Abstract—We present a method for estimating the complexity of an image based on the concept of logical depth. Unlike the application of the concept of algorithmic complexity by itself, the addition of the concept of logical depth results in a characterization of objects by organizational (physical) complexity. We use this measure to classify images by their information content. The method provides a means for evaluating and classifying objects by way of their visual representations.

Index Terms—image classification, information content, algorithmic complexity, logical depth.

I. INTRODUCTION

 IMAGES have a number of properties containing information in the form of pixels. As a representation of an object, an image constitutes a partial description of the object, capturing some of its features.

The present work develops an automatic method based on the theory of algorithmic information, particularly on the concept of Bennett’s logical depth[1], to assess and quantify the information content of an image, providing a means for evaluating and classifying images by their organized complexity.

Algorithmic information theory[16, 4] is a field of computer science that formalizes the concepts of simplicity and complexity by means of information. Many applications of the theory of algorithmic information have been developed to date. For a detailed survey see [13, 17]. None seems, however, to have exploited the concept of logical depth so far, nor to have used an algorithmic information-theoretic approach to the problem of image classification. Logical depth was originally identified with what is usually believed to be the right measure for evaluating the complexity of real-world objects such as living beings. Hence its alternative designation: physical complexity (By Bennett himself). This is because the concept of logical depth takes into account the plausible history of an object as an unfolding phenomenon. It combines the concept of the shortest possible description of an object with the time that it takes to evolve to its current state. Unlike the application of the concept of algorithmic complexity by itself, the addition of logical depth results in a reasonable characterization of the organizational (physical) complexity of an object, as will be elaborated in the following sections.

The main hypothesis of this paper is that images can be used to determine the physical complexity of an object (or a part of it) [12] at a specific scale and level, or if preferred, to determine the complexity of the image containing information about an object. And as such, they cannot all be presumed to have the same history, but rather to span a wide and measurable spectrum of evolutions leading to patterns with different complexities, ranging from random-looking to highly organized. To test the applicability of the concept of logical depth to a real-world problem, we first approximate the shortest description of an image by way of current available lossless compression algorithms, then the decompression times are estimated as an approximation to the logical depth of the image. This allows us to assess a relative measure and to produce a classification based in this measure.

The paper is organized as follows: In section II the theoretical background that will constitute the formal basis for the proposed method is introduced. In Section III we describe the method and the battery of tests to evaluate it. Finally, in section IV we present the results followed by the conclusions in V.

II. THEORETICAL BASIS

A. Algorithmic complexity

The idea that information can be measured and described as a quantity using bits as units was first introduced by Shannon [21]. In computer science, objects can always be viewed as binary strings[1]. Thus we will refer to objects and strings interchangeably in this discussion.

The information content in bits of a string can be described by the shortest possible version of the string. The descriptional complexity of a string of bits can be defined in terms of algorithmic complexity[1] because given a rule-based machine, the machine can follow the algorithm and make a copy of an object from its description (in our case the image of the represented object). The relationship between Shannon’s information theory and the theory of algorithmic complexity is described in [14]. The information content of a string of bits can be defined as the length (in bits) of the smallest program which produces the said string as output.

This enables us to define a measure such that a string has maximal (algorithmic) complexity if the shortest algorithm able to generate it is not significantly shorter than the string itself. In other words, there is no more economical way of communicating the information that it contains than by transmitting the string in its entirety.

In algorithmic information theory a string is algorithmically random if it is incompressible. The difference in length between a string and the shortest algorithm able to generate it is

1 Strings of characters, just like computer programs can also always be translated, with no loss of information, into bit strings.

2 Also known under the names program-size complexity and Kolmogorov-Chaitin complexity.
the string’s degree of complexity. A string of low complexity is highly compressible, as the information that it contains can be encoded in an algorithm much shorter than the string itself, while a string of high complexity is hard to compress because, in a fixed language, its shortest possible description is itself.

A classic example is a string composed by an alternation of bits such as (01)$^n$ that can be described by "n repetitions of 01", despite the number n. So the string can grow quite fast while the description grows by only about $log_2(n)$. On the contrary, a random-looking finite string such as 011001011010110101 may not have a shorter description than itself.

