Algorithmic Specified Complexity in the Game of Life

Winston Ewert, William Dembski, *Senior Member, IEEE,* and Robert J. Marks, II, *Fellow, IEEE*

***Abstract*—Algorithmic specified complexity (ASC) measures the degree to which an object is meaningful. Neither fundamental Shannon nor Kolmogorov information models are equipped to do so. ASC uses performance context in an information theoretic framework to measure the degree of specified complexity in bits. To illustrate, we apply ASC to Conway's *Game of Life* to differentiate patterns designed by programmers from those originating by chance. A variety of machines created by Game of Life hobbyists, as expected, exhibit high ASC thereby corroborating ASC's efficacy.**

***Index Terms*—Algorithmic specified complexity, cellular automata, Conway's Game of Life, Kolmogorov information, Kolmogorov-Chaitin-Solomonoff information, Shannon information, specified complexity.**

I. Introduction

BOTH Shannon *et al*. [1], [2] and Kolmogorov–Chaitin–Solomonoff (KCS)[1] [2]–[9] measures of information are famous for not being able to measure meaning. A DVD containing the movie *Braveheart* and a DVD full of correlated random noise can both require the same Shannon information as measured in bytes. Likewise, a maximally compressed text file with fixed byte size can either contain a classic European novel or can correspond to random meaningless alphanumeric characters. The KCS measure of information is therefore also not able to, by itself, measure informational meaning.

We propose an information theoretic method to measure meaning [10], [11]. Fundamentally, we model meaning to be in the context of the observer. A page filled with Kanji symbols will have little meaning to someone who neither speaks nor reads Japanese. Likewise, a machine is an arrangement of parts that exhibit some meaningful function whose appreciation requires context. The distinguishing characteristic of machines is that the parts themselves are not responsible for the machine's functionality, but rather they are only functional due to the particular arrangement of the parts. Almost any other arrangement of the same parts would not produce anything interesting. A functioning computational machine is more meaningful than a large drawer full of computer parts. We appreciate the meaningful functionality of the machine only because we have the contextual experience to recognize what the machine can do or is capable of doing.

The arranging of a large collection of parts into a working machine is highly improbable. However, *any* arrangement would be improbable regardless of whether the configuration had any functionality whatsoever. For this reason, neither Shannon nor KCS information models are capable of directly measuring meaning. Functional machines are specified—they follow some independent pattern. When something is both improbable and specified, we say that it exhibits *specified complexity*. An elaborate functional machine exemplifies high specified complexity. We propose a model, algorithmic specified complexity (ACS), whereby specified complexity can be measured in bits.

ASC was introduced by Dembski [12]. The topic has been developed and illustrated with a number of elementary examples [10], [11]. Durston *et al*.'s functional information model [13] is a special case. The approach differs from conventional signal and image detection [14]–[19] including matched filter correlation identification of the index of one of a number of library images [20]–[22]. Alternately, ACS uses conditional KCS complexity to measure the minimum information required to reproduce an image losslessly (i.e., exactly - pixel by pixel) in the presence of context. Use of KCS complexity has been used elsewhere to measure meaning in other ways. Kolmogorov sufficient statistics [2], [23] can be used in a two-part procedure. First, the degree to which an object deviates from random is ascertained. What remains is algorithmically random. The algorithmically nonrandom portion of the object is then said to capture the meaning of the object [24]. The term meaning here is solely determined by the internal structure of the object under consideration and does not directly consider the context available to the observer as is done in ASC.

A. Game of Life

To illustrate quantitative measurement of specified complexity, we examine the well-defined universe of Conway's *Game of Life* [25]. In the Game of Life, a 2-D grid of living and dead cells live and die based on simple rules. At each time step a cell is determined to be alive or dead depending on the state of its neighbors in the previous generation.

Within the Game of Life, interesting and elaborate forms have been discovered and invented. These are particular arrangements of living and dead cells that when left to operate by the rules of the game, exhibit meaningful functionality.

Manuscript received September 1, 2013; revised February 6, 2014; accepted April 21, 2014. Date of publication August 6, 2014; date of current version March 13, 2015. This paper was recommended by Associate Editor A. Bargiela.

W. Ewert is a Software Engineer in Kirkland, WA 98033 USA.

W. Dembski is with the Discovery Institute, Seattle, WA 98104 USA, and also with the Evolutionary Informatics Laboratory, McGregor, TX, USA.

R. J. Marks II is with the Electrical and Computer Engineering, Baylor University, Waco, TX 76798-7356, USA (e-mail: robert_marks@baylor.edu).

Digital Object Identifier 10.1109/TSMC.2014.2331917

[1]Sometimes referred to as only Kolmogorov complexity or Kolmogorov information.

Some oscillate, some move, some produce other patterns, etc. Some of these are simple enough that they arise from random configurations of cell space. Others required careful construction, such as the very large Gemini [26]. Our goal is to formulate and apply specified complexity measures to these patterns. We would like to be able to quantify what separates a simple glider, readily produced from almost any randomly configured soup, from Gemini—a large, complex design whose formation by chance is probabilistically minuscule. Likewise, we would like to be able to differentiate the functionality of Gemini from a soup of randomly chosen pixels over a similarly sized field of grid squares.

A highly probable object can be explained by randomness, but it will lack complexity and thus not have specified complexity. Conversely, any sample of random noise will be improbable, but will lack specification and thus also lack specified complexity. In order to have specified complexity, both components must be present. The object must exhibit a describable functioning pattern while being improbable.

Our paper differs from the study of emergence in cellular automata first proposed by von Neumann [27], [28] for investigating self-reproduction. The study of dynamic cellular automata properties, popularized by Wolfram [29], deals largely with investigation of the temporal development of emergent behavior [30]. As an example, the set of all initializations that lead to the same emergent behavior, dubbed the basin of attraction, has been studied as a model for evolution of artificial life with application to the modeling of memory and neural networks [30]–[33]. Akin to biological swarms obeying simple nonlinear rules [34]–[36], emergence is difficult to predict thereby necessitating largely experimental analysis.

Our paper, in contrast, deals with measuring meaning in existing objects. In assessing the specified complexity of *Braveheart* versus the DVD of noise, we are not interested in finding the dynamics of where each DVD came from but, rather, how we assess what has meaning and what does not. It is in this context that we deal with assessment of specified complexity of the Game of Life.

II. Algorithmic Specified Complexity

How does one measure specification? One possibility is to use KCS complexity [4], [5], [9]. An introduction to the topic is in the widely used information theory text by Cover and Thomas [2]. A more advanced presentation is given by Li and Vitâanyi [23]. KCS complexity or variations thereof have been previously proposed as a way to measure specification [12], [37], [38].

KCS complexity is defined as the length of the shortest computer program, p, in the set of all programs, P, that produces a specified output X using a universal Turing machine, U

$$K(X) = \min_{U(p,)=X|p\in P} |p|.$$

Such programs are said to be elite [8]. Conditional KCS complexity [3] allows programs to have input, Y, which is not considered a part of the elite program

$$K(X|Y) = \min_{U(p,Y)=X|p\in P} |p|.$$

For our purposes, Y can be considered as context.

