $\mathbb{R}^2$ and let

$$x = \langle X_{a-1} X_{a-2} \cdots X_1 X_0 \rangle \langle X'_{b-1} X'_{b-2} \cdots X'_1, X'_0 \rangle,$$

be the form of the decimal number x, where $\langle X_{a-1} X_{a-2} \cdots X_1 X_0 \rangle$ $(X_q \in \{0, 1, \ldots, 9\}, 0 \leq q \leq a-1)$ is the integer part of x and $\langle X'_{b-1} X'_{b-2} \cdots X'_1 X'_0 \rangle$ $(X'_r \in \{0, 1, \ldots, 9\}, 0 \leq r \leq b-1)$ is the decimal part of x; and similarly for y:

$$y = \langle Y_{c-1} Y_{c-2} \cdots Y_1 Y_0 \rangle \langle Y'_{e-1} Y'_{e-2} \cdots Y'_1 Y'_0 \rangle.$$

If $a \neq c$ and/or $b \neq e$, then we complete with 0 as the appropriate number, so that both numbers X and Y have $a+c+|a-c|$ digits in their integer part, and $b+e+|b-e|$ digits in their decimal part. For example, if $a < c$ and $b > e$ we have,

$$x = \langle \underbrace{0 \cdots 0}_{c-a} X_{a-1} X_{a-2} \cdots X_1 X_0 \rangle \langle X'_{b-1} X'_{b-2} \cdots X'_1, X'_0 \rangle,$$

$$y = \langle Y_{c-1} Y_{c-2} \cdots Y_1 Y_0 \rangle \langle Y'_{e-1} Y'_{e-2} \cdots Y'_1 Y'_0 \underbrace{0 \cdots 0}_{b-e} \rangle.$$

The intercalated point of x and y is the real number $\mathcal{N}_{xy} = \mathcal{I}_{xy}\mathcal{D}_{xy}$, with an integer part $\mathcal{I}_{xy}$ consisting of $2c$ digits,

$$\mathcal{I}_{xy} = \langle 0\, Y_{c-1}\, 0\, Y_{c-2} \cdots 0\, Y_a X_{a-1} Y_{a-1} \cdots X_1 Y_1 X_0 Y_0 \rangle,$$

and a decimal part $\mathcal{D}_{xy}$ consisting of $2b$ digits,

$$\mathcal{D}_{xy} = \langle X'_{b-1} Y'_{e-1} X'_{b-2} Y'_{e-2} \cdots X'_{b-e} Y'_0 X'_{b-e-1} 0\, X'_{b-e-2}\, 0 \cdots X'_0\, 0 \rangle.$$

It is well known that, given n points $p_i = (x_1^i, x_2^i, \ldots, x_d^i) \in \mathcal{V}$ in d-dimensional space, with constant $d \geq 1$, the numbers $\mathcal{N}_i = \mathcal{N}_{x^i \ldots x_d^i}$ $(1 \leq i \leq n)$ can be sorted in $O(n \log n)$ time.

A *Box Quadtree* T, in d-dimensional space $\mathbb{R}^d$ is defined by associating each node v in T with a hypercube in $\mathbb{R}^d$. The hypercube in the root of T contains the entire data set $\mathcal{V}$. We assume that the coordinates of each point in $\mathcal{V}$ are real numbers which lie in the interval $[0, 10^k - 1]$, for any $k > 0$, and have k digits in their integer part and a given constant length in their decimal part. Thus, the hypercube in the root of T has size $(10^k - 1)^d$. If the associated hypercube of a node v contains more than a single point, the hypercube is subdivided into at most 10^d equally sized hypercubes, associated with the children of v. The subset of $\mathcal{V}$, which is included in some hypercube of node v, is denoted by $\mathcal{V}[v]$. If $\mathcal{V}(v) = \emptyset$, the node v is called *empty*.

For each node v, the left-to-right order of the nonempty children $v_0, v_1, \ldots, v_k$ of v ($k \in \{0, 1, 2, \ldots, 10^d - 1\}$), corresponds to the increasing order of the set $\mathcal{N}$, i.e. let v_p and v_t be two nonempty children of v (with $p < t$); then, for each point $p_i = (x_1^i, x_2^i, \ldots, x_d^i) \in \mathcal{V}[v_p]$ and for each point $p_j = (x_1^j, x_2^j, \ldots, x_d^j) \in \mathcal{V}[v_t]$ we have: $\mathcal{N}_{x^i \ldots x_d^i} < \mathcal{N}_{x^j \ldots x_d^j}$. Each leaf of T stores a hypercube which includes a single point $p_i \in \mathcal{V}$. We conclude that,

Theorem 2.1. *The left-to-right order of the leaves of T corresponds to the increasing order of the set $\mathcal{N} = \{\mathcal{N}_i \mid 1 \leq i \leq n\}$.*

Unfortunately, a Box Quadtree T may have arbitrary depth, independently of the number of input points. If a node v obtains two or more nonempty children then, it is called an *interesting* node, otherwise, the tree T has a maximal path $v, w, \ldots, u$ such that the corresponding subsets are equal: $\mathcal{V}[v] = \mathcal{V}[w] = \cdots = \mathcal{V}[u]$; any node in this path has exactly one nonempty child except for the last node u which has two or more nonempty children. We compress the path $v, w, \ldots, u$ and the node u replaces the node v; any other node in this path and their empty children are deleted. The interesting node u is called a *compressed node*. By compressing any such path in T, we obtain a *compressed Box Quadtree* T with height $O(n)$ (Fig. 1). Notice that, for simplicity, the examples in Figs. 1–3 are given in $d = 2$ dimensions, and the coordinates of each point of $\mathcal{V}$ lie into the interval $[0, 2^k - 1]$ and are expressed in binary form; thus, each node of T has at most 2^2 nonempty children.