Algorithmic complexity is inversely related to the degree of regularity of a string. Any patterns in a string constitute redundancy: they enable one portion of the string to be recovered from another, allowing a more concise description. This is what is exploited by a lossless image compression algorithm, designed to find regularities in their bi-dimensional array.

The ability of a universal Turing machine to simulate any algorithmic process has motivated and justified the use of universal Turing machines as the language framework within which definitions and properties of mathematical objects are given and studied.

The algorithmic complexity of a binary string $s$ with respect to a universal Turing machine $U$ of length $|p|$ that produces $s$ as output:

$$K_U(s) = \min\{|p|, U(p) = s\}$$

Due to the halting problem, a given program which computes only correct answers can compute the exact algorithmic complexity of at most a finite set of strings.

Since $K(s)$ is the length of the shortest compressed form of $s$, i.e. the best possible compression (albeit with some fixed additive constant), one can approximate $K$ by compression means, using current state lossless compression programs. The better these programs compress, the better the approximations. The length of the compressed string in bits together with the decompressor in bits is an approximation of the shortest program generating the string. The compressed version of $s$ is an upper bound of its algorithmic complexity and therefore an approximation to $K(s)$.

B. Bennett’s logical depth

A measure of the complexity of a string can be arrived at by combining the notions of algorithmic information content and time complexity. According to the concept of logical depth, the complexity of a string is best defined by the time that an unfolding process takes to reproduce the string from its shortest description. The longer it takes the more complex. Hence complex objects are those which can be seen as "containing internal evidence of a nontrivial causal history."

Unlike algorithmic complexity, which assigns a high complexity to both random and highly organized objects placing them at the same level, logical depth assigns a low complexity to both random and trivial objects, thus being more in keeping with our intuitive sense of the complexity of physical objects because trivial and random objects are intuitively easy to produce, have no long history and unfold quickly. A clear, detailed explanation pointing out the convenience of the concept of logical depth as a measure of organized complexity as opposed to the usual plain algorithmic complexity is provided in [10].

A typical example that illustrates the concept of logical depth and its characterization as a measure of physical complexity is exemplified in sequence of fair coin tosses. Such a sequence would have high information content (algorithmic complexity) because the outcomes are random, but little value (logical depth) because they are easily generated and carry no message, no meaning. The string 1111...111 is also logically shallow. Its minimal program, whilst very small, requires little time to evaluate. In contrast, the binary representation of the number $\pi$ is logically deep, because although it is highly compressed (by identifying that it is $\pi$), its generating algorithm takes a lot of time, relatively speaking, to output any number of digits of its decimal expansion.

Unlike in algorithmic complexity, in real-world computation physical resources usually do matter. A computation is seen as inherently difficult if computing it requires a large amount of time. In the real world, some objects such as a gas filling a room or a perfect crystal are intuitively trivial because they unfold in zero computing time, while others such as living beings contain internal evidence of a nontrivial causal history.

For finite strings, one of Bennett’s formal approaches to the logical depth of a string is defined as follows:

Let $s$ be a string and $d$ a significance parameter. A string’s depth at significance $s$, is given by

$$D_d(s) = \min\{|p| : \langle p \rangle - |p'| < d \lor (U(p) = s)\}$$

with $|p'|$ the length of the shortest program for $s$, (therefore $K(s)$). In other words, $D_d(s)$ is the least time $T$ required to compute $s$ from a $d$-incompressible program $p$ on a Turing machine $U$. The concept of logical depth is an attempt to connect the description of an object with the time that it might take to produce it.

Charles Bennett’s claim is that it is this time connecting the current state of an object with its plausible origin that is the appropriate measure of its complexity in physical terms. Bennett’s main motivation was actually to provide a reasonable means of measuring the physical complexity of real-world objects, since the notion of logical depth stratifies them, placing those one would expect to be complex above those that one would expect to be simple. Logical depth does this by taking into consideration the time that a process takes to produce the current state of an object from its plausible origin.

3 Under Church’s hypothesis.
4 The problem of deciding whether, given a program and an input, the program will eventually halt when run with that input.
5 Bennett provides a careful development of the notion of logical depth taking into account near-shortest programs as well as the shortest one, for a reasonably robust and machine-independent measure.
6 Although random strings are hardly compressible they can be quickly reproduced by a ‘print program’ containing only a verbatim description of the data.
Algorithmic complexity and logical depth are intimately related. The latter depends on the former because one has to first find an approximation to the algorithmic complexity of the data by compressing it in order to evaluate the time that the decompression process takes to reconstruct the original uncompressed version of the data. While the compression lengths are approximations of its algorithmic complexity the evaluation of the decompression times are an approximation of the logical depth.