An example is Shakespeare's *Hamlet* compressed with two different resources: 1) Y_{alpha} = the English alphabet, including numbers and punctuation and 2) Y_{con} = an exhaustive concordance of the words used in all of Shakespeare's writings [39]. Both resources can be viewed as a code book in which the entries are lexicographically listed and numbered. *Hamlet*, corresponding to the output X, can then either be expressed as a sequence of integers each corresponding to an entry in the alphabet list, or indexed as an entry in the concordance. Shakespeare used 31 534 different words [40]. Although both the alphabet and concordance characterizations bound the conditional KCS complexity, we would expect

$$K\left(X|Y_{con}\right) < K\left(X|Y_{alpha}\right) < K(X).$$

The more specific the context, the smaller the elite program. Either the frequency of occurrence of the words used by Shakespeare or a concordance of words used only in *Hamlet* can be used to reduce the conditional KCS complexity even further. Small conditional KCS complexity can be caused by the following.

1) Placing X in the context of Y and/or
2) A small (unconditional) KCS complexity, $K(X)$.

A small value of $K(X|Y)$ can therefore arise from the small complexity of X and/or from the available context, Y.

A. ASC: Actual and Observed

Algorithmic specified complexity [41] is defined as

$$ASC(X, C, P) = I(X) - K(X|C) \tag{1}$$

where

1) X is the object or event under consideration;
2) C is the context (given information) which can be used to describe the object;
3) $K(X|C)$ is the KCS complexity of object X given context C;
4) $P(X)$ is the probability of X under the given stochastic model;
5) $I(X) = -\log_2(P(X))$ is the corresponding self information.

The ASC measure bears a resemblance to both Shannon [1], [2] and KCS [23] mutual information.

ASC is probabilistically rare in the sense that [42]

$$\Pr[ASC(X, C, P) \geq \alpha] \leq 2^{-\alpha}. \tag{2}$$

For example, the chance of observing ten or more bits of ASC does not exceed $2^{-10} \approx$ one chance in a thousand. ASC provides evidence that a stochastic outcome modeled by the distribution, $P(X)$, does not explain a given object. ASC is incomputable because KCS complexity is incomputable [2]. However, the true KCS complexity is always equal to or less than any achieved lossless compression. This means that the true ASC is always equal to or more than an estimate. We will refer to the known estimate as the observed observed algorithmic specified complexity (OASC). We know that

$$ASC(X, C, P) \geq OASC(X, C, P). \tag{3}$$

The inequality in (2) applies to OASC. From (3), we conclude there is a $k \geq 0$ such that[2]

$$OASC = ASC - k.$$

Thus

$$\begin{aligned}\Pr[OASC \geq \alpha] &= \Pr[ASC - k \geq \alpha]\\ &= \Pr[ASC \geq \alpha + k]\\ &\leq 2^{-\alpha-k}\\ &\leq 2^{-\alpha}. \end{aligned} \tag{4}$$

OASC therefore obeys the same bound as does ASC in (2).

ASC can be nicely illustrated using various functional patterns in Conway's *Game of Life*. The Game of Life and similar systems allow a variety of fascinating behaviors [29]. In the game, determining the probability of a pattern arising from a random configuration of cells is difficult. The complex interactions of patterns arising from such a random configuration makes it difficult to predict what types of patterns will eventually arise. It would be straightforward to calculate the probability of a pattern arising directly from some sort of random pattern generator. However, once the Game of Life rules are applied, determining what patterns would arise from the initial random patterns is nontrivial. In order to approximate the probabilities, we will assume that the probability of a pattern arising is about the same whether or not the rules of the Game of Life are applied, i.e., the rules of the Game of Life do not make interesting patterns much more probable then they would otherwise be.

Objects with high ASC defy explanation by the stochastic process model. Thus, we expect objects with large ASC are designed rather than arising spontaneously. Note, however, we are only approximating the complexity of patterns and the result is only probabilistic. We expect that patterns requiring more design will have higher values of ASC. Smaller designed patterns exist, but it is not possible to conclude that they were not produced by random configurations.

Section III documents the methodology of the paper. We define a mathematical formulation to capture the functionality of various patterns. This can be encoded as a bitstring and a program written to generate the original pattern from this functional description. Section IV uses this methodology to calculate ASC for a variety of patterns found in the Game of Life.

III. Methods

A. Specification

The Game of Life is played on grid of square cells. A cell is either alive (a one) or dead (a zero). A cell's status is determined by the status of other cells around it. Only four rules are followed.

1) *Under-Population:* A living cell with fewer than two live neighbors dies.
2) *Family:* A living cell with two or three live neighbors lives on to the next generation.
3) *Overcrowding:* A living cell with more than three living neighbors dies.
4) *Reproduction:* A dead cell with exactly three living neighbors becomes a living cell.

[2]The arguments of ASC and OASC are always the same so we will henceforth drop them from the notation.

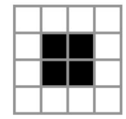


Fig. 1. Block, a simple still life.

As witnessed by videos on YouTube, astonishing functionality can be achieved with these few simple rules [43]–[45]. If the reader is unfamiliar with the diversity achievable with these operations, we encourage them to view these and other short videos demonstrating the Game of Life. The static pictures in this paper do not do justice to the remarkable underlying dynamics. There is also an active users group [46].

The rules for the Game of Life are deterministic. Performance is therefore dictated only by the initially chosen pattern. In order to demonstrate the compression of functional Game of Life patterns, we first devise a contextual mathematical formulation for describing functionality. A method for interpreting this formulation is considered to be part of the context. Let X be some arbitrary pattern corresponding to a configuration of living and dead pixels. Let $X\oplus$ be the result of one iteration of the Game of Life applied to X. Suppose that the following equality holds:

$$X = X\oplus.$$

This says that a pattern does not change from one iteration to the next. This is known as a still-life [25], and an example is presented in Fig. 1. A more interesting pattern can be described as

$$X = X \oplus \oplus$$

which can be a pattern that returns to its original state after two iterations. The relationship is also valid for two iterations of a still-life. In order to differentiate a two-iteration flip-flop from a still life form, two equations are required

$$\begin{aligned} X &\neq X\oplus\\ X &= X\oplus^2. \end{aligned} \tag{5}$$

We often need to specify that a rule holds only for some parameter and not for any smaller version of that. We therefore adopt the notation

$$X = X\oplus^i \tag{6}$$

to mean a pattern that repeats in i iterations, but not in less than i iterations. An example for $i = 2$, shown in Fig. 2, is a period-2 oscillator [46] or a flip-flop [25].

One of the more famous Game of Life patterns is the glider. This is a pattern which moves as it iterates. A depiction is shown in Fig. 3. In order to represent movements we introduce arrows, so $X\uparrow$ is the pattern X shifted up one row. Since four

TABLE I
LIBRARY OF AVAILABLE OPERATIONS

Name	Symbol	Meaning
Pattern	X	The pattern being tested
Variable	Y, Z, W	A defined variable
Parameter	i, j, k, l	An integer index
Shift	$\uparrow, \downarrow, \leftarrow, \rightarrow$	The pattern shifted in the specified direction
Intersection	$\cap$	A pattern consisting of all cells live in two patterns
Union	$\cup$	A pattern consisting of all cells live in either of two patterns
Set-Difference	$\setminus$	A pattern with lives cells whenever there is live cell in the first pattern, but not the second.
Pattern	, , , ...	An arbitrary pattern
Integer	1, 5, 7, ...	An arbitrary integer
Math	$+, -, /, \times$	Mathematical operations on two integers
Repeat	(superscript)	Any symbol repeated a specified number of times
Define	$:=$	Defines a variable to an expression
Equal	$=$	Rejects a pattern unless the parameters are equal
Not Equal	$\neq$	Rejects a pattern if the parameters are equal

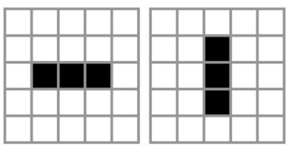


Fig. 2. Blinker, a simple period-2 oscillator.

