Mining for High Complexity Regions Using Entropy and Box Counting Dimension Quad-Trees

Rosanne Vetro, Wei Ding, Dan A. Simovici
Univ. of Massachusetts Boston, Dept. of Comp. Science, 100 Morrissey Blvd., Boston, Massachusetts 02125 USA
{rvetro,ding,dsim} at cs.umb.edu

***Abstract*—This paper introduces an algorithm for capturing high complexity regions of a data domain. In this work, we focus on domains in R^2. In particular, we analyze 2-dimensional image domains. Two different methods for mining are considered. The first method performs an information-theoretic analysis based on entropy to find diverse areas. The second method applies the concept of box-counting dimension related to fractal geometry. We propose the use of a quad-tree as main search structure where complex areas are represented by leaves with high feature[1] values at the highest level on the tree. Nodes that refer to specific sub-domains are split when the level of the analyzed feature exceeds a chosen threshold. The relationship between the threshold and the number of pixels located in high value feature sub-domains at the highest level on the resultant quad-tree is demonstrated on test images for both methods. Experimental results also show the relation between the former measurements and characteristics of the images. Finally, we identify a correlation between the methods presented.**

***Keywords*-Entropy, Box Counting Dimension, Quad-Trees**

I. Introduction

The concept of complexity relates to the presence of variation. In science there are many approaches that characterize complexity. A variety of scientific fields have dealt with complex mechanisms, simulations, systems, behavior and data complexity as those have always been a part of our environment. In this work, we focus on the topic of data complexity which is studied in information theory.

While randomness is not considered complexity in certain areas such as those related to the study of complex systems, information theory tends to assign high values of complexity to random noise. Many fields benefit from the identification of content or noise related complex areas. In data hiding, adaptive steganography takes advantage of high concentration of self-information on high complexity areas originated from both content and noise to embed data. The authors of [1] describe the benefits of selective embedding related to the reduction of perceptual degradation for transform domain steganographic techniques. Bio diversity is another area where complexity can be used for identification and localization of different species. In this case, the complexity originated from content is more important than the one originated from noise.

[1]Entropy or box-counting dimension

Our goal in this paper is to provide an algorithm that captures high complexity sub domains of a data domain and introduce two distinct methods to achieve that goal. The first method has its base in information theory where information entropy is also used as indicative of complexity. The second has its roots in fractal geometry where the so called box-counting dimension (BCD) is used to determine the fractal dimension of a set S in a Euclidean space R^n. We focus our work on 2-dimensional image domains and observe that high complexity areas originated from both data content and noise are mined by the methods proposed.

In the next section we introduce the proposed algorithm and explain the searching process. In section III we describe the information-theoretic method used for mining complex sub-domains. The second method uses the concept of box-counting dimension and is introduced in section IV. We provide a brief description about implementation details in section V. In section VI we discuss the experiments and compare the results generated by both methods. The paper is concluded in section VII.

II. Algorithm Description

The algorithm proposed constructs a full quad-tree related to the image entropy or box-counting dimension concentration to find high complexity areas.

Definition A *quad-tree Y* is a structure defined on a finite set of nodes that either contains no nodes or is comprised of a root node and 4 quad-subtrees. In a full quad-tree, each node is either a leaf or has degree exactly 4.

The construction of the quad-tree is based on the measurements of the feature in image sub-areas, which can also be regarded as tree nodes. The algorithm receives as input the gray scale version of an image, a minimum area size for analysis and arguments relevant to the node splitting condition. For the entropy based method described in section III, we use a predetermined threshold for the entropy in order to decide wether or not to split a node. For the box-counting dimension method, two distinct arguments are used in the splitting condition: a predefined threshold for the fraction of intercepting boxes or rectangles at any image sub-area and a predefined threshold for the number of gray shades to be considered at the intercepting analysis. The

entire image area corresponds to the root of the quad-tree. The expansion of each node is based on its feature value and the predetermined threshold(s) used for the splitting condition, as well as the size of the corresponding sub-area. Only nodes with area greater or equal to the defined minimum area size are expanded.