A common drawback of these concepts is that both $K(s)$ and $D(s)$ as functions of $s$ are uncomputable. Yet it is not difficult to arrive at approximations by using compression algorithms. These approximations may depend on the compression algorithm, but $K$ only does so by an additive constant and $D$ by a linear polynomial.

An image can be coded simply as a string $s$ over a finite alphabet, say the binary alphabet—a black and white image. We will denote by $K_c$ and $D_c$ the approximations obtained by means of a compression algorithm $c$. When the algorithmic complexity $K(s)$ is approximated by a lossless compression algorithm, this approximation corresponds to an upper-bound of $K(s)$ [18].

As will be shown in section IV the method captured the essential (visual) features of the objects contained in the images, correctly characterizing their relative complexity. The method herein provides a general procedure for image characterization and classification.

III. Methodology

In the wake of Bennett’s seminal ideas, there seems to have been no previous attempt to implement an application of these ideas to a real-world problem. In order to assess the feasibility of an application of the concept to the problem of image characterization and classification by complexity we performed a series of experiments of gradually increasing sophistication, starting from fully controlled experiments and proceeding to the use of the best known compression algorithms over a large dataset.

The battery of tests consisted of a series of images devised to verify different aspects of the methodology and a more realistic dataset, indicating whether the results were stable enough to yield the same values after each experiment repetition and whether they were consistent with the theory and consonant with the intuition of a complex vs. a simple object.

The first experiments consisted in controlling all the environmental parameters involved, from the data to the compression algorithm, in order to test the first attempts to calculate decompression times. A test to measure the correlation between image sizes (random vs. uniformly colored images) and decompression times was carried out. The results are in section [IV-A1]

A second test in section [IV-A2] consisted of a series of images meant to evaluate the change and magnitude of the decompression times when manipulating the internal structure of an image. That is, it served to verify that the decompression times decreased as expected when the content of a random image was artificially manipulated to make it more simple.

This consisted of a set of images in which uniform structures consisting of large single-bit strings of a fixed size were randomly inserted into the images containing originally only pseudo-random generated pixels.

The framework consisted of using a toy compression program involving an algorithm grouping runs of the same bit replaced by a couple of values: the replaced bit followed by the number of times the bit was found. No further allowances were made, either for dealing with special cases or for detecting any other kinds of patterns. We wanted the algorithm and the data to be as simple as possible to be fully controlled in every detail to better understand the role of a structure of increasing size in the decompression time. It consisted of the series of computer-generated random images shown in figure 1.

This was a way of injecting structure into the images in order to study the behavior of the compression algorithm and assure control of the variables involved in order to better understand the next set of results.

A procedure described in [IV-A4] was devised to test the complementary case of the previous test to verify that the decompression times increased when the content of an image was artificially manipulated, transforming it from a simple state (an all-white image) to a more complex state by generating images with an increasing number of structures. A collection of one hundred images with an increasing number of straight lines randomly depicted was artificially generated (see figure 2). The process led to interesting results described in [IV-A4] showing the robustness of the method.

Three more tests to calibrate the measure based on logical depth are proposed in [IV-B] with three different series of images. The first two series of images were computer generated. The third one was a set of pictures of a real-world object. The description of the 3 series follows:

1) A series of images consisting of random points converted to black and white with a probability of a point’s existing at any given location having a certain value depending on a threshold. This can be seen as variation
in information content since the ratio of black and white dots varies from high to low from image to image depending on the threshold value.

2) Cellular automaton rule 30, then another image with the same automaton superposed upon itself (rotated 90°) followed by the same automaton to which was applied a function introducing random bits to 50 percent of the image pixels. We expect a greater standard deviation for the series of cellular automata because they come in pairs (meaning each image comes with its inverse), and permutations should be common and ought to be between these pairs because intuitively they should have the same complexity.

3) A wall and the same wall but viewed from a closer vantage point.