According to the standard practice in computational geometry, we assume that certain operations on points in $\mathbb{R}^d$ can be done in $O(1)$ time. Given a point x and a hypercube h_j in T, we can decide in $O(1)$ time if h_j covers x and in the positive case we can also decide which child of h_j covers x.

A point location search in a compressed Box Quadtree T is to locate the smallest hypercube in the nodes of T, covering the query point x. The search starts at the root of T and returns either a leaf in T or a compressed node with none of its children nodes covering the location of x. Hence, the search time in T is $O(n)$. An insertion or deletion can be done by a search in $O(n)$ time plus $O(1)$ updates in T. For an insertion of a point x, the location search returns a node n_i (which is either a leaf already containing a point or a compressed

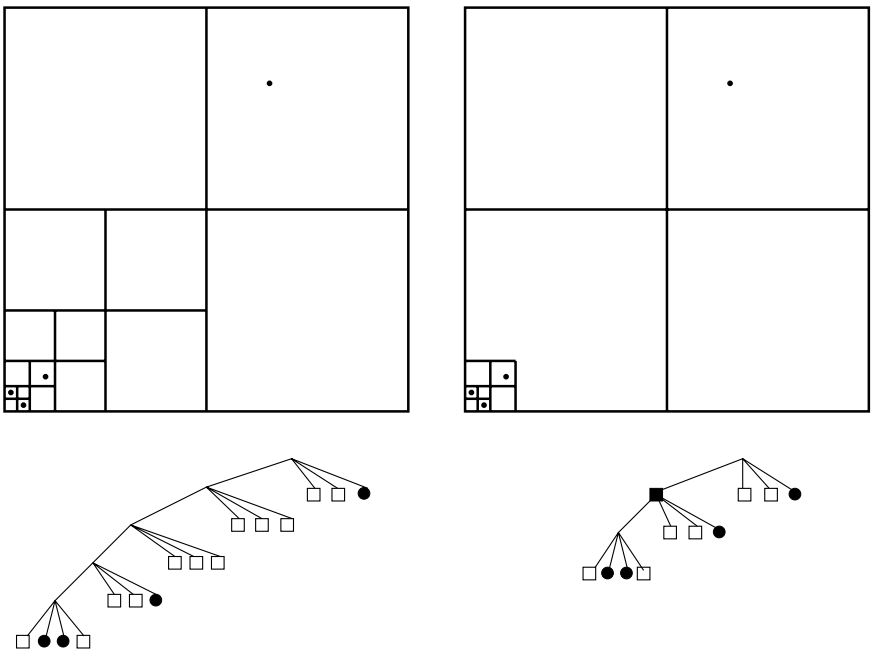


Fig. 1. An ordered Box Quadtree with four points (left) and its compressed Box Quadtree (right) with height 3. The node with a "*bold square*" is a compressed node, the node with an "*empty square*" represents an empty hypercube, and the node with a "*bold circle*" represents a hypercube which contains a single point.

node). Thus, we add in $O(1)$ time in T, an interesting node n_j that covers both x and $\mathcal{V}[n_i]$. The node n_j becomes a child of the parent of n_i and obtains as children the node of x and the node n_i.

Deleting a point x from T implies that the leaf n_i which covers x becomes empty, and is deleted. If the parent n_j of n_i remains with one nonempty child n_k then, we compress the path n_j, n_k. Thus, the compressed Box Quadtree is updated correctly in $O(1)$ time.

In order to improve the $O(n)$ search time in T, first we sort in $O(n \log n)$ time the set $\mathcal{V}$ on the basis of the increasing order of the set $\mathcal{N}$, and we implement $\mathcal{V}$ as a 2–3 tree $\mathcal{S}$ with height $h = O(\log n)$ (Fig. 2). Let N_i be the subset of $\mathcal{V}$ stored in the nodes of height i in the tree $\mathcal{S}$. For all $i = h, h-1, \ldots, 0$ we define the subset $\mathcal{V}_i \subseteq \mathcal{V}$:

(a) $\mathcal{V}_h = N_h$,
(b) $\mathcal{V}_{i-1} = \mathcal{V}_i \cup N_{i-1}$ $(i = h, h-1, \ldots, 1)$.

We obtain,

(1) $2^{(h-i)+1} - 1 \leq |\mathcal{V}_i| \leq 3^{(h-i)+1} - 1$ $(i = h, h-1, \ldots, 0)$
(2) $\mathcal{V} = \mathcal{V}_0$
(3) $\sum_{i=0}^{h} \mathcal{V}_i = O(n)$.

For each $\mathcal{V}_i$ $(i = h, h-1, \ldots, 0)$ we construct an ordered compressed Box Quadtree T_i and obtain the sequence of Box Quadtrees $T_h, T_{h-1}, \ldots, T_0$ that are defined respectively on the subsets $\mathcal{V}_h, \mathcal{V}_{h-1}, \ldots, \mathcal{V}_0$ of the data set $\mathcal{V}$ (Fig. 3).