The algorithm, introduced in Algorithm 1, outputs a quad-tree showing the feature concentration along the whole image area. In this representation, leaves are assigned with a shade of gray, depending on their location on the tree level. Leaves located closer to the root correspond to areas of the image assigned with darker shades of gray whereas leaves located further from the root correspond to areas of the image assigned with lighter shades of gray. The algorithm also highlights the leaves at the highest tree level with highest feature value. In most cases, those leaves correspond to high complexity regions of the image.

Algorithm 1 ComputeHCRegions(*image,minArea,thr1,thr2*)

Input: Gray scale image, minimum area size for the analysis, feature threshold, threshold corresponding to the number of shades of gray (used only by the BCD method)
Output: Quad-tree showing the feature concentration along the whole image area

$nId \longleftarrow ROOT$
$nLevel \longleftarrow 0$
$root \longleftarrow newNode(nId, nLevel, image.width, image.height)$
$ComputeFeature(root)$
$Split(root)$
$HighlightHighFeatureLeaves()$

The function *ComputeFeature* evaluates the feature associated with the histogram of the pixels in the node's area. We present two version for this function in section III and section IV as it differs according to the method used. The recursive method *Split* introduced in Algorithm 2 expands a node if its feature satisfies the method related splitting condition and if its area is greater or equal to the defined minimum area size. A gray shade corresponding to a level in the final tree, is assigned to every leaf node by the method *Draw*. Information about each leaf such as its id, feature value and level is saved in a text file by the method *SaveNodeInfo*. The method *Release* frees the memory space previously allocated to a node. Finally, the method *HighlightHighFeatureLeaves* highlights in pink or white the leaves at the highest tree level with highest feature values, corresponding in most cases to high complexity regions. The white color leaves are the ones with the highest feature value among all pink leaves.

III. Information-theoretical Method

Information theory involves the quantification of information and was created with the purpose of finding fundamental limits on compressing, reliably storing and communicating data. Entropy is a important measure of information in the theory that quantifies the uncertainty associated with the value of a discrete random variable $\mathcal{X}$. Equal values taken by $\mathcal{X}$ can be separated into disjoint sets or block to form a partition. Simovici and Djeraba [2] present a generalized notion of entropy of a partition, with Shannon entropy as a special case:

Definition Let S be a finite set containing the possible values for the random variable $\mathcal{X}$ and let $\pi = B_1, ..., B_n$ be a partition of S. The *Shannon Entropy of* π is the number:

$$\mathcal{H} = -\sum_{i=1}^{n} \frac{|B_i|}{|S|} \log_2 \frac{|B_i|}{|S|}$$

The *Shannon Entropy* can be used to evaluate the uniformity of the elements of S in the blocks π since the entropy

Algorithm 2 Split(*n*)

Input: A node *n* from a quad-tree
Output: Expands the node creating four children, if node satisfies the necessary requirements

if $(n.feature > method\_lower\_bound)$ and $(n.area > minArea)$ **then**
$nLevel \longleftarrow n.level + 1$
$nId \longleftarrow n.id + A$
$topLeft \longleftarrow newNode(nId, nLevel, n.rect.x, n.rect.y, n.rect.width/2, n.rect.height/2)$
$ComputeFeature(topLeft)$
$nId \longleftarrow n.id + B$
$topRight \longleftarrow newNode(nId, nLevel, n.rect.x + n.rect.width/2, n.rect.y, n.rect.width/2, n.rect.height/2)$
$ComputeFeature(topRight)$
$nId \longleftarrow n.id + C$
$bottonLeft \longleftarrow newNode(nId, nLevel, n.rect.x, n.rect.y + n.rect.height/2, n.rect.width/2, n.rect.height/2)$
$ComputeFeature(bottonLeft)$
$nId \longleftarrow n.id + D$
$bottonRight \longleftarrow newNode(nId, nLevel, n.rect.x + n.rect.width/2, n.rect.y + n.rect.height/2, n.rect.width/2, n.rect.height/2)$
$ComputeFeature(bottonRight)$
$Release(n)$
$Split(topLeft)$
$Split(topRight)$
$Split(bottonLeft)$
$Split(bottonRight)$
else
$SaveNodeInfo(n)$
$Draw(n)$
$Release(n)$
end if

value increases with the uniformity of the distribution of the elements of S. Note that as the uniformity increases, so does the uncertainty associated.