The last test in IV-C is the main result of the paper, showing the full power of the method. The experiment was performed on a larger dataset of randomly chosen pictures, 56 black-and-white pictures coming from different sources and representing objects of all kinds. Some were pictures of actual objects produced by nature and humans, such as car plans, pictures of faces, handwriting, drawings, walls, insects, and so on. Others were computer-generated, such as straight lines, Peano curves, fractals, cellular automata, monochrome and pseudo-random generated images.

The selection of images in section IV-C was made by hand bearing in mind the objective of spanning a large variety of objects covering a range of seemingly different complexities. We chose images of faces, people, engines and electronic boards, images singled out for their high degree of complexity, being each usually the result of a relatively long history, whether they were human artifacts or long-lived natural entities. The image bearing the name "table" is a table of numerical values, a computer spreadsheet, likely characterized by a significant level of complexity. Images tagged "people" are pictures of a group of people taken from a distance. The tag "inv" following picture names indicates that they are the color inversions of other images in the same dataset that we expected would be close in complexity, and they are presented side-by-side with their non-inverted versions. "Writing" refers to handwriting by a human being, which should also have a significant level of complexity, a level of complexity approaching that of other handwritten pieces such as the image tagged "formulae" which depicts formulae rather than words. "Watch" is the internal engine of a watch. "Cpu B" is a picture of a microprocessor showing less detail than "cpu A". "Escher" is a painting of tiles by Escher. "Paper" is a corrugated sheet of paper. "Shadows B" are the shadows produced by sea tides. "Symmetric B" is "shadows B" repeated 4 times. "Symmetric A" is "symmetric B" repeated 4 times. "Rectangle C" is "rectangle B" repeated 4 times. Those images tagged "Peano curve" are the space filling curves. "Periodic" and "Alternated" are of a similar type, and we thought they would have low complexity, being simple images. "Random" was a computer-generated image, technically pseudo-random.

This random-generated picture would illustrate for us the difference, if any, between algorithmic complexity and logical depth. Based on its algorithmic complexity the random image should be ranked at the top, whereas its logical depth would place it at the bottom. All pictures were randomly chosen from the web, transformed to B&W and reduced to 600 × 600 pixels.

A. Towards the choice of B&W algorithm

Deflate is a compliant lossless compressor and decompressor available within the zlib package. The Deflate compression compresses data using a combination of the Lempel-Ziv coding algorithm and the Huffman coding. It is one of the most widely-used compression encoding systems.

The Huffman coding assigns short codewords to those input blocks with high probabilities and long codewords to those with low probabilities. In other words, the compressor encodes more frequent sequences with a few bytes and spends more bytes only on rare sequences.

The Lempel-Ziv coding algorithm builds a dictionary and encodes the string by blocks using symbols in the dictionary. The Lempel-Ziv algorithm leads to actual compression when the input data is sufficiently large and there is sufficient redundancy (patterns) in the data. Cover and Thomas show that the Lempel-Ziv compression algorithm provides upper bounds to $K$.

B. Compression method

Since digital images are just strings of values arranged in a special way, one can use image compression techniques to approximate the algorithmic complexity of an image. And through its image, the algorithmic complexity of the object represented by the image (at the scale and level of detail of the said image).

A representative sample of lossless image-compression algorithms (GIF, TIFF and JPG 2000) was tested in order to compare the decompression runtimes of the testing images. Although we experimented with these lossless compression algorithms, we ended up choosing PNG for several reasons, including its stability and the flexibility afforded by the ability to use open-source developed optimizers for further compression.

The Portable Network Graphics (PNG) is a bitmapped image format that employs lossless data compression. It uses a 2-stage compression process, a pre-compression or filtering, and Deflate. The filtering process is particularly important in relating separate rows, since Deflate alone has no understanding that an image is a bi-dimensional array, and instead just sees the image data as a stream of bytes.

There are several types of filters embedded in image compression algorithms which exploit regularities found in a stream of data based on the corresponding byte of the pixel.