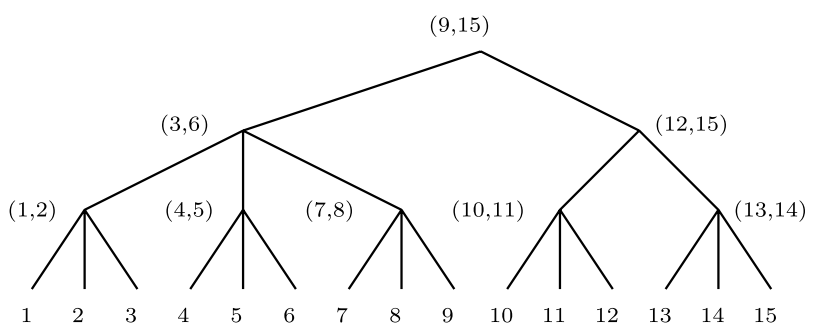


Fig. 2. The 2–3 tree used for the construction of the Box Quadtrees in Fig. 3.

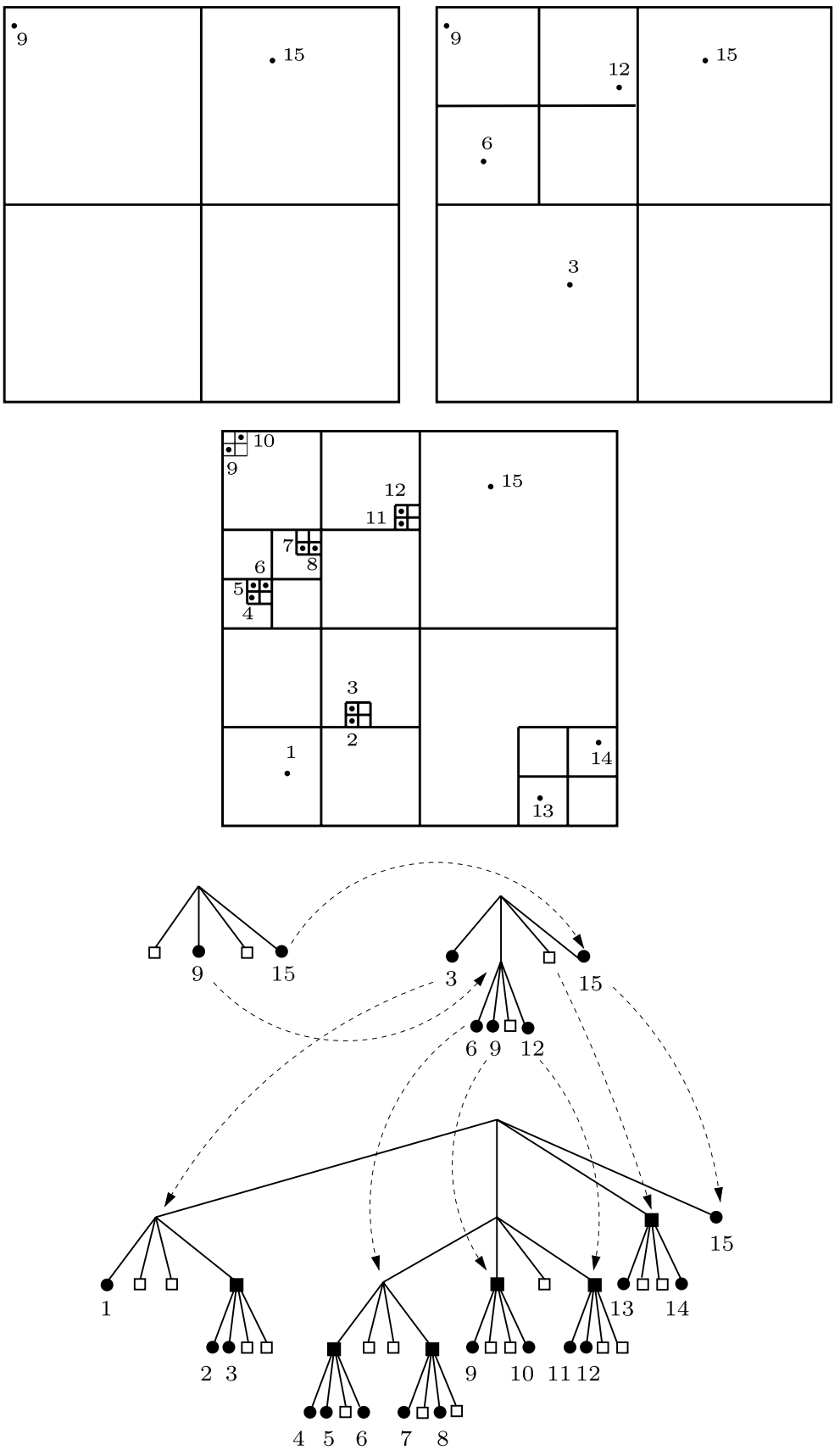


Fig. 3. A sequence of three compressed Box Quadtrees and the corresponding two-dimensional boxes, over the 2–3 tree in Fig. 2.