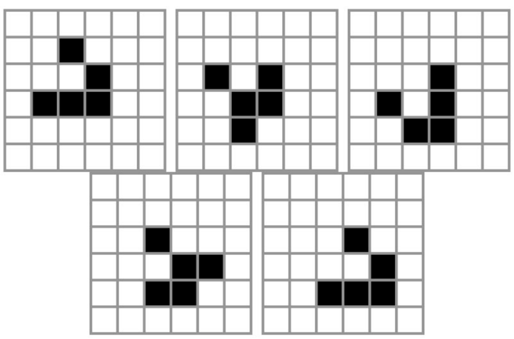


Fig. 3. Glider, a simple spaceship.

iterations regenerate the glider shifted one unit to the right and one unit down, we can write

$$X \downarrow \rightarrow = X \oplus^4. \tag{7}$$

This defines the functionality of moving in the direction and speed of the glider.

We can also simply insert a pattern into the mathematical formulation. For the simplest case, we can say that the pattern is equal to a particular pattern. For example

$X =$ .

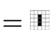
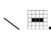
Note that to the right of the equals sign here is a small picture of the glider in Fig. 3. We can also combine patterns, for example taking the union

 $=$  $\cup$ 

or the intersection

 $=$  $\cap$ .

We can also describe a pattern as the set-difference of two other patterns. Since $A \setminus B$ denote elements in A not in B, we have for example

 $=$  $\setminus$ . (8)

At times, it may be useful to define variables. For example

$$Y := X \oplus^{32} \tag{9}$$

$$Y = Y \oplus^{32} \tag{10}$$

where $:=$ denotes "equal to by definition." This reduces to

$$X \oplus^{32} = X \oplus^{64}.$$

Table I provides a listing of operations. The selected set of operation was chosen in attempt to cover all bases which might be useful, but is still arbitrary. Another set of operations with more or less power could be chosen. By using a consistent set of operations between the various examples explored in this paper, we obtain comparable OASC values.

More that one X will display two step oscillation in accordance to $X = X \oplus^2$. In fact, this and other equations will admit an infinite set of patterns that satisfy the description. In order to make an equation description apply to a unique pattern, we can lexicographically order all patterns obeying a description. This can be done by defining a box that bounds the initial pattern. To do so, we will constrain initialization to a finite number of living cells to avoid an infinite number of living cells.

The full ordering can be defined by the following priority set of rules with lower-numbered rules listed first.

1) Smaller number of living cells.
2) Smaller bounding box area.
3) Smaller bounding box width.
4) Lexicographically ordering according to the encoding of cells within a box bounding the living cells. For example, bounding the living cells in the upper left configuration in Fig. 3 and reading left to right then down gives $010001111 = (143)_{10}$.

The first rule could be removed, leaving a consistent system. However, among Game of Life hobbyists, patterns with fewer living cells are considered smaller and maintain this for

TABLE II
BINARY ENCODING

Nullary operations

Symbol	Encoding
X	00000
Y	00001
Z	00010
W	00011
i	00100
j	00101
k	00110
l	00111

Unary Operations

Symbol	Encoding
$\oplus$	01000
$\uparrow$	01001
$\downarrow$	01010
$\leftarrow$	01011
$\rightarrow$	01100

Binary Operations

Symbol	Encoding
$\cap$	01101
$\cup$	01110
$\setminus$	01111
$+$	10000
$-$	10001
$\times$	10010
$/$	10011
$:=$	10100
$=$	10101
$\neq$	10110
(repeat)	10111

Literals

Type	Encoding
Number	11001
Pattern	11010

Special

Type	Encoding
Stop	11111

consistency. This does add some complications to the model which is discussed in the Appendix.

We will append each equation with a number, in the form $\#i$ indicating that we are interested in the ith pattern to fit the equation. Thus, the glider becomes

$$X \downarrow\rightarrow = X\oplus^4, \#0$$

as the smallest pattern which fits the description.

In our discussions of ASC, establishing rules for lexicographical ordering is important whereas assessing the computational resources needed to explicitly populate the list is not.

B. Binary Representation

In order to use the ASC results, we need to encode the mathematical representation as a binary sequence. Each symbol is assigned a 5-bit binary code as specified in Table II. Any valid formula will be encoded as a binary string using those codes. All such formulas will be encoded as prefix-free codes.

Firstly, a number of the operations have zero arguments, known as nullary operators. These are listed first in Table II. Such operations are simply encoded using their 5-bit sequence. Since they have no arguments, their sequence is completed directly after the five bits. As noted, a different set of operations could be chosen that would require a different number of bits to specify. Thus, X will be encoded as 00000 and W will be encoded as 00011. All the nullary operations are trivially prefix free since all have exactly five bits.

An operation that takes a single argument, known as a unary operation, can be encoded with its 5-bit code followed by representation of the subexpression. Thus, $X \uparrow$ can be represented as 0100100000. Since the subexpression can be represented in a prefix free code, we can determine the end of it, and adding five bits to the beginning maintains the prefix-free property.

Operations with two arguments, or binary operations, are encoded using the 5-bit sequence followed by the sequence for the two subexpressions. So $X = X\oplus$ can be recorded as 10101000000100000000. $\oplus^i$ can be recorded as 101110100000100. Note that $\oplus$ usually takes an argument, but this is not needed when it is used as the target of a repeat. As with the unary case, the prefix free nature of the subexpressions allows the construction of the large formula.

The literals in Table II are denoted by the 5-bit code along with an encoding of the integer or pattern. Any positive integer n can be encoded using $\lceil \log_2(n+1) + \log_2 n \rceil + 1$ bits, hereafter $l(n)$ bits in a prefix free code using the Levenstein code [47]. See Section III-C for a discussion of binary encodings for arbitrary patterns.

To declare there are no more operations to be had, we will use the five bit sequence, 11111. Simply concatenating all the equations would not be a prefix-free code since the binary encoding would be a valid prefix to other codes. After the last equation, 11111 is appended as a suffix preventing any longer codes from being valid and making the system prefix free.

To calculate the length of the encoding we add up the following.

1) Five bits for every symbol.
2) $l(n)$ bits for each number n in the equation.
3) The length of the bit encoding of any pattern literals.
4) Five bits for the stop symbol.
5) $l(n)$ bits for the parameters and sequence numbers.

C. Binary Encoding for Patterns

In order to use OASC we need to define the complexity or probability of the patterns. We would like to define the probability based on the actual probability of the pattern arising from a random configuration. We will model the patterns as being generated by a random sequence of bits.

In order to use a random encoding of bits, we need to define the bit encoding for a Game of Life pattern. Section III-B contains a definition of an encoding, but it is based on functionality. The probability of a pattern arising is clearly not related to its functionality, and thus this measure is not a useful encoding for this purpose.

