ABSTRACT

Algorithmic Specified Complexity

Winston Ewert, Ph.D.

Chairperson: Robert J. Marks II, Ph.D.

Information theory is a well developed field, but does not capture the essence of what information is. Shannon Information captures something in its definition of improbability as information. But not all improbable events convey information. Kolmogorov complexity captures the idea of information as something easily described. But not all easily described objects are information. The proposed *Algorithmic Specified Complexity* takes into account both Shannon Information and Kolmogorov complexity to gain a fuller evaluation of information. We demonstrate this concept and develop several examples. We show the low probability of high Algorithmic Specified Complexity. We apply the concept to both images and functional machines from the Game of Life.

Algorithmic Specified Complexity

by

Winston Ewert, B.Sc., M.S.

A Dissertation

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy

Approved by the Dissertation Committee

_____

Robert J. Marks II, Ph.D., Chairperson

_____

John M. Davis, Ph.D.

_____

Steven R. Eisenbarth, Ph.D.

_____

Kwang Y. Lee, Ph.D.

_____

Michael W. Thompson, Ph.D.

Accepted by the Graduate School
August 2013

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my parents; their patience, understanding, and love have made what I am today.

I would also like to think my Advisor, Robert J. Marks II, as well as his research partner, William A. Dembski for the opportunity to work with them while at Baylor.

CHAPTER ONE

Measuring Information

This chapter published as: Ewert, W., Dembski, W. A., & Marks II, R. J. (2012). Algorithmic Specified Complexity. Engineering and Metaphysics. Tulsa, OK.

1.1 Introduction

Intuitively, humans identify objects such as the carved faces at Mount Rushmore as qualitatively different from that of a random mountainside. However, quantifying this concept is an objective manner has proved difficult. Both mountainsides are made up of the same material components. They are both subject to the same physical forces, and will react the same to almost all physical tests. Yet, there does appear to be something quite different about Mount Rushmore. There is a special something about carved faces that separates it from the rock it is carved in.

This "special something" is information. Information is what distinguishes an empty hard disk from a full one. Information is the difference between random scribbling and carefully printed prose. Information is the difference between car parts strewn over a lawn, and a working truck.

While humans operate using an intuitive concept of information, attempts to develop a theory of information has thus far fallen short of the intuitive concept. Claude Shannon developed what its today known as Shannon information theory [1]. Shannon's concern was studying the problem of communication, that of sending information from one point to another. However, Shannon explicitly avoided the question of the meaningfulness of the information being transmitted, thus not quite capturing the concept of information as we are defining it. In fact, under Shannon's model random noise typically exhibits a large amount of information, the precise opposite of the intuitive concept.

Another model of information is that of algorithmic information theory [2, 3, 4]. Techniques such as Kolmogorov complexity measure the complexity of an object as the minimum length computer program required to recreate the object; Chaitin refers to such minimum length programs as *elegant* [5]. As with Shannon information, random noise is the most complex because it requires a long computer program to describe. In contrast, simple patterns are not complex because a short computer program can describe the pattern. But neither simple patterns or random noise are what we think of as information. As with Shannon information, there is a disconnect between Kolmogorov complexity and conceptual information.

Other models are based on algorithmic information theory, but also take in account the computational resources required for the programs being run. Levin complexity adds the log of the execution time to the complexity of the problem [6]. *Logical depth*, on the other hand, is concerned with the execution time of the shortest program [7]. There is a class of objects which are easy to describe but expensive to actually produce. It is argued [8] that objects in this class must have been produced over a long history. Such objects are interesting, but do not seem to capture all of what we consider to be the intuitive concept of information. English text or Mount Rushmore correspond to what we think of as information, but its not clear that they can be most efficiently described as long running programs.

One approach to information is the *specified complexity* as expressed by Dembski [9]. Dembski's concern is that of detecting design, the separation of that which can be explained by chance or necessity from that which is the product of intelligence. In order to infer design, and object must be both complex and specified. Complexity refers essentially to improbability. The probability of any given object depends on the chance hypothesis proposed to explain it. Improbability is a necessary but not sufficient condition for rejecting a chance hypothesis. Events which have a high probability under a given chance hypothesis do not give us reason to reject that hypothesis.

Specification is defined as conforming to an independently given pattern. The requirement for the pattern to be independent of the object being investigated is fundamental. Given absolute freedom of pattern selection, any object can be made specified by selecting that object as the pattern. It is not impressive to hit a bullseye if the bullseye is painted on after the arrow has hit the wall. It is impressive to hit the bullseye if the bullseye was painted before the arrow was fired.

Investigators are often not in the position of being able to choose the target prior to investigating the object. Consider the example of life. Life is a self-replicating process, and it would seem that an appropriate specification would be self-replication. Self-replication is what makes life such a fascinating area of investigation as compared to rocks. We know about self-replication *because of* our knowledge of life, not as an independent fact. Therefore it does not qualify as an independent specification. If we did not already have examples of self-replicating entities, we would not have picked as the specification.