---

7 The images are available online under the paper title at [http://www.mathrix.org/experimentalAIT/](http://www.mathrix.org/experimentalAIT/)

8 Unlike the printed version, if seen in electronic form the images can be zoomed in, allowing better visualization. They are also available online under the paper title at [http://www.mathrix.org/experimentalAIT/](http://www.mathrix.org/experimentalAIT/)

to the left, above, above and to the left, or any combination thereof, and encode the difference between the predicted value and the actual value.

$$\begin{array}{ccc}
X_1 & X_2 & X_3 \\
X_4 & X_5 & X_6 \\
X_7 & X_8 & \\
\end{array}$$

Fig. 3. Image pixel neighborhood. By applying different filters a lossless image compression algorithm uses the data contained in the pixels $x_1, \ldots, x_8$ to predict the value of another pixel $x_9$ that may save space, allowing the compression of the said image without losing any information. The number of combination tested is finite and limited by the compression algorithm. Yet one can optimize the search of a successful combination by setting the compression algorithm to try harder and spend more time trying to better compress the data.

Compression can be further improved by so-called PNG-optimizers using more filter methods and several other lossless data compression algorithms. Among these optimizers are Pngcrush, and AdvanceCOMP, two of the most popular open-source optimizers. They tried various compression methods and were able to reduce the PNG files by about 10 to 20% of their original length. AdvanceCOMP recompresses PNG files (and other file formats) using the Deflate 7-Zip implementation. The 7-Zip Deflate encoder effectively extends the scope of Deflate further by performing a much more detailed search of compression possibilities at the expense of significant further processor time spent searching, which for this experiment was not a matter of concern. 7-Zip Deflate also uses the LZMA algorithm, an improved and optimized version of the LZ77 compression algorithm. The LZMA algorithm divides the data into packets, each packet describing either a single byte or an LZ77 sequence with its length and distance implicitly or explicitly encoded.

As a sort of verification, we ran a popular zip-based commercial compressor, set to the maximum possible compression, over the already compressed and optimized files. The zip-based archiver was unable to further compress any of the files (they were actually always a little larger in size).

C. Using decompression times to estimate complexity

Inflate is the decoding process that takes a Deflate bit stream for decompression and correctly produces the original full-size data file. To decode an LZW-compressed file, one needs to know the dictionary encoding the matched strings in the original data to be reconstructed. The decoding algorithm for LZ77 works by reading a value from the encoded input and outputting the corresponding string from the shorter file description.

It is this decoding information that Inflate takes when importing a PNG image for display, so the lengthier the directions for decoding the longer the time it takes. We are interested in these compression/decompression processes, particularly the compression size and the decompression time, as an approximation of the algorithmic complexity and the logical depth of an image.

The decompression directions for trivial or random-looking objects are simple to follow, with the decompression process taking only a small fraction of time, simply because the compression algorithm either compresses very well and the decompression is just straightforward (trivial case) or does not compress at all (random case). Longer runtimes, however, are usually the result of a process following a set of time-consuming decompression instructions, hence a complex process.

D. Timing method

The execution time was given by the Mathematica function Timing. The function Timing evaluates an expression and returns a list of the time used in seconds, together with the result obtained. The function includes only CPU time spent in the Mathematica kernel.

The fact that several processes run concurrently in computing systems as part of their normal operation is one of the greatest difficulties faced in attempting to measure with accuracy the time that a decompression process takes, even when it is isolated from all other computer processes. This instability is due to the fact that the internal operation of the computer comes in several layers, mostly at the operating system level. In order to avoid measurement perturbations as much as possible, several stabilizing measures were undertaken:

- Most computer batch processes and operating system services were disabled, including services such as wireless and bluetooth and energy saving features, such as the hard drive sleeping and display dimming feature.
- The microprocessor was warmed up before each experiment by running an equivalent process (e.g. a mock experiment run) before running the actual one in order to have the fan and everything else already working at a high rate (like preheating an oven).
- The cache memory was cleared after each function call using the Mathematica function ClearSystemCache. No history was saved in RAM memory.
- A different order of images was used for each experiment run. The result was averaged with at least 30 runs, each compressing the images in a different random order. This helped to define a confidence level on the basis of the standard deviation of the runs, when they are normally distributed. A confidence level with which we were satisfied and at which we arrived in a reasonable amount of time. Further runs showed no further improvement.