Our method evaluates the Shannon Entropy of the local histograms of image sub-areas to find high complexity regions. The partition blocks of a node, used for the entropy analysis, consist of pixels with the same shade of gray.

Algorithm 3 ComputeFeature(*n*)

Input: A node *n* from a quad-tree
Output: The node entropy related to the histogram of the pixels in the area.

```
entropy ⟵ 0
for all pixel ∈ n.area do
    InsertGrayShade(histogram,pixel.shade)
end for
for all shade ∈ histogram do
    p ⟵ number_of_pixels_with_shade
    s ⟵ total_number_of_pixels_in_the_node
    g ⟵ (p ÷ s)
    entropy− = (g) × (lg2(g))
end for
return entropy
```

The first version of *ComputeFeature*, presented in Algorithm 3, corresponds to the information-theoretic method proposed. It computes the Shannon entropy associated with the histogram of the pixels in a node's area. This histogram is created by the method *InsertGrayShade*. The result generated by *ComputeFeature* is successively used by the recursive method *Split* shown in Algorithm 2. Only the nodes corresponding to sub-areas of the image where the Shannon entropy is above the predefined entropy threshold and have area greater or equal to the pre-defined minimum area size are expanded. We observed that leaves at the highest level in the resultant quad-tree may naturally have different associated Shannon entropy values. As we show in section VI, the leaves with highest entropy among the ones at highest level can better represent high complexity areas of the image.

IV. Box-counting dimension Method

The box-counting dimension is a measure used to determine the fractal dimension of a set $\mathcal{S}$ in a metric space. It reflects the variation of the results of measuring a set at a diminishing scale, which allows the observation of progressively smaller details.

Let ($\mathcal{S}$,$\mathcal{O}_d$) be a topological metric space and let *T* be a precompact set. For every positive *r*, there exists a *r*-net for *T*; that is a finite subset N_r of *S* such that $T \subseteq \bigcup\{C(x, r|x \in N_r\}$ for every $r > 0$. Denote by $n_T(r)$ the smallest size of an *r*-net of *T*. It is clear that $r < r'$ implies $n_T(r) \geq n_T(r')$. The box-counting dimension is introduced next (see [2]).

Definition Let ($\mathcal{S}$,$\mathcal{O}_d$) be a topological metric space and let *T* be a precompact set. The *upper box-counting dimension* of *T* is the number

$$ubd(T) = \limsup_{r \to 0} \frac{n_T(r)}{\log \frac{1}{r}}.$$

The *lower box-counting dimension* of *T* is the number

$$lbd(T) = \liminf_{r \to 0} \frac{n_T(r)}{\log \frac{1}{r}}.$$

If $ubd(T) = lbd(T)$, we refer to their common values as the box-counting dimension of *T*, denoted by $bd(T)$.

We use the box-counting dimension of the local histograms of image sub-areas to find high complexity regions. The box-counting dimension of a sub-area is based on to the number of intercepting boxes in the sub-area.

Definition A box is a sub-area of the image with size equal to the predefined minimum area size. An intercepting box corresponds to a box where the number of different shades of gray is greater or equal to the a predefined threshold.