The above construction is based on a related work in [Eppstein *et al.*, 2005], which constructs a deterministic skip PR-quadtree guided by a deterministic 1–2–3 skip list [Munro *et al.*, 1992]. We conclude,

Theorem 2.2. *The sequence of compressed Box Quadtrees $T_h, T_{h-1}, \ldots, T_0$ is constructed in $O(n)$ space and $O(n \log n)$ time.*

To resolve a point location searching problem in a sequence $\mathcal{T} = \{T_h, T_{h-1}, \ldots, T_0\}$ we must find the smallest hypercube in the tree T_0 covering a query point x. We now show that a searching problem in $\mathcal{T}$ can be done in $O(\log n)$ time: Start at the root of $\mathcal{S}$ and search for the smallest hypercube in the tree T_h covering a query point x. When we terminate the search for a tree T_i we continue the search for tree T_{i-1} $(i = h, h-1, \ldots, 1)$. Let $v_{j(i)}$ be the node in T_i which stores the smallest hypercube covering the query point x, and let $v_{j(i-1)}$ be the copy of $v_{j(i)}$ in the tree T_{i-1}. We continue the search in the subtree of T_{i-1} with root $v_{j(i-1)}$, and let $v_{j(i-1)}, v_{a_1}, \ldots, v_{a_r}$ be the searching path. We will show that the length of the search path is $r = O(1)$. The interesting nodes v_{a_i} $(\forall\, i \in \{1, 2, \ldots, r\})$ of the tree T_{i-1} do not exist in the tree T_i. Moreover, the ordered set $\mathcal{V}_{i-1}$ is a merge of the ordered sets $\mathcal{V}_i$ and N_{i-1}. The 2–3 property of $\mathcal{S}$ implies that in the set $\mathcal{V}_{i-1}$, there are alternatively q $(1 \leq q \leq 2)$ points of N_{i-1} with one point of $\mathcal{V}_i$. For example from the 2–3 tree in Fig. 2 we have (the underlining elements in $\mathcal{V}_i$ denote the elements of N_i, $0 \leq i \leq 2$):

$$\mathcal{V}_2 = \{\underline{9}, \underline{15}\}$$

$$\mathcal{V}_1 = \{\underline{3}, \underline{6}, 9, \underline{12}, 15\}$$

$$\mathcal{V}_0 = \{\underline{1}, \underline{2}, 3, \underline{4}, \underline{5}, 6, \underline{7}, \underline{8}, 9, \underline{10}, \underline{11}, 12, \underline{13}, \underline{14}, 15\}.$$

Thus, any path $v_{j(i-1)}, v_{a_1}, \ldots, v_{a_r}$ may contain at most five points and we obtain $r \leq 4$. An insertion or deletion of a point x in the tree $\mathcal{S}$ can be accompanied by an update in the sequences $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_h$ and $T_0, T_1, \ldots, T_h$, in $O(\log n)$ time.

3. Determining the Fractal Dimension of Nonempty Set $\mathcal{V}$

It is well known, that the fractal dimension describes how many new pieces of a set are resolved as the resolution scale is decreased [Mandelbrot, 1983]. Furthermore, since a fractal set $\mathcal{V}$ is self-similar, this means that the fractal dimension can be evaluated by comparing properties between any two scales, namely [Liebovitch & Toth, 1989; Kruger, 1996],

$$\dim_{\mathrm{B}}(\mathcal{V}) \simeq -\frac{\Delta \log N(\varepsilon_l)}{\Delta \log \varepsilon_l}, \tag{2}$$

where $N(\varepsilon_l)$ is the number of boxes with minimum size ε_l, needed to cover the set $\mathcal{V}$. One common approach is to plot $\log N(\varepsilon_l)$ against $\log \varepsilon_l$ and use the slope of this curve to approximate $\dim_{\mathrm{B}}(\mathcal{V})$. To find the number $N(\varepsilon_l)$ we count the number n of nonempty leaves in the tree T_0.

Insertion or deletion of a point from the set $\mathcal{V}$, implies the insertion or deletion of a leaf in the tree T_0 and the number of points required in Eq. (2) becomes $n + 1$ or $n - 1$, respectively. Moreover, an insertion or deletion may decrease or increase respectively the minimum size ε_l, which is updated in constant time. Thus the fractal dimension $\dim_{\mathrm{B}}(\mathcal{V})$ is updated in $O(\log n)$ total time.

The box-counting method in one-dimension $(d = 1)$ involves finding the number of intervals with minimum size on the x-axis, needed to cover the set $\mathcal{V}$. This means that, for any point $v \in \mathcal{V}$ we know the predecessor and the successor points in $\mathcal{V}$, on the x-axis. Therefore, the box-counting method resolves the sorting problem of the n points on the x-axis. Since $O(n \log n)$ is the lower bound for the time complexity of the sorting problem of n real numbers, we conclude,

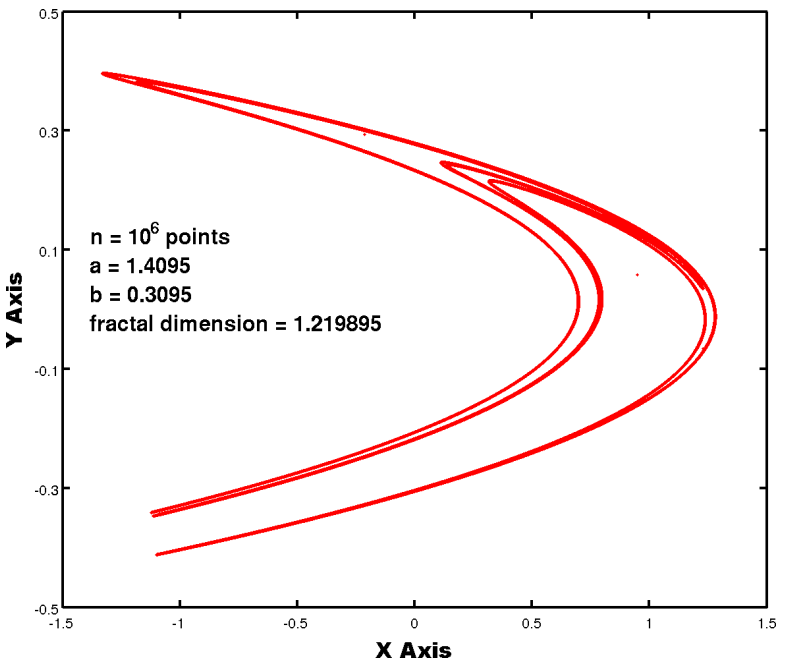
Theorem 3.1. *The dynamic box-counting method can be done in $O(n \log n)$ time and $O(n)$ space, and this is optimal. An insertion or deletion in the data point set is updated in optimal $O(\log n)$ time.*