11 Syntax example: e.g. advdef -z -4 *.png (4 indicating the so-called “insane” compression according to the developers). More info: http://advancemame.sourceforge.net
12 More technical details are given in http://www.7-zip.org/7z.html.
13 A useful website showing a benchmark of compression algorithms is at urlhttp://tukaani.org/lzma/benchmarks.html.
14Timed on two different computers for validation. On a MacBook Intel Core 2 Duo 2GHz, 2048MB DDR2 667Mhz with a solid-state drive (SSD) and on MacBook Pro Intel Core 2 Duo 2.26Ghz, 4096MB DDR3 1067Mhz with a traditional hard disk drive (HDD), both running Mac OS X Version 10.6.1 (Snow Leopard). The MacBook Pro was always twice as fast on average.
15 To understand the number of layers of complexity involved in a modern computing system see Tanenbaum’s book on operating systems.
Measurements stabilization was reached after about 20 to 30 runs.

The presence of some perturbations in the time measure values were unavoidable due to lack of complete control over all the computing system parameters. As the following battery of tests will show, one can reduce and statistically curtail the impact of this uncertainty to reasonable levels.

IV. RESULTS

A. Controlled experiments

![Decompression times and standard deviations for images increasing in size.](image1)

Fig. 4. Decompression times and standard deviations for monochromatic images increasing in size.

![Decompression times and standard deviations for images with (pseudo) random noise increasing in size.](image2)

Fig. 5. Decompression times and standard deviations for images with (pseudo) random noise increasing in size.

1) Image size uncertainty variation: Figures 4 and 5 show that decompression times using the PNG algorithm and the optimizers increase in linear fashion when image sizes increase linearly. The distributions of standard deviations in the same figures show no particular tendency other than suggesting that the standard deviations are likely due to random perturbations and not to a bias in the methodology.

A comparison with figure 6 shows that standard deviations for images containing random noise remained always low, while for uniformly colored images standard deviations were significantly larger.

![Standard deviations behavior.](image3)

Fig. 6. For uniformly colored images standard deviations of decompression times were larger in average. For random images the standard deviations were smaller and compact. The size of the image seemed to have no impact in the behavior of the standard deviations for either case.

2) Fully controlled test: For this test only, we used a toy compression algorithm designed to be as simplest as possible in order to estimate the uncertainty due to system perturbations. We devised a test in which we would have full control of the data and the compression algorithm. By understanding the processes we would be able to predict compression rates and decompression uncertainties better, and quantify the uncertainty of the measurement of decompression times in the general case (i.e. using other compression algorithms).

It was found that the algorithmic complexity approximation (the file size) decreased linearly at a rate of 1990 bits per image, the same as the number of regular inserted bits per image, taking about \( \ln(2000) \) to encode the compressed regular string, as the would have predicted. One can also predict the decompression runtime by calculating the slope of the decompression rate. Each insertion of 2000 regular bits into the random image took 0.00356 seconds less each time, fitting the estimation computed by the rate ratio.

The standard deviations behavior suggest that the more random an image the less stable the decompression time and the more regular the more stable. This may be explained by the number of operations that the toy compression algorithm uses for encoding a regular string. In the extreme case of a uniformly colored image, the code comes up with a single loop operation to reproduce the image, taking only one operation time. On the other hand, when an image is random, wholly or partially, the number of operations is larger because the algorithm finds small patterns everywhere, patterns that have a timing cost per operation performed. Each loop operation using the Table function in Mathematica takes 0.000025 seconds on average, while each iteration takes only about 0.00012 seconds on average, suggesting that it is the number of operations that determines the runtime.

![Using the toy compression algorithm, the larger the white region in an image with random noise the shorter the decompression runtime and the lower the standard deviations.](image4)

The standard deviation of the mean 0.0000728 is smaller
than the average of the standard deviations of each point, which is 0.000381. This shows that runtime perturbations were not significant enough.

3) Using the PNG compression algorithm: This time we proceeded to apply the PNG compressor algorithm together with the optimizers (Pngcrush and AdvancedCOMP) to the same computer-generated images with random noise used in the previous section, confident that we understood the variables involved in the previous experiment, and that everything seemed to be controlled.

Figure 9 shows that the PNG compression algorithm followed the same trend as the toy compression algorithm, namely the more uniformity in a random image the greater the compression. PNG however achieved greater compression ratios compared to the toy compression algorithm, for obvious reasons.