Algorithm 4 ComputeFeature(*n*)

Input: A node *n* from a quad-tree
Output: Box-counting dimension associated to the node's area.

```
boxesIntercepting ⟵ 0
bcd ⟵ 0
for all box ∈ n.area do
    for all pixel ∈ box do
        InsertGrayShade(box.histogram,pixel.shade)
    end for
    if n.area = box.area then
        boxesIntercepting ⟵ box.histogram.size
    else if box.histogram.size ≥ threshold then
        boxesIntercepting ⟵ boxesIntercepting + 1
    end if
    Release(box.histogram)
end for
if boxesIntercepting > 0 then
    bcd− = boxesIntercepting ÷ lg10(1 ÷ (n.area))
end if
return bcd
```

The version of the function *ComputeFeature* presented in Algorithm 4 corresponds to the box-counting dimension method. It computes the box-counting dimension associated with the histogram of the boxes in a node's area. As in the Information-theoretic version, the method *InsertGrayShade* constructs a histogram of each box in the node or image sub-area. If the area corresponding to the node is equal to a box area, the number of intercepting boxes is the same as the number of different shades of gray in its histogram.

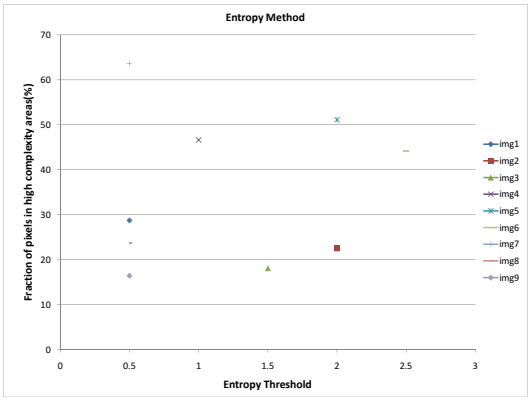
Otherwise, the number of intercepting boxes is equal to the number of boxes with histogram containing a number of shades of gray greater or equal to a predefined threshold. When the box-counting dimension method is used, the recursive method *Split* shown in Algorithm 2 expands a node according to a threshold related to the fraction of intercepting boxes found. For instance, a $fraction\ thereshold = 0.1$ represents a node having 10% of intercepting boxes among all its boxes. So in this case, the algorithm expands a node if its box-counting dimension corresponds to a fraction greater than 10% of intercepting boxes. The area corresponding to the node should also be greater or equal to the predefined minimum area size in order to promote expansion. As in the Information-theoretic method, we also observed that leaves at the highest level in the resultant quad-tree may naturally have different BCD values associated. We show in section VI that the leaves with highest BCD value among the ones at highest level can better represent high complexity areas of the image.

V. System Description

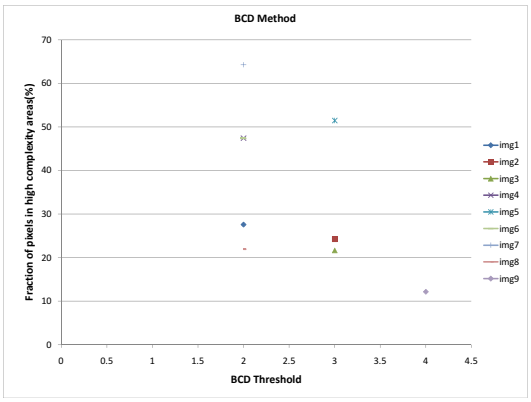
The algorithm was implemented in Java (JDK 6 Update 7) and the program is composed by 8 classes: $Main$, $Image$, $Tree$, $EntropyTree$, $BoxCountingTree$, $Node$, $EntropyNode$ and $BoxCountingNode$. The class $Tree$ is a super class for the classes $EntropyTree$ and $BoxCountingTree$ and the class $Node$ is a super class for the classes $EntropyNode$ and $BoxCountingNode$.The class $Main$ instantiates an $Image$ object.The class $Image$ implements the methods for encoding and decoding images, as well as for treating the image prior to the generation of the quad-tree. Image treatment may involve resizing and conversion to gray scale. The class $Tree$ is a super class with attributes and methods shared by both classes $EntropyTree$ and $BoxCountingTree$. Those two classes, together with the classes $EntropyNode$ and $BoxCountingNode$ contain the implementation of the methods presented in III and in IV. The class $Node$ is a super class with attributes and methods shared by both classes $EntropyNode$ and $BoxCountingNode$.