4. Experimental Results

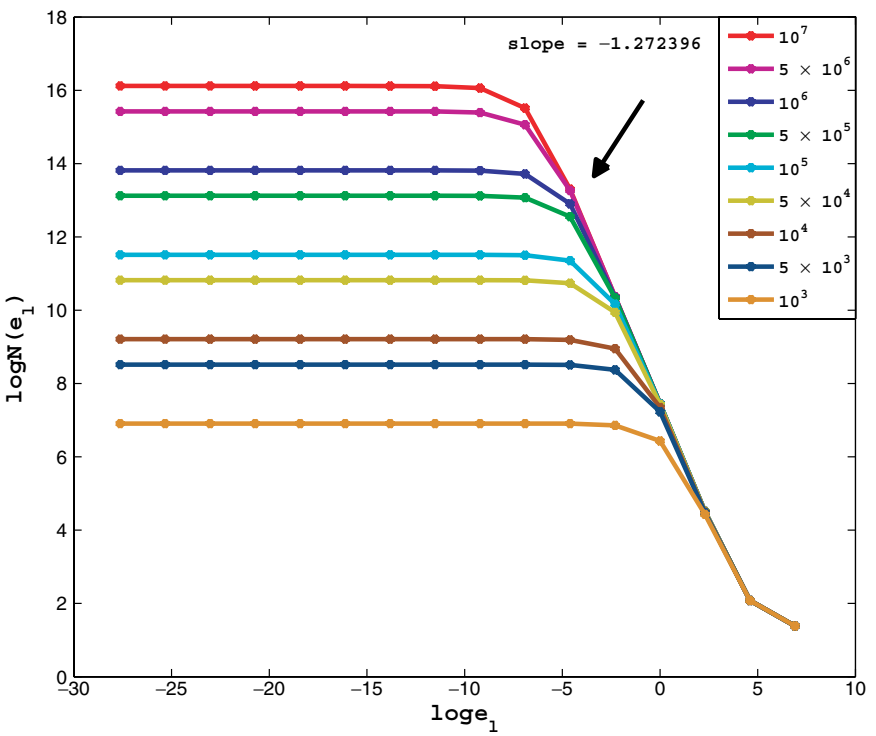
Next, we present experimental results of our approach using one of the most studied examples of dynamical systems that exhibit chaotic behavior which is the *Hénon Map*. Hénon map is a discrete-time dynamical system and it is defined as follows [Hénon, 1976]:

$$\begin{cases} x_{i+1} = y_i + 1 - a x_i^2 \\ y_{i+1} = b x_i \end{cases} \quad \text{for } i = 0, 1, 2, \ldots. \tag{3}$$

This map takes a point (x_i, y_i) in the plane and maps it to a new point (x_{i+1}, y_{i+1}). It depends on two parameters, a and b, which for the *canonical Hénon Map* have the values $a = 1.4$ and $b = 0.3$. For these *canonical values* the Hénon map is chaotic. Depending on the initial point (x_0, y_0), the

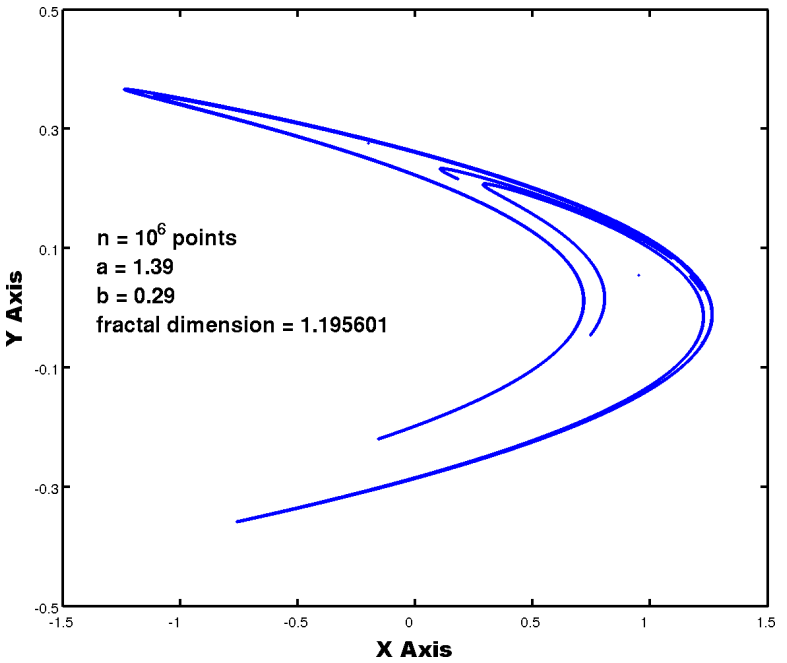


(a)