There are different ways to define this encoding. We can encode the width and height of the encoding using Levenstein encoding and each cell encoded as a single bit indicating whether it is living or not. This gives a total length of

$$\alpha(p) = l(p_w) + l(p_h) + p_w p_h$$

where p_w is the width of the pattern p and p_h is the height of the pattern. We will call this the standard encoding.

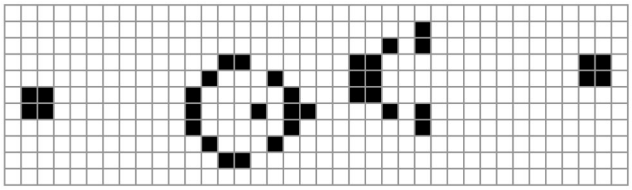


Fig. 4. Gosper gliding gun.

However, in many cases patterns consist of mostly dead cells. A lot of bits are spent indicating that a cell is empty. We can improve this situation by recording the number of live bits and then we can encode the actual pattern using less bits

$$\beta(p) = l(p_w) + l(p_h) + l(p_a) + \left\lceil \log_2 \binom{p_w p_h}{p_a} \right\rceil$$

where p_a is the number of alive cells. We will call this compressed encoding.

To demonstrate these methods, consider the Gosper gliding gun in Fig. 4. Using the standard encoding this requires

$$\begin{aligned} \alpha(p) &= l(p_w) + l(p_h) + p_w p_h \\ &= l(36) + l(9) + 36 \times 9 \\ &= 12 + 8 + 324 \\ &= 344 \text{ bits.} \end{aligned}$$

Using the compressed encoding requires

$$\begin{aligned} \beta(p) &= l(p_w) + l(p_h) + l(p_a) + \left\lceil \log_2 \binom{p_w p_h}{p_a} \right\rceil \\ &= l(36) + l(9) + l(36) + \left\lceil \log_2 \binom{324}{36} \right\rceil \\ &= 12 + 8 + 12 + 160 = 192 \text{ bits.} \end{aligned}$$

The compressed method will not always produce smaller descriptions as it does here. However, we can use both methods, and simply add an initial bit to specify which method was being used. Thus, the length of the encoding for a pattern, p is then

$$P(p) = 1 + \min(\alpha(p), \beta(p)) \tag{11}$$

where the 1 is to account for the extra bit used to determine which of the two methods was used for encoding.

However, we need to determine the Shannon information for a pattern, p. There are two ways to encode each pattern and both need to be considered

$$\Pr[X = p] = \Pr[X = p|C] \Pr[C] + \Pr[X = p|\overline{C}] \Pr[\overline{C}]$$

where X is the random variable of the chosen pattern, and C is the random event which is true when the compressed encoding is used. Since either method is chosen with 50% probability

$$\begin{aligned} \Pr[X = p] &= \frac{2^{-\alpha(p)}}{2} + \frac{2^{-\beta(p)}}{2} \\ &= \frac{2^{-\alpha(p)} + 2^{-\beta(p)}}{2}. \end{aligned}$$

Our primary purpose in this paper is to demonstrate OASC for functional machines in the Game of Life. However, the process also serves as a test of the hypothesis that the approximation to the probability of a pattern and its corresponding information in (11) arising is reasonably close. Are there features of random Game of Life patterns that tend to produce additional functionality? If so, we expect that we will obtain larger than expected values of ASC.

D. Computability

The contextual mathematical formulation thus far developed here for the Game of Life is less powerful than a Turing complete language. For example, there is no conditional looping mechanism. The Game of Life itself is Turing complete [48]; however, our equations using the components in Table II describing the Game of Life are not. There are concepts that cannot be described using the operations we have defined. A large array of a billion closely spaced albeit noninteracting blinkers has low KCS complexity akin to the celebrated low KCS complexity of a repetitive crystalline structure. A looping or a `REPEAT` command is required to capture low KCS complexity bound in such cases. The list in Table II of course, can be expanded to include these and other cases. However, the proof on the bound of ASC only requires that the language used to describe the pattern is prefix-free. Thus, the ASC bounds using the context in Table II still apply to the language defined here.

In order to use ASC, we must algorithmically derive the machine from the equations describing it. A program would systematically test all pattern in order of increasing size while checking whether they pass the test. We term this program the interpreter. Since the pattern specified whether it is the first, second, third, etc., pattern to pass the test, the process can stop and output the pattern once it is reached. Thus, a constant length interpreter program can derive the pattern from the equations, and ASC using a standard Turing machine is a constant longer than the OASC results presented here. If an alternate formulation is used to describe the pattern, then a different constant would apply as a different interpreter would be required.

The language used here is motivated in part for simplicity in understanding. It allows the comparison of the complexity of various specifications without constants which is difficult in standard KCS complexity.

Essentially, we have included the interpreter for our formulation as part of the context. The interpreter has details on the Game of Life, but not on the nature of patterns in it. This allows the description of the pattern in the Game of Life without any undue bias toward the patterns found in the Game of Life.

IV. Results

A. Oscillators

The simplest oscillator is one which does not actually change, i.e., a still life. An example is depicted in Fig. 1. This object can be described as

$$X = X\oplus, \#0 \tag{12}$$

TABLE III
ASC FOR THE SMALLEST KNOWN OSCILLATORS IN EACH CATEGORY

Name	Period	Complexity	$K(X\|C)$	OASC	Bound	$\Pr[X]$
block	1	12.68	38.0	-25.32	$4.189 * 10^{+07}$	$3.232 * 10^{-01}$
blinker	2	10.68	40.0	-29.32	$6.702 * 10^{+08}$	$3.292 * 10^{-01}$
caterer	3	61.68	41.0	20.68	$5.953 * 10^{-07}$	$7.692 * 10^{-11}$
mazing	4	60.83	42.0	18.83	$2.146 * 10^{-06}$	$4.545 * 10^{-09}$
pseudo-barberpole	5	95.0	43.0	52.0	$2.220 * 10^{-16}$	
unix	6	75.96	43.0	32.96	$1.197 * 10^{-10}$	$5.882 * 10^{-10}$
burloaferimeter	7	117.0	43.0	74.0	$5.294 * 10^{-23}$	
figure eight	8	50.91	44.0	6.91	$8.315 * 10^{-03}$	$3.030 * 10^{-08}$
29p9	9	113.96	45.0	68.96	$1.742 * 10^{-21}$	

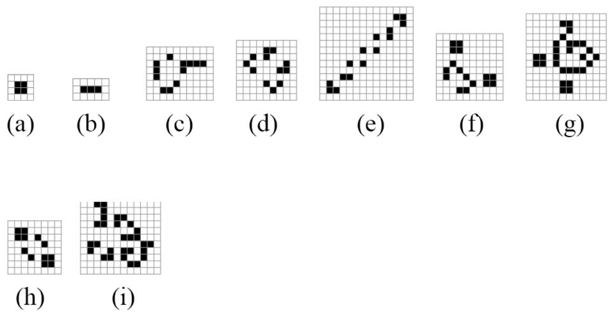


Fig. 5. Smallest known oscillators for each category. (a) Block. (b) Blinker. (c) Caterer. (d) Mazing. (e) Pseudo-barberpole. (f) Unix. (g) Burloaferimeter. (h) Figure eight. (i) 29p9.

as this is the smallest pattern that can fit the test. There are four symbols taking 20 bits to describe. There are five bits for the stop symbol and one bit to describe the sequence number. This gives a total of 26 bits to describe this pattern. Using standard encoding will require $l(2)+l(2)+2\times 2+1 = 4+4+4+4 = 13$. Thus, the ASC is at least $13-26 = -14$ bits of ASC. Since ASC is negative, the pattern is well explained by the stochastic process.