In figure 9 one can see that there are some minor bumps and hollows all along. Their deviation does not seem however significant from a uniformly distribution, as shown in figure 10, suggesting bias toward no particular direction. As was the case with the toy compression algorithm, by using the PNG algorithmic complexity decreased as expected—upon the insertion of uniform strings.

Figure 11 (left) is that the standard deviations remained the same on average, suggesting that they were not related to the image, but to external processes. The process of randomly depicting lines seems to have a low maximum limit of complexity, which is rapidly reached. Nevertheless, eight bins of images with significantly different decompression time values, i.e. with non-overlapping standard deviations, were identified. What figure 12 suggests is that increasing the number of lines can only lead to a limited maximum degree of complexity bounded by a convex curve. The eight significantly different images have been selected according to jumps that were maximizing their differences: $n = \Delta D/\sigma$ with $\Delta = \max\{D(I_i) | i \in \{1, \ldots, n\}\} - \min\{D(I_i) | i \in \{1, \ldots, n\}\}$, with $I_i$ each image in the set, and $D$ the logical depth.

For the discontinuity in the graph, one hypothesis is that the compression algorithm surpasses a threshold where it favors some regularities over others, producing jumps in the decompression times. For a certain quantity of randomly depicted lines, the limit remains stable for a while once reached, until the moment when the increasing number of lines depicted fill up the space and the configuration reaches its lowest complexity by decompression time, when it begins to approach a phase in which it resembles a uniformly colored image.

The decompression time depicted in figure 12 turned out to be very interesting suggesting what one might expect for this kind of experiment: A path traced between two monochromatic (fully colored) images, an initial all-white image succeeded by images of increasing complexity comprising random lines, and ending at the horizontal departure line in a final monochromatic almost all-black image, when the space becomes entirely filled with black lines.

B. Calibration tests

The following are the classifications of series 1, 2 and 3 according to their decompression runtimes as described in the methodology section.

- Series 1 classification (figure 13): Images labeled 1 and 2 never came last, while images 4 and 5 never came first. The first half of the images remained the same run

\[ \text{(with Pngcrush and AdvanceCOMP)} \]
As was expected (see section III), a greater standard deviation for the series of cellular automata was found due to the image pairing. Each image occurred in tandem with its inverse, but permutations were common— and expected— between members of pairs, which intuition tells us ought to have the same complexity. The expected permutations produced a larger standard deviation.

C. Algorithmic complexity classification

The following experiments were carried out as described in section III using 56 black-and-white images spanning a range of different kind of objects each seemingly having different complexity which we could intuitively gauge more or less accurately. Each image has a very short description, followed by the image itself and by the approximated values of $K_c$ (figure 16) and $D_c$ (figure 17) for our general compression algorithm $c$.

The classification in figure 16 presents the images ranked according to their compressed lengths using the PNG image compression together with the PNG optimizers. It goes from larger to smaller, and as can be seen, the more random-looking or highly structured, the better classified, while trivial images come last. One can verify that the procedure is invariant to simple transformations such as inversions, since the images

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
series number & description & Std. deviation \\
\hline
2 & Cellular automata & 0.0004656 \\
3 & Wall zooming & 0.000456 \\
1 & Random points & 0.000549 \\
\hline
\end{tabular}
\caption{Standard deviations per series sorted from greatest to lowest standard deviation.}
\end{table}

The classification in figure 16 presents the images ranked according to their compressed lengths using the PNG image compression together with the PNG optimizers. It goes from larger to smaller, and as can be seen, the more random-looking or highly structured, the better classified, while trivial images come last. One can verify that the procedure is invariant to simple transformations such as inversions, since the images

17 The images are available online under the paper title at [http://www.mathrix.org/experimentalAIT/](http://www.mathrix.org/experimentalAIT/)
and their inversions are always side-by-side, indicating that the compression algorithm behaves just as expected. One would hardly say, however, that a random image is physically complex (random processes, like a gas filling a room, unfold in zero time and seem to require no great computational power), which is why plain algorithmic complexity does not help to distinguish between complexity associated with randomness and complexity associated with highly structured objects.