The same is true of almost any specification in biology. It is tempting to consider flight a specification, but we would only be defining the pattern of flight because we have seen flying animals. As with life in general, specific features in biology cannot be specified independently of the objects themselves.

The concept of specification has been criticized for being imprecisely defined and unquantifiable. It is also charged that the maintaining the independence of the patterns is difficult. But specification has been defined in a mathematically rigorous manner in several different ways [9, 10, 11]. Kolmogorov complexity, or a similar concept, is a persistent method used in this definitions. Our goal is to present and defend a simple measure of specification that clearly alleviates these concerns. Towards this end, we propose to use *conditional Kolmogorov complexity* to quantify the degree of specification in an object. Combining this with the complexity, we can quantify the specified complexity as *algorithmic specified complexity*.

As noted, Kolmogorov complexity has been suggested as a method for measuring specification. The novelty in method presented in this paper is the use of conditional Kolmogorov complexity. However, this paper also elucidates a number of examples of algorithmic compressibility demonstrating wider applicability then is often realized.

1.2 Method

1.2.1 Kolmogorov

Kolmogorov complexity is a method of measuring information. It is defined as the minimum length computer program, in bits, required to produce a binary string.

$$K(X) = \min_{U(p,)=X|p\in P} |p| \tag{1.1}$$

where

- $K(X)$ is the Kolmogorov complexity of X
- P is the set of all possible computer programs
- $U(p,)$ is the output of program p run without input

The definition is given for producing binary strings.

Kolmogorov complexity measures the degree to which a given bitstring follows a pattern. The more a bitstring follows a pattern, the shorter the program required to reproduce it. In contrast, if a bitstring exhibits no patterns, it is simply random, and a much longer program will be required to produce it.

Consider the example of a random binary string, `100100000010100000001010`. It can be produced by the following Python program:

```
print '100100000010100000001010'
```

In contrast, we have the string, `000000000000000000000000`, which can be produced by

```
print '0' * 24
```

Both strings are of the same length, but the string following a pattern requires a shorter program to produce. Thus we have a technique for measuring the degree to which a binary string follows a pattern.

Specification is defined as following an independently given pattern. Kolmogorov complexity gives us the ability to precisely define and quantify the degree to which a binary string follows a pattern. Therefore, it seems plausible that we can measure specification using Kolmogorov complexity. The more compressible a bitstring, the more specified it is.

However, Kolmogorov complexity seems unable to capture the entirety of what is intended by specification. Natural language text is not reducible to a simple pattern; however, it is an example of what we'd consider specification. The design of an electronic circuit should also be specified, but it is not reducible to a simple pattern. In fact, the cases of specification that Kolmogorov complexity seems able to capture are limited to objects which exhibit some very simple pattern. But these are not the objects of most interest in terms of specification.

We use an extension of Kolmogorov complexity known as *conditional Kolmogorov complexity* [12]. The program now has access to additional data as its input.

$$K(X|Y) = \min_{U(p,Y)=X|p\in P} |p| \tag{1.2}$$

where $U(p, Y)$ is the output of running program p with input Y. The input provides additional data to the program. As a result, the program is no longer restricted to exploiting pattern in the desired output, but can take advantage of the information provided by the input. Henceforth, we will refer to this input as the *context*.

The use of context allows the measure to capture a broader range of specifications. It is possible to describe many bitstrings by combining a short program along with the contextual information. A useful range of specifications can be captured using this technique.

1.2.2 Algorithmic Specified Complexity

To combine the measurement of specification and complexity, we use the following formula for algorithmic specified complexity (ASC).

$$A(X, C, p) = -\log p(X) - K(X|C) \tag{1.3}$$

where

- X is the bitstring being investigated
- C is the context as a bitstring
- p is the probability distribution which we suppose X to have been selected from
- $p(X)$ is the probability of X occurring according to the chance hypothesis under consideration

Since high compressibility corresponds to specification, we subtract the compressed length of the string. Thus high improbability counts for specified complexity, but incompressible strings count against it.

For this number to become large requires X to be both complex, (i.e. improbable), and specified, (i.e. compressible). Failing on either of these counts will produce a low or negative value. Since Kolmogorov complexity can, at best, be upper bounded, the ASC can, at best, be lower bounded.

At best this measure can reject a given probability distribution. It makes no attempt to rule out chance based hypothesis in general. However, it can conclude that a given probability distribution does a poor job in explaining a particular item. The value of ASC gives a measure of the confidence we can have in rejecting a chance hypothesis.

1.2.3 Functionality

Perhaps the most interesting form of specification is that of functionality. It is clear that machines, biological structures, and buildings all have functionality. But

quantifying that in an objective manner has proven difficult. However, ASC gives us the ability to do this.