A flip-flop or period two oscillator as depicted in Fig. 2 can be described as

$$X = X\oplus^{i}, i = 2, \#0. \quad (13)$$

This takes six symbols (the repeat counts as a symbol) plus the stop symbol the parameter and the sequence number. That is a total of $35+l(2)+l(0) = 35+4+1 = 40$ bits. The blinker encoded using standard encoding will take $l(1)+l(3)+3+1 = 2+5+3+1 = 11$ bits. The OASC is then $11-40 = -29$ bits. Again, this pattern fits the modeled stochastic process well.

However, the same pattern could be described as

$$X = \ , \#0 \quad (14)$$

which has three symbols, and will require 11 bits for the pattern. The #0 is required, despite there being only one pattern which fits the equation, for consistency with the search process described in Section III-D. Thus, the length is $3\times 5+5+l(0)+11 = 20+1+11 = 32$ giving at least $11-32 = -21$ bits of ASC. In fact any pattern can be said to have at least -21 bits of ASC, because that is the overhead required to simply embed the pattern in its own description.

Simply by changing the value of i this same construct can describe an oscillator of any period. It will describe the smallest known oscillator of that period. Fig. 5[3] shows the smallest known oscillators for periods up to nine. Smaller oscillators than these may exist, but for now we believe these to be the ones described by the formulation. Table III shows the calculated values of OASC for the various oscillators. The $\Pr[X]$ column derives from experiments on random soups [50]. The missing entries do not appear to have been observed in random trials.

The $K(X|C)$ for the smallest known oscillator increases slowly as the period increases. The complexity generally increases, but not always. Caterer is the first oscillator with a positive amount of ASC. It does appear out of random configurations but at a rate much lower than the ASC bound. The ASC bound is violated in only one case, that of the unix oscillator. This oscillator shows up more often than our assumption regarding the probabilities would suggest. The pattern has a certain simplicity to it which is not captured by our metric.

Any pattern in the Game of Life can be constructed by colliding gliders [46]. The unix pattern can be constructed by the collision of six gliders. The psuedo-barberpole, the smallest known period five oscillator, requires 28 gliders. The burloaferimeter, the smallest known period seven oscillator, requires 27 gliders. The unix pattern requires fewer gliders to construct than either of the two most similar oscillators considered here. For its size, the unix pattern is easier to construct than might be expected.

B. Spaceships

A spaceship is a pattern in life which travels across the grid. It continually returns back to its original state but in a different position. The first discovered spaceship was the glider depicted in Fig. 3. We previously showed in (7) that it could be described as

$$X \downarrow\rightarrow = X\oplus^{4}, \#0.$$

This has eight symbols so the length will be $5\times 8+5+l(4)+l(0) = 45+6+1 = 52$. The complexity is 20 and the ASC is at least $20-52 = -32$ bits. As previously noted, any pattern can be described such that it has at least -21. This matches the observation that glider often arise from random configurations.

As with oscillators we can readily describe the smallest version of a spaceship. In addition to varying with respect to the

[3]David Buckingham found the strangely named *burloaferimeter* in 1972 [49].

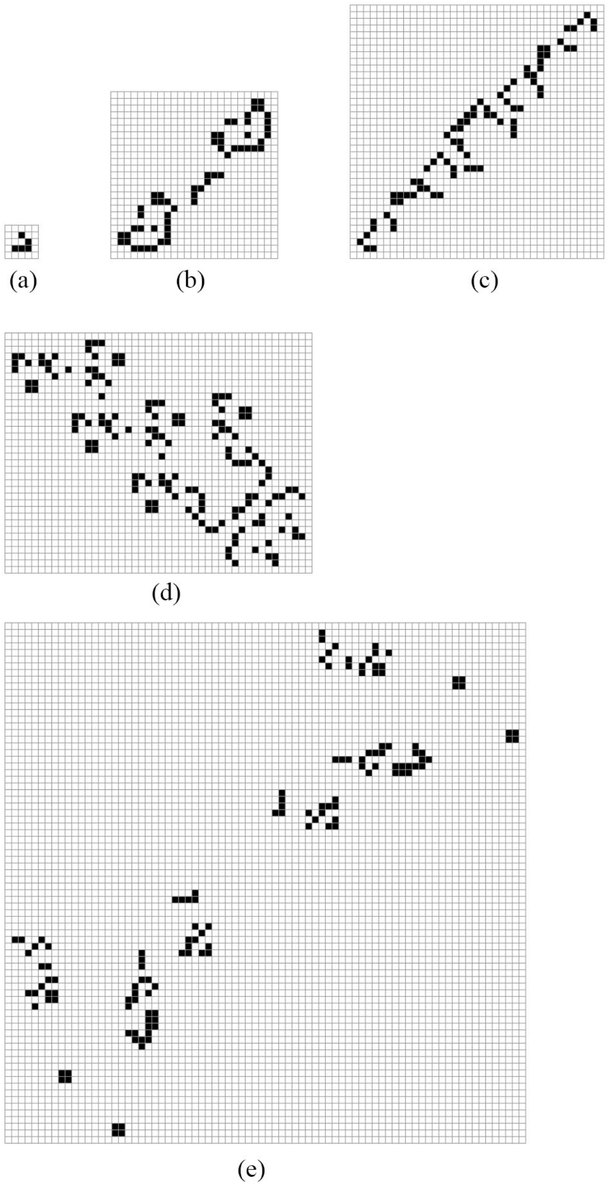


Fig. 6. Smallest known spaceships for each speed moving diagonally. (a) Glider. (b) 58P5H1V1. (c) 77P6H1V1. (d) 83P7H1V1. (e) Four engine cordership.

period, spaceships vary with respect to the speed and direction. Speeds are rendered as fractions of c, where c is one cell per iteration. First we will consider spaceships that travel diagonally like the glider. In general to travel with a speed of c/s with period p can be described as

$$X \downarrow^{\frac{p}{s}} \rightarrow^{\frac{p}{s}} = X \oplus^{p}, \#0. \tag{15}$$

This describes a spaceship moving down and the right. Due to the symmetry of the rules of the Game of Life, the same spaceships could all be reoriented to point in different directions. That would change the direction of the arrows, but not the length of the description. The length of this is $5 \times 12 + 5 + l(\frac{p}{s}) + l(p) + l(0) = 66 + l(\frac{p}{s}) + l(p)$.

Fig. 6 shows the smallest known diagonally moving spaceships for different speeds. If we assume that these are the smallest spaceships for these speeds, then (15) describes them.