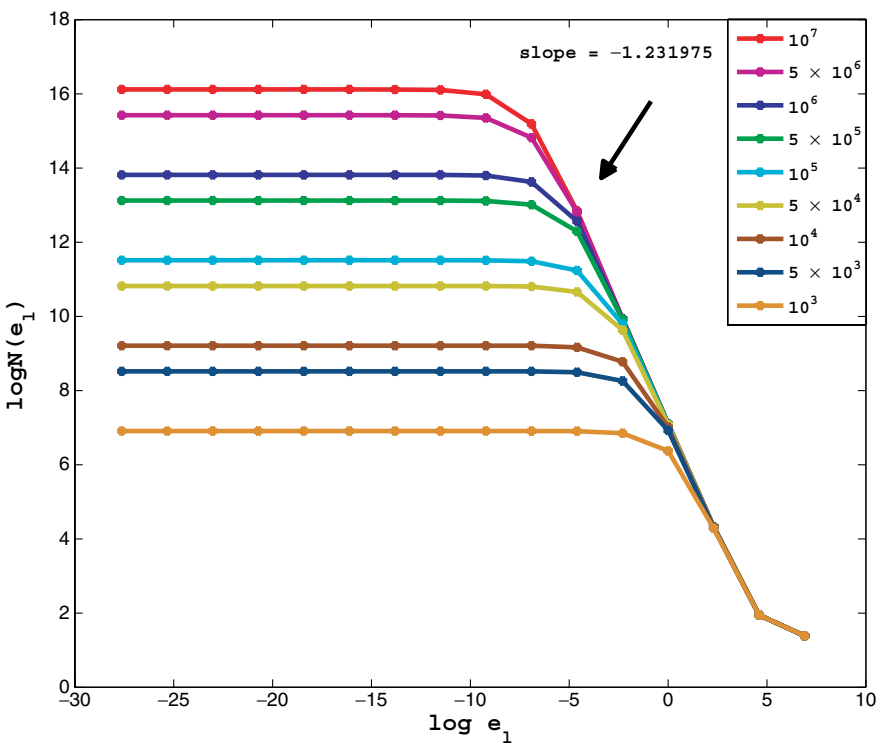


(b)

Fig. 4. (a) The data set produced by the Hénon map with parameters $a = 1.4095$ and $b = 0.3095$. (b) Box-counting results for nine data sets of $n = 10^3$, $n = 5 \times 10^3$, $n = 10^4$, $n = 5 \times 10^4$, $n = 10^5$, $n = 5 \times 10^5$, $n = 10^6$, $n = 5 \times 10^6$ and $n = 10^7$ points. The estimated slope of each graph gives the corresponding fractal dimension exhibited in Table 1.

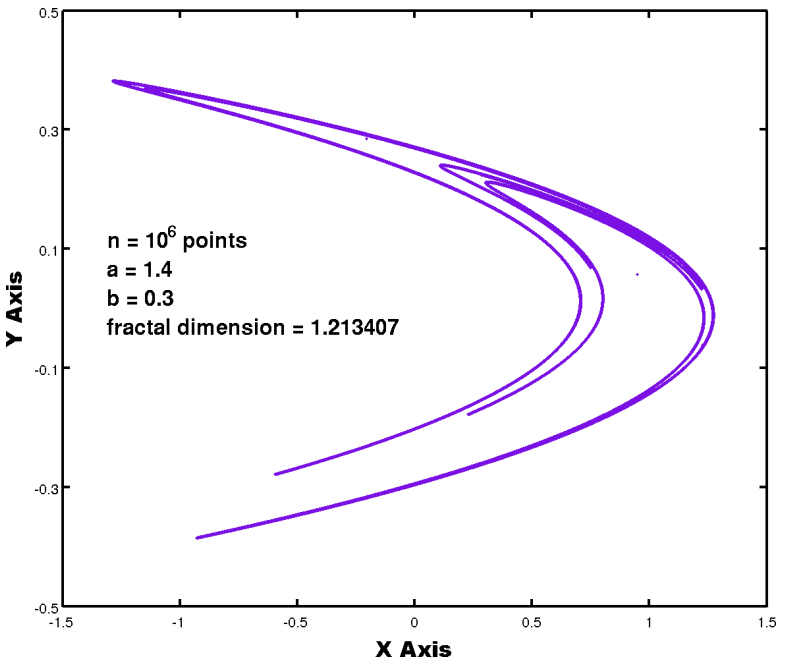


(a)

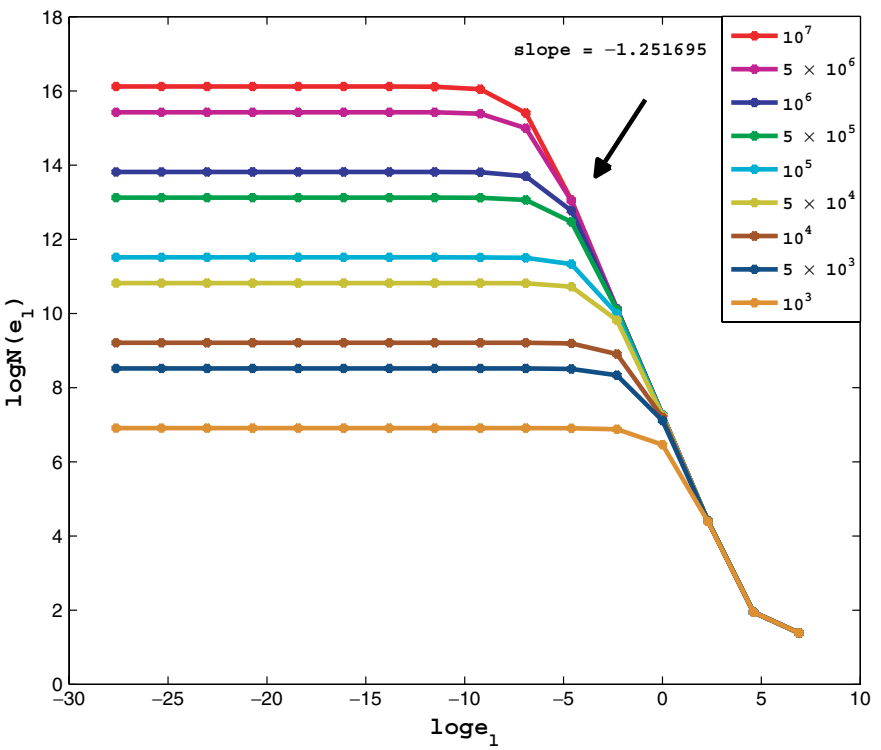


(b)