### D. Logical depth classification

The classification in figure 17 goes from greater to smaller logical depth based on the decompression runtimes. The number at the bottom is the time in seconds that the PNG algorithm took to decompress the images when applied to their compressed versions. Images we gauged to have the highest physical complexity come first. It is also worth mentioning that the images and their inverted versions remained close to each other, meaning that they were always in the same significantly different complexity group, which is also just what we expected. This also indicates the soundness of the procedure, since images and their inversions should be equal in complexity, and therefore equal in complexity to the object as well, at a commensurate scale and level of detail.

Figure 18 shows some of the jumps seen before in the experiments in section IV-A4. We think they may be due to the behavior of the compression algorithm. The compression algorithm applies several filters and it may favor some regularities over others that are better (faster) decoded after certain threshold producing these jumps.

Images were grouped in 8 significantly different groups (with different decompression times and therefore seemingly different logical depth). Formally, \( \bar{x}(g_i) \pm \sigma(g_i) > \bar{x}(g_j) \pm \sigma(g_j) \) for any two different group indexes \( i, j \in \{A, B, \ldots, H\} \).

The average difference between the largest and the smallest \( K_c \) value was about 62090 bits, while the average difference between the largest and the smallest \( D_c \) was 0.095 seconds. The largest calculated compressed image size (image 1) was 159.8 times larger than the shortest compressed image size (image 56). The largest evaluated decompression time (image 5) was 2.46 times larger than the shortest calculated decompression time (image 53).

No significant statistical correlation between \( K_c \) and \( D_c \) was found, indicating that the is actually evaluating two different complexity measures. Figure 19 and 20 illustrate the classification values from \( K_c \) to 8 significant different decompression time (\( D_c \)) groups according to each of the two measures. The ranking of images based on their decompression times correspond to the intuitive ranking resulting from a visual inspection, with things like microprocessors, human faces, cities, engines and fractals figuring at the top as the most complex objects as shown in figure 21 (group A).
while random-looking images, ranked high by algorithmic complexity, are sent to the bottom (group G) according to the logical depth expectation, classified together with other trivial images such as the uniformly colored (group H) indicating the characteristic feature of the measure of logical depth. A gradation of different complexities can be found in the groups between, gradually increasing in complexity from bottom to top.

V. CONCLUSIONS

Along the way we think we have shown that:
1) The concept of logical depth can be implemented as a feasible and applicable method to approach a real-world problem
2) After studying several cases and tested several compression algorithms, the method described in this paper has shown to work and to be of use for identifying and classifying images by their apparent physical complexity.
3) The procedure described herein constitutes an unsupervised method for evaluating the information content of an image by physical complexity.
4) As the theory predicted, logical depth yields a reasonable measure of complexity that is different from the measure obtained by considering algorithmic complexity alone, while being in accordance with one’s intuitive expectations of greater and lesser complexity.

Fig. 21. Significant different groups by decreasing decompression time.

REFERENCES


Hector Zenil is a final year PhD candidate in computer science at the University of Science and Technology of Lille (Laboratoire d’Informatique Fondamentale de Lille) and in philosophy of mathematics at the IHPST (Paris I/ENS Ulm/CNRS), preparing both dissertations on algorithmic randomness. He is also a senior research associate at Wolfram Research, Inc., a member of the Turing Centenary Advisory Committee and editor of Randomness through Computation, a book forthcoming this year to be published by World Scientific. Home page: http://zenil.mathrix.org/

Jean-Paul Delahaye is professor of computer science at the University of Science and Technology of Lille (Lifl), a branch of the National Center for Scientific Research (CNRS) based at the university. His work focuses on computational game theory (such as the iterated prisoner’s dilemma) and complexity theory (such as Kolmogorov complexity) and applications of these theories to genetic analysis and, recently, to economics. Home page: http://www2.lifl.fr/~delahaye/

Cédric Gaucherel has been a Research Scientist with the French National Center of Agricultural Research (INRA, AMAP Laboratory), since 2006. He received a Ph.D. degree in astrophysics, and a D.Sc. degree in ecology from the University of Orsay, Paris (France), in 1997 and 2006. His background is in physics allowing him to transfer skills in modeling and spatial analyses into environmental topics. He has developed mechanistic models of various ecosystem components such as vegetation distributions, animal populations, human influences, and climatologic variables, with the ultimate aim to merge them into an integrated view of the ecosystem complexity. Home page: http://unramap.cirad.fr/amap3/cm/index.php?page=cedric-gaucherel