TABLE IV
ASC for the Smallest Known Diagonal Spaceships for Each Speed

Name	Period	Speed	Complexity	$K(X\|C)$	OASC
glider	4	$\frac{c}{4}$	19.96	74.0	-54.04
58P5H1V1	5	$\frac{c}{5}$	296.0	75.0	221.0
77P6H1V1	6	$\frac{c}{6}$	459.0	75.0	384.0
83P7H1V1	7	$\frac{c}{7}$	733.0	75.0	658.0
Four engine cordership	96	$\frac{c}{12}$	962.0	89.0	873.0

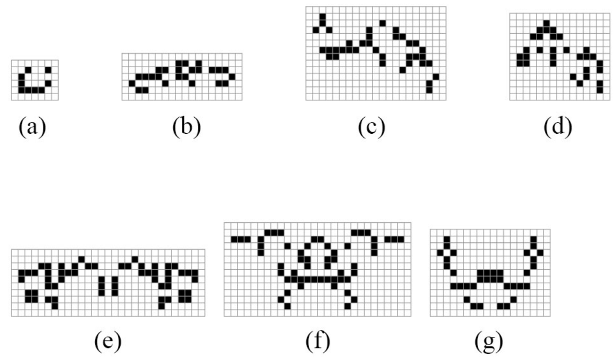


Fig. 7. Smallest known spaceships for each speed moving orthogonally. (a) Light weight space-ship. (b) 25P3HV1V0.2. (c) 37P4H1V0. (d) 30P5H2V0. (e) Spider. (f) 56P6H1V0. (g) Weekender.

TABLE V
ASC for the Smallest Known Orthogonal Spaceships for Each Speed

Name	Period	Speed	Complexity	$K(X\|C)$	OASC
lightweight spaceship	2	$\frac{c}{2}$	33.99	57.0	-23.01
25P3HV1V0.2	3	$\frac{c}{3}$	97.0	58.0	39.0
37P4H1V0	4	$\frac{c}{4}$	177.0	59.0	118.0
30P5H2V0	5	$\frac{2c}{5}$	133.0	62.0	71.0
Spider	5	$\frac{c}{5}$	211.0	60.0	151.0
56P6H1V0	6	$\frac{c}{6}$	242.0	60.0	182.0
Weekender	7	$\frac{2c}{7}$	158.0	62.0	96.0

Table IV shows the ASC for these various spaceships. The glider has negative ASC, it is simple enough to be readily explained by a random configuration. The remaining diagonal spaceships exhibit a large amount of ASC, fitting the fact that they are all complex designs. This is expected from look at Fig. 6 where the remaining patterns are much larger than the glider.

In addition to the diagonally moving spaceships we can also consider orthogonally moving spaceships. These move in only one direction, and so can be described as

$$X \uparrow^{\frac{p}{s}} = X \oplus^{p}, \#0. \tag{16}$$

The length of this is $5 \times 9 + 5 + l(\frac{p}{s}) + l(p) + l(0) = 51 + l(\frac{p}{s}) + l(p) + l(0)$. As with the diagonal spaceships, the same designs can be reoriented to move in any direction. The equation can be updated by simply changing the arrow. Fig. 7 shows the smallest known spaceship for a number of different speeds. Table V shows the ASC for the various spaceships. The simplest orthogonal spaceship, the lightweight spaceship, has negative bits of ASC. This matches the observation that these spaceships do arise out of random configurations [51]. The remaining spaceships exhibit significant amounts of ASC, although not as much as the diagonal spaceships, and are not reported to have been observed arising at random.

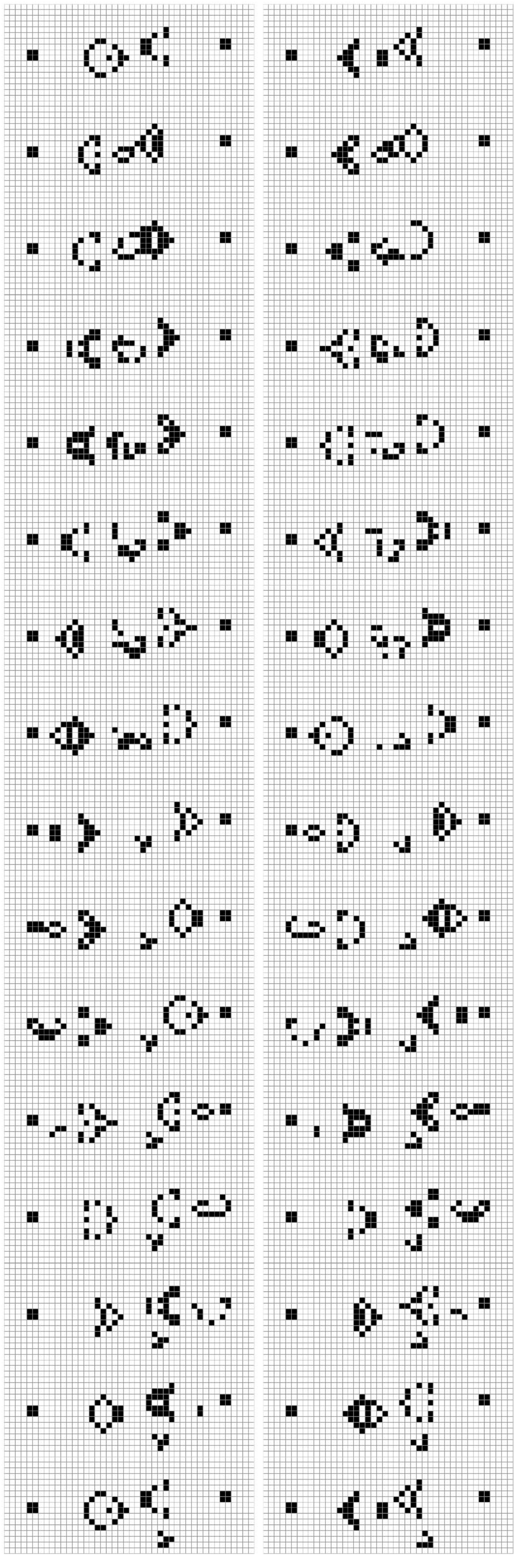


Fig. 8. Thirty-one iterations of the Gosper gun.

C. Guns

Fig. 8 shows the Gosper gun running through 31 iterations. The 30th iteration is the same as the original configuration except that it also includes a glider. The glider will escape and the gun will continue to produce gliders indefinitely. This is known as a gun. We can describe this gun as

$$X\oplus^{30} = X \cup \rightarrow^{24}\downarrow^{10}, \#0. \tag{17}$$

That is, the configuration after 30 iterations is equal to the original configuration with a glider added at a particular position. There are 60 bits for the symbols and it will require 20 bits to describe the glider, so $60 + 20 + l(30) + l(24) + l(10) + l(0)$ which is $60 + 20 + 11 + 11 + 8 + 1 = 111$ bits. The complexity is 196 bits. This gives us $196 - 111 = 85$ bits of OASC. At a probability of 2^{-85}, we conclude the Gosper gun is unlikely to be produced by a random configuration.

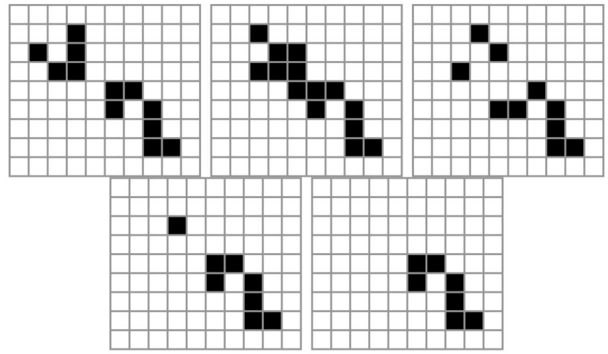


Fig. 9. Glider being eaten by the eater.