Fig. 5. (a) The data set produced by the Hénon map with parameters $a = 1.39$, $b = 0.29$. (b) Box-counting results for nine data sets of $n = 10^3$, $n = 5 \times 10^3$, $n = 10^4$, $n = 5 \times 10^4$, $n = 10^5$, $n = 5 \times 10^5$, $n = 10^6$, $n = 5 \times 10^6$ and $n = 10^7$ points. The estimated slope of each graph gives the corresponding fractal dimension exhibited in Table 1.



(a)



(b)

Fig. 6. (a) The data set produced by the Hénon map with canonical parameters $a = 1.4$, $b = 0.3$. (b) Box-counting results for nine data sets of $n = 10^3$, $n = 5 \times 10^3$, $n = 10^4$, $n = 5 \times 10^4$, $n = 10^5$, $n = 5 \times 10^5$, $n = 10^6$, $n = 5 \times 10^6$ and $n = 10^7$ points. The estimated slope of each graph gives the corresponding fractal dimension exhibited in Table 1.

Table 1. Fractal dimension ($\dim_B(\mathcal{V})$) for the nine data sets of Figs. 4–6 for $n = 10^3$, $n = 5 \times 10^3$, $n = 10^4$, $n = 5 \times 10^4$, $n = 10^5$, $n = 5 \times 10^5$, $n = 10^6$, $n = 5 \times 10^6$ and $n = 10^7$ points.

	$\dim_B(\mathcal{V})$		
n	$a = 1.4095, b = 0.3095$	$a = 1.39, b = 0.29$	$a = 1.4, b = 0.3$
10^3	0.945001	0.961029	0.980880
5×10^3	1.118395	1.081111	1.121696
10^4	1.143340	1.100308	1.140679
5×10^4	1.150556	1.120797	1.148970
10^5	1.180866	1.142795	1.172097
5×10^5	1.203798	1.159760	1.174206
10^6	1.219895	1.195601	1.213407
5×10^6	1.267300	1.228493	1.248346
10^7	1.272396	1.231975	1.251695

sequence of points obtained by iteration of the mapping either diverges to infinity or tends to a strange attractor, known as *Hénon attractor*, which appears to be the product of an one-dimensional manifold by a Cantor set [Hénon, 1976]. The Hénon attractor is a fractal, smooth in one direction and a Cantor set in another.

We have applied the proposed dynamic box-counting algorithm to compute the fractal dimension of various data sets $\mathcal{V}$ that consist of n points (x_j, y_j), $j = 0, 1, \ldots, n$ produced by the Hénon map. First, by using the Hénon Map (3) with values $a = 1.4095$ and $b = 0.3095$ and starting point $(x_0, y_0) = (0.63135448, 0.18940634)$ we produced nine data sets for $n = 10^3$, $n = 5 \times 10^3$, $n = 10^4$, $n = 5 \times 10^4$, $n = 10^5$, $n = 5 \times 10^5$, $n = 10^6$, $n = 5 \times 10^6$ and $n = 10^7$ points. The corresponding set for $n = 10^6$ is exhibited in Fig. 4(a). In Fig. 4(b), we plot $\log N(\varepsilon_l)$ against $\log \varepsilon_l$ where $N(\varepsilon_l)$ is the number of boxes with minimum size ε_l, needed to cover the corresponding set $\mathcal{V}$ for all the considered six data sets. To find the number $N(\varepsilon_l)$ we have counted the number of nonempty leaves in the corresponding tree T_0. Then by using the slope of the curve that correspond to $n = 10^7$, we obtained $\dim_B(\mathcal{V}) = 1.272396$.

In Fig. 5 using the parameters $a = 1.39$, $b = 0.29$ we exhibit the corresponding results with the same amount of points. The data set produced with 10^7 points gives $\dim_B(\mathcal{V}) = 1.231975$. In Fig. 6, we give the corresponding results for the canonical parameters $a = 1.4$, $b = 0.3$. In this case, the data set produced with 10^7 points gives $\dim_B(\mathcal{V}) = 1.251695$.

In Table 1 we give the fractal dimension, $\dim_B(\mathcal{V})$, for the nine data sets of Figs. 4–6 for $n = 10^3$, $n = 5 \times 10^3$, $n = 10^4$, $n = 5 \times 10^4$, $n = 10^5$, $n = 5 \times 10^5$, $n = 10^6$, $n = 5 \times 10^6$ and $n = 10^7$ points.