VI. Experimental Results

Experiments were performed over decompressed gray scale version of Jpeg images. The use of gray scale images allowed the methods to be applied over a reduced color space. The resultant image files were again compressed and presented as Jpeg files. The values chosen for all the thresholds promote a good capture of the complexity. The resultant images and statistics show that the quad-trees generated by both methods are quite similar. Fig. 1(a) and Fig. 1(b) present the relation between the values chosen as threshold for both methods and the percentage of the number of pixels located in high complexity areas relative to the total number of pixels in each sample image. One



(a)



(b)

Figure 1. Fraction of pixels in high complexity areas of the image given the corresponding (a) lower bound used for the entropy, (b) BCD threshold.

can notice that the percentages of pixels in high complexity areas generated for each image file are very close in value for both methods. The files corresponding to the quad-trees generated for the sample images are presented in Fig. 2. As mentioned in section II, a gray shade corresponding to a level in the final tree, is assigned to every leaf node. The leaves at the highest tree level with highest feature values, corresponding in most cases to high complexity regions, are highlights in pink or white. The white color leaves are the ones with the highest feature value among all pink leaves. Results for both methods also show the relation between the characteristics of the images and the values used for the node splitting condition. Images corresponding to natural scenes or objects and faces with a textured background require a higher value for the entropy threshold, as well as for the threshold used for the box-counting dimension evaluation in order to capture well the complex regions. Those images present a higher number of pixels located in high complexity areas. Images with objects and faces

exposed over a more uniform background require lower values for those parameters. Those images present a lower number of pixels located in high complexity areas.

Although only Jpeg files were used in the experiments here presented, the algorithm and methods described in this paper are independent of image type. So in order to compare the results between different formats, we also performed experiments with Bmp image files. In this case, each Jpeg file was created from an original Bmp image. Results for both formats regarding both methods were also quite similar and demonstrate that our algorithm can capture high complexity domains independent of a image format. We also observed that as we lowered the compression quality of Jpeg images, there was a decrease on the number of pixels located in high complexity sub-domains. Jpeg compression removes high frequency details from images as considered by Pevny and Fridrich [3]. Furthermore, the number of image artifacts increases as we lower the compression quality. Uncompressed formats(Bmp,Pcx) or lossless compression formats (Pgm,Tiff) usually carry a higher degree of noise and less artifacts. As a consequence of the high frequency removal and addition of more artifacts, Jpeg files with low quality usually have less high complexity areas when compared to the correspondent Jpeg image files compressed with higher quality and Bmp images. The results regarding the comparison between image files in Bmp and Jpeg formats are available at *http://www.cs.umb.edu/∼rvetro/index.htm*.

VII. Conclusion

Both methods used by the proposed algorithm (information-theoretic and box-counting dimension) successfully capture high complexity sub domains of a domain. The analysis of 2-dimensional image domains generated similar results for both methods, and images with different formats(medium/high quality Jpeg and Bmp). We also observed that besides capturing image regions corresponding to content related complex areas, both methods also capture other regions with high variance of shades among pixels caused by external factors such as light reflection originated from a camera flash. Nevertheless, the identification of any kind of high complexity region plays an important role for a variety of applications such as data hiding and bio-diversity systems.

References

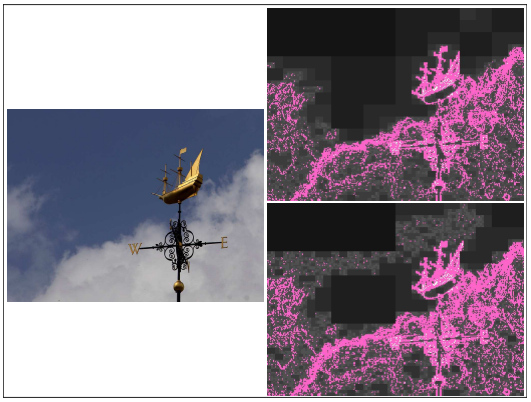
[1] Solanki, K., Dabeer, O., Madhow, U., Manjunath, B.S., Chandrasekaran, S.: Robust image-adaptive data hiding: Modeling, source coding, and channel coding. In: 41st Allerton Conference on Communications, Control, and Computing. (2003)