D. Eaters

Most of time when a glider hits a still life, the still life will react with the glider and end up being changed into some other pattern. However, with patterns known as eaters, such as that displayed in Fig. 9, the pattern "eats" the incoming glider resulting it returning to its original state. There are two aspects that make it an eater. Firstly, it must be a still life

$$X = X \oplus. \tag{18}$$

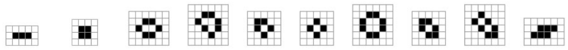
Secondly, it must recover from eating the glider

$$\left(X \cup \uparrow^{3}\leftarrow^{4}\right)\oplus^{4} = X. \tag{19}$$

The two equation have a total of 18 symbols, and the glider will require 20 bits to encode. Thus, the total length will be $5 \times 18 + 5 + 20 + l(3) + l(4) + l(4) + l(0) = 5 \times 18 + 5 + 20 + 4 + 7 + 7 + 1 = 134$ bits. The complexity of the eater is 29 bits. The OASC is thus $29 - 134 = -105$ bits. The eater is thus simple enough to be explain by a random configuration.

E. Ash Objects

Within the Game of Life, it is possible to create a random soup of cells and observe what types of objects arise from the soup. The resulting stable objects, still-lifes and oscillators, are known as *ash* [46]. Experiments have been performed to measure the frequencies of various objects arising from this soup [50]. Fig. 10 shows the ten most common ash objects, together comprising 99.6% of all ash objects. We observe that these objects are fairly small, and thus will not exhibit much complexity. The largest bounding box is 4×4 which will require at most $1 + l(4) + l(4) + 16 = 1 + 7 + 7 + 16 = 31$



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Fig. 10. Ten most common Game of Life ash objects. (a) Blinker. (b) Block. (c) Bee-hive. (d) Loaf. (e) Boat. (f) Tub. (g) Pond. (h) Ship. (i) Long-boat. (j) Toad.

bits. Describing the simplest still life required 26 bits, leaving at most 4 bits of ASC. Consequently, none of these exhibit a large amount of ASC.

V. Conclusion

We have demonstrated the ability to describe functional Game of Life pattern using a mathematical formulation. This allows us in principle to compress Game of Life patterns which exhibit some functionality. Thus, ASC has the ability to measure meaningful functionality.

We made a simplified assumption about the probabilities of various pattern arising. We have merely calculated the probability of generating the pattern through some simply random process not through the actual Game of Life process. We hypothesized that it was close enough to differentiate randomly achievable patterns from one that were deliberately created. This appears to work, with the exception of the unix pattern. However, even that pattern was less than an order of magnitude more probable than the bound suggested. This suggests the approximation was reasonable, but there is room for improvement.

We conclude that many of the machines built in the Game of Life do exhibit significant ASC. ASC was able to largely distinguish constructed patterns from those which were produced by random configurations. They do not appear to have been generated by a stochastic process approximated by the probability model we presented.

There are many more patterns in the Game of Life which have been invented or discovered. We have only investigated a sampling of the most basic patterns. Further investigation of specification in Game of Life pattern is certainly possible. Our work here demonstrates the applicability of ASC to the measure of functional meaning.

Appendix

Limiting the Number of Patterns for a Given Number of Live Cells

There are an infinite number of patterns for any given number of living cells. For example, two living cells could be separated by any amount of space. A systematic process working through all patterns would never get beyond two living cells because of the infinite number of such patterns. However, because a cell is only affected by its immediate neighbors, cells cannot affect the state of other cells which are sufficiently far away. We can thus ignore any pattern containing cells too far away to interact with one another. How far away is sufficient? We can inspect the equations we are testing against to see the number of $\oplus$ operations, after taking repetition into account. This gives us the number of iterations that could be checked, and thus the size of the observable universe for any given cell. We are not interested in any pattern where there is a gap larger than the size of the observable universe. Let $U = L + T + 1$ where L is the number of living cells in a pattern, and T is the number of $\oplus$ operations. Given a bounding-box larger than $U \times U$, there must exist a gap larger than the size of the observable universe. Consequently there is a finite number of interesting patterns for a given number of living cells, and we can number them.

Acknowledgment

The authors would like to thank the reviewers' suggestion of contrasting our paper with the study of temporal emergence properties of cellular automata.

References

[1] C. E. Shannon, W. Weaver, and N. Wiener, "The mathematical theory of communication," *Phys. Today*, vol. 3, no. 9, p. 31, 1950.

[2] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Hoboken, NJ, USA: Wiley-Interscience, 2006.

[3] A. N. Kolmogorov, "Logical basis for information theory and probability theory," *IEEE Trans. Inf. Theory*, vol. 14, no. 5, pp. 662–664, Sep. 1968.

[4] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Int. J. Comput. Math.*, vol. 2, nos. 1–4, pp. 157–168, 1968.

[5] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *J. ACM*, vol. 13, no. 4, pp. 547–569, 1966.

[6] G. J. Chaitin, "A theory of program size formally identical to information theory," *J. ACM*, vol. 22, no. 3, pp. 329–340, Jul. 1975.

[7] G. J. Chaitin, *The Unknowable*. New York, NY, USA: Springer, 1999.

[8] G. J. Chaitin, *Meta Math!: The Quest for Omega*. New York, NY, USA: Vintage, 2006.

[9] R. J. Solomonoff, "A preliminary report on a general theory of inductive inference," Zator Co. and Air Force Office Sci. Res., Cambridge, MA, USA, Tech. Rep. V-131, 1960.

[10] W. Ewert, W. A. Dembski, and R. J. Marks, II, "On the improbability of algorithmic specified complexity," in *Proc. IEEE 45th Southeastern Symp. Syst. Theory (SSST)*, Waco, TX, USA, Mar. 2013, pp. 68–70.

[11] W. Ewert, W. A. Dembski, and R. J. Marks, II, "Algorithmic specified complexity," in *Engineering and the Ultimate: An Interdisciplinary Investigation of Order and Design in Nature and Craft*, J. Bartlett, D. Halsmer, and M. Hall, Eds. Tulsa, OK, USA: Blyth Institute Press, 2014, pp. 131–149.

[12] W. A. Dembski, *The Design Inference: Eliminating Chance Through Small Probabilities* (Cambridge Studies in Probability, Induction, and Decision Theory). New York, NY, USA: Cambridge Univ. Press, 1998.

[13] K. K. Durston, D. K. Y. Chiu, D. L. Abel, and J. T. Trevors, "Measuring the functional sequence complexity of proteins," *Theor. Biol. Med. Model.*, vol. 4, p. 47, Jan. 2007.

[14] B. Cyganek, *Object Detection and Recognition in Digital Images: Theory and Practice*. Hoboken, NJ, USA: Wiley, 2013.

[15] H. V. Poor, *An Introduction to Signal Detection and Estimation*, 2nd ed. New York, NY, USA: Springer, 1994.

[16] J. Thomas, *An Introduction to Statistical Communication Theory*. Hoboken, NJ, USA: Wiley, 1969.