For comparative purposes with the static algorithm of [Hou *et al.*, 1990] which is implemented in [Kruger, 1996], we have calculated the CPU time required by this algorithm as well as by the proposed dynamic box-counting algorithm. To this end, by using Hénon Map and parameters $a = 1.4$, $b = 0.3$ we constructed a data set produced with 10^4 points and each time we insert 10^4 points in the data set. In Fig. 7, we exhibit the CPU time required by the static algorithm when it is re-executed over the updated data sets as well as the corresponding CPU

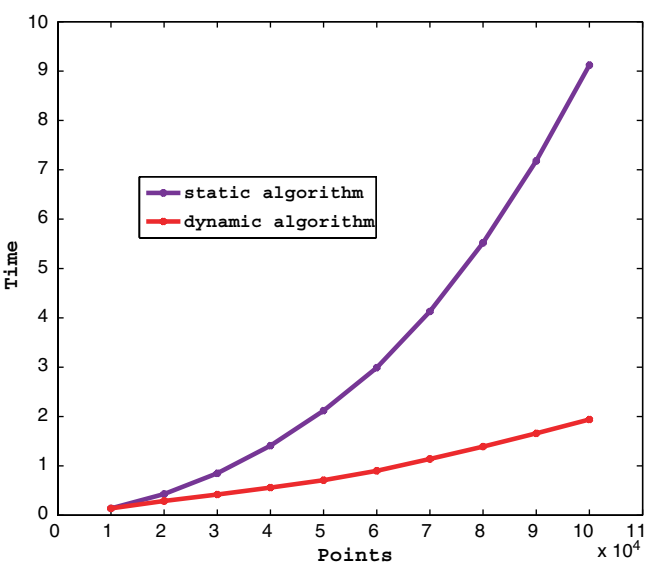


Fig. 7. The CPU time of the static and dynamic box-counting algorithm with parameters $a = 1.4$ and $b = 0.3$; each time we insert 10^4 points in the data set.

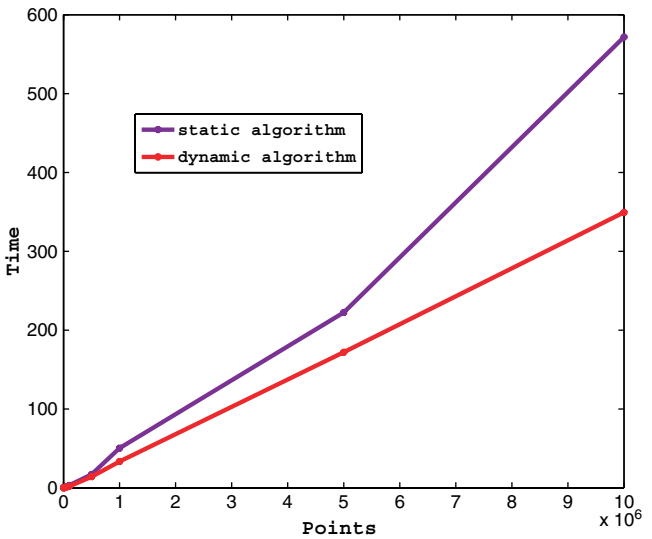


Fig. 8. The CPU time of the static and dynamic box-counting algorithms for the data sets of Table 1 with parameters $a = 1.4$ and $b = 0.3$.

time consumed by our dynamic box-counting algorithm. An additional comparison is given in Fig. 8 where we exhibit the CPU time required by the static algorithm when it is re-executed over the updated data sets considered in Table 1 as well as the corresponding CPU time consumed by our dynamic box-counting algorithm. It is evident that the proposed algorithm achieves smaller running time in all the considered cases.

5. Synopsis

In summary, we have proposed the optimal dynamic box-counting algorithm for estimating the dynamic fractal dimension of a set in a nonstationary environment which changes over time and it is characterized by insertion of new points into the set as well as deletion of existing points from the set. We have succeeded to update the box-counting result at appropriate time intervals with the lowermost possible computational cost. The proposed algorithm tackles the dynamic box-counting problem by using a sequence of $O(\log n)$ compressed Box Quadtrees, guided by a 2–3 tree, to store the data points. The time complexity of the proposed algorithm is $O(n \log n)$, while the space complexity is $O(n)$, where d denotes the dimension of the problem at hand and n is the number of considered points. In addition, the time complexity of an insertion or a deletion is $O(\log n)$. We have shown that the above time and space complexity are optimal. Experimental results suggest that the proposed dynamic box-counting algorithm achieves smaller running time compared with the corresponding static algorithm.

Acknowledgment

We gratefully acknowledge many stimulating and useful discussions with Professor T. C. Bountis.

References

Aho, A. V., Hopcroft, J. E. & Ullman, J. D. [1974] *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA).

Eppstein, D., Goodrich, M. T. & Sun, J. Z. [2005] "The skip quadtree: A simple dynamic data structure for multidimensional data," *Proc. 21st ACM Symp. Computational Geometry*, pp. 296–305.

Hénon, M. [1976] "A two-dimensional mapping with a strange attractor," *Commun. Math. Phys.* **50**, 69–77.

Hou, X.-J., Gilmore, R., Mindlin, G. B. & Solari, H. G. [1990] "An efficient algorithm for fast $O(N \log N)$ box counting," *Phys. Lett. A* **151**, 43–46.

Kaplan, D. (unpublished).

Kruger, A. [1996] "Implementation of a fast box-counting algorithm," *Comput. Phys. Commun.* **98**, 224–234.

Liebovitch, L. S. & Toth, T. [1989] "A fast algorithm to determine fractal dimensions by box counting," *Phys. Lett. A* **141**, 386–390.

Mandelbrot, B. B. [1983] *The Fractal Geometry of Nature* (Freeman, San Francisco).

Munro, J. I., Papadakis, T. & Sedgewick, R. [1992] "Deterministic skip lists," *Proc. Third Annual ACM-SIAM Symp. Discrete Algorithms* (*SODA*), pp. 367–375.