[2] Simovici, D.A., Djeraba, C.: Mathematical Tools for Data Mining – Set Theory, Partial Orders, Combinatorics. Springer-Verlag, London (2008)

[3] Pevny, T., Fridrich, J.: Benchmarking for steganography. In: Information Hiding: 10th International Workshop, IH 2008, Sana Barbara, CA, USA. Volume 5284., (Springer)

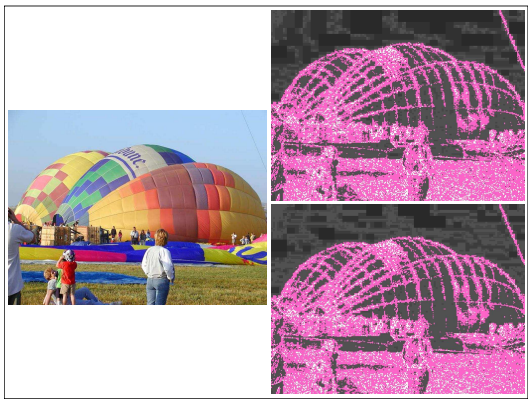
(a) img1

(b) img2



(c) img3

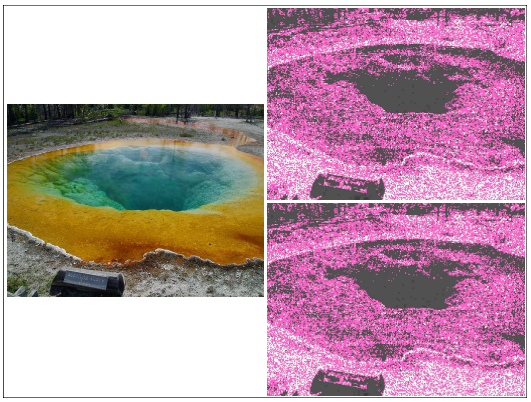
Figure 2. Sample simulation results for several original images comparing the corresponding Entropy Quad-Tree (top right in each subfigure) and the corresponding BCD Quad-Tree (bottom right in each subfigure).



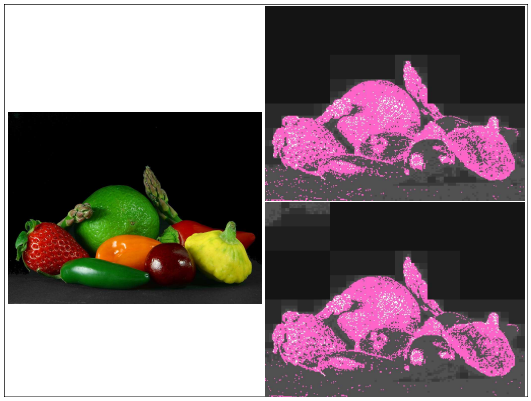
(d) img4

(e) img5

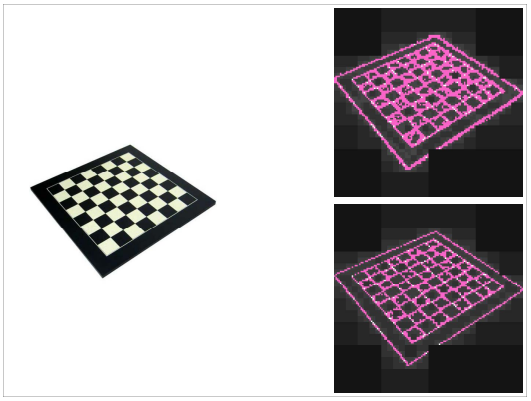
(f) img6



(g) img7



(h) img8



(i) img9

Figure 2. (cont.) Sample simulation results for several original images comparing the corresponding Entropy Tree (top right in each subfigure) and the corresponding BCD Tree (bottom right in each subfigure).