[17] J. Miller and J. Thomas,"Detectors for discrete-time signals in non-Gaussian noise," *IEEE Trans. Inf. Theory*, vol. 18, no. 2, pp. 241–250, Mar. 1972.

[18] R. J. Marks, II, G. Wise, D. Haldeman, and J. Whited, "Detection in Laplace noise," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-14, no. 6, pp. 866–872, Nov. 1978.

[19] M. Dadi and R. J. Marks, II, "Detector relative efficiencies in the presence of Laplace noise," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-23, no. 4, pp. 568–582, Jul. 1987.

[20] K. Cheung, L. Atlas, J. Ritcey, C. Green, and R. J. Marks, II, "Conventional and composite matched filters with error correction: A comparison," *Appl. Opt.*, vol. 26, no. 19, pp. 4235–4239, 1987.

[21] R. J. Marks, II and L. Atlas, "Composite matched filtering with error correction," *Opt. Lett.*, vol. 12, no. 2, pp. 135–137, 1987.

[22] R. J. Marks, II, J. Ritcey, L. Atlas, and K. Cheung, "Composite matched filter output partitioning," *Appl. Opt.*, vol. 26, no. 11, pp. 2274–2278, 1987.

[23] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*. New York, NY, USA: Springer, 2008.
[24] P. M. Vitányi, "Meaningful information," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4617–4626, Oct. 2006.
[25] M. Gardner, "Mathematical Games: The fantastic combinations of John Conway's new solitaire game life," *Sci. Amer.*, vol. 223, no. 4, pp. 120–123, 1970.
[26] A. P. Goucher. (2014, Jul. 10). "Oblique Life spaceship created," *Game of Life News* [Online]. Available: http://pentadecathlon.com/lifenews/2010/05/oblique_life_spaceship_created.html
[27] J. von Neumann, *Theory of Self Reproducing Automata*, A. W. Burks, Ed. Champaign, IL, USA: Univ. Illinois Press, 1966.
[28] J. von Neumann, "The general and logical theory of automata," in *Cerebral Mechanisms in Behavior the Hixon Symposium*, L. Jeffress, Ed. New York, NY, USA: Wiley, 1966.
[29] S. Wolfram, *A New Kind of Science*. Champaign, IL, USA: Wolfram Media, 2002.
[30] S. A. Kauffman, "Antichaos and adaptation," *Sci. Amer.*, vol. 265, no. 2, pp. 78–84, Sep. 1991.
[31] A. Wuensche and M. Lesser, *The Global Dynamics of Cellular Automata: An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata*. Reading, MA, USA: Wokingham, 1992.
[32] A. Wuensche, "The emergence of memory: Categorisation far from equilibrium," in *Towards a Science of Consciousness: The First Tucson Discussions and Debates*, S. Hameroff, A. Kaszniak, and A. Scott, Eds. Cambridge, MA, USA: MIT Press, 1996, pp. 383–392.
[33] A. Wuensche, "Complex and chaotic dynamics, basins of attraction, and memory in discrete networks," *Acta Phys. Pol.*, vol. 3, no. 2, pp. 463–478, 2010.
[34] E. Bonabeau and C. Meyer, "Swarm intelligence, a whole new way to think about business," *Harvard Bus. Rev.*, vol. 79, no. 5, pp. 106–114, May 2001.
[35] W. Ewert, R. J. Marks, B. B. Thompson, and A. Yu, "Evolutionary inversion of swarm emergence using disjunctive combs control," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 43, no. 5, pp. 1063–1076, Sep. 2013.
[36] J. Roach, W. Ewert, R. J. Marks, and B. B. Thompson, "Unexpected emergent behaviors from elementary swarms," in *Proc. IEEE 45th Southeastern Symp. Syst. Theory (SSST)*, Waco, TX, USA, Mar. 2013. pp. 41–50.
[37] W. A. Dembski, *No Free Lunch: Why Specified Complexity Cannot be Purchased Without Intelligence*. Lanham, MD, USA: Rowman & Littlefield, 2002.
[38] W. A. Dembski, "Specification: The pattern that signifies intelligence," *Philos. Christi*, vol. 7, no. 2, pp. 299–343, 2005.
[39] Open Source Shakespeare. (2014, Jul. 10). *Concordance of Shakespeare's Complete Works* [Online]. Available: www.opensourceshakespeare.org
[40] J. Bennett, W. Briggs, and M. Triola, *Statistical Reasoning for Everyday Life*, 2nd ed. Addison-Wesley, 2003.
[41] W. Ewert *et al.*, "Algorithmic specified complexity," in *Engineering and Metaphysics*. Tulsa, OK, USA: Blythe Inst. Press, 2012.
[42] (2014, Jul. 10). *Amazing Game of Life Demo* [Online]. Available: http://youtu.be/ XcuBvj0pw-E
[43] (2014, Jul. 10). *Epic Conway's Game of Life* [Online]. Available: http://youtu.be/ C2vgICfQawE
[44] (2014, Jul. 10). *Life In Life* [Online]. Available: http://youtu.be/xP5-iIeKXE8
[45] (2014, Jul. 10). *LifeWiki* [Online]. Available: http://www.conwaylife.com/wiki/Main_Page
[46] D. Salomon, *Variable-Length Codes for Data Compression*. London, U.K.: Springer, 2007.
[47] A. Adamatzky, Ed., *Collision Based Computing*. London, U.K.: Springer Verlag, 2002.
[48] (2014, Jul. 10). *Burloaferimeter*. [Online]. Available: http://www.conwaylife.com/wiki/Burloaferimeter
[49] A. Flammenkamp. (2004). *Top 100 of Game-of-Life Ash Objects* [Online]. Available: http://wwwhomes.uni-bielefeld.de/achim/freq_top_life.html
[50] (2014, Jul. 10). A. Flammenkamp. (1995). *Spontaneous appeared Spaceships out of Random Dust* [Online]. Available: http://www.homes.uni-bielefeld.de/achim/moving.html

Winston Ewert received the Ph.D. degree from Baylor University, Waco, TX, USA.

He has written a number of papers relating to search, information, and complexity. He was a Research Assistant at the Evolutionary Informatics Laboratory. He is currently a Software Engineer in Kirkland, WA, USA.

William Dembski (M'07–SM'07) received the Ph.D. degrees in mathematics with the University of Chicago, Chicago, IL, USA, in 1988, and in philosophy with the University of Illinois at Chicago, Chicago, IL, in 1996.

He is a mathematician and a philosopher. He has authored over 20 books, and is a Senior Fellow with Discovery Institute, Seattle, WA, USA, and a Senior Research Scientist with the Evolutionary Informatics Laboratory. His current research interests include information-theoretic underpinnings of the natural sciences. His newest book is *Being as Communion: A Metaphysics of Information* (Ashgate Pub. Co., August, 2014).

Robert J. Marks, II (S'71–M'72–SM'83–F'94) is a Distinguished Professor of electrical and computer engineering at Baylor University, Waco, TX, USA.

His most recent books are *Handbook of Fourier Analysis and Its Applications* (Oxford University Press, 2009), and *Biological Information—New Perspectives* (Singapore: World Scientific, 2013), coedited by M. J. Behe, W. A. Dembski, B. L. Gordon, and J. C. Sanford. He has an Erdős-Bacon number of five.