Any machine can be described, in part, by tests that it will pass. You can test the functionality of a car by seeing whether it accelerates when the gas or brake pedals are pushed. You can test the functionality of a cell by seeing whether it self-replicates. A test, or a number of tests, can be defined to identify the functionality of an object The existence of a test gives us the ability to compress the object. Consider the following pseuocode program:

```
counter = 0
for each possible building design
    if building won't fall over
        counter += 1
        if counter == X
             return building design
```

where X is some number. This program will output the design for a specific building. Different values of X will produce different buildings. But any building that will not fall over can be expressed by this program. It may take a considerable amount of space to encode this number. However, if few designs are stable, the number will take much less space than required to actually specify the building plans. Thus the stability of the building plan enables compression, which in turn indicates specification.

Kolmogorov complexity is not limited to exploiting what humans perceive as simple patterns. It can also capture other aspect such as functionality. Functionality can be described as passing a test. As a result functional objects are compressible.

1.3.1 Natural Language

As the first example, consider the sentence: "the quick brown fox jumps over the lazy dog." We can encode this sentence as UTF-32, a system for encoding that allows the encoding of symbols from almost any alphabet. Since each character takes 32 bits, the message will be encoded as a total of 1376 bits. In this case, we will take the context to be the English alphabet along with a space. This is a minimal level of information about the English language.

To specify one of the 27 characters, will require $\log_2 27$ bits. To specify the 43 character in the sentence will thus take $43 \log_2 27$ bits. We also need to record the number of characters at $2 \log_2 43 \approx 10.85$ bits. [1] Altogether, the bits required to specify the message requires $43 \log_2 27 + 2 \log_2 43 \approx 215.32$ bits.

However, in order to actually give a bound for Kolmogorov complexity, we must also include the length of the computer program which interprets the bits. Here is an example computer program in Python which could interpret the message

```
print ''.join(alphabet[index] for index in encoded_message)
```

This assumes that the alphabet and encoded message are readily available and in form amenable to processing within the language. It may be that the input has to be preprocessed, which would make the program longer. Additionally, the length of the program will vary heavily depending on which programming language is used. However, the distances between different computers and languages only differs by a constant [13]. As a result, it is common practice in algorithmic information theory, to discount any actual program length and merely include that length as a constant, c. Consequently, we can express the conditional Kolmogorov complexity as

$$K(X|C) \leq 215.32 \text{ bits} + c. \tag{1.4}$$

[1] A more compact representation for numbers is available. See the log^* method in Cover and Thomas [13].

The expression is less than rather than equal, because it is possible that an even more efficient way of expressing the sentence exists. But we know that at least this efficiency is possible.

The encoded version of the sentence requires 32 bits for each character, giving a total of 1376 bits. We adopt a simplistic probability model, supposing that each bit is generated by the equivalent of a coin flip. This means that the complexity, $-\log P(X)$ is 1376 bits. Using equation 1.3,

$$A(X, C, p) = -\log(p) - K(X|C) \geq 1376 \text{ bits} - 215.32 \text{ bits} - c = 1160.68 \text{ bits} - c. \tag{1.5}$$

This means that we have 1166 bits of algorithmic specified complexity by equation 1.3. Those 1166 bits are a measure of the confidence in rejecting the hypothesis that the sentence was generated by random coin flips. The large number of bits gives a good indication that is highly unlikely that this sentence was generated by randomly choosing bits.

However, we can also analyze the hypothesis that the sentence was generated by choosing random English letters. In this case we can calculate the probability of this sentence as

$$P(X) = \left(\frac{1}{27}\right)^{43}. \tag{1.6}$$

The complexity is then

$$-\log P(X) = -\log\left(\frac{1}{27}\right)^{43} = 43\log 27 \approx 204.46 \text{ bits}. \tag{1.7}$$

In which case the algorithmic specified complexity becomes

$$A(X, C, p) = -\log p(X) - K(X|C) \tag{1.8}$$

$$=\geq 204.46 \text{ bits} - 215.32 \text{ bits} - c \tag{1.9}$$

$$= -10.85 \text{ bits} - c. \tag{1.10}$$

The negative bound indicates that we have no reason to suppose that this sentence could not have been generated by a random choice of English letters. The bound is

negative as a result of two factors. In the specification, 10.85 bits were required to encode the length. On the other hand, the probability model assumes a length. Hence the negative bits indicate information which the probability model had, but was not provided in the context. Since the only provided context is that of English letters, this is not a surprising result. We did not identify any sort of pattern beyond that explained by the probability model.

We can also expand the context. Instead of providing the English alphabet as our context, we provide the word list of the Oxford English Dictionary [14]. In the second edition of that dictionary there were 615,100 word forms defined or illustrated. For the purpose of the alphabet context, we encode each letter as a number corresponding to that character. In this case, we can choose a number corresponding to words in the dictionary. We can thus encode this message:

$$K(X|C) \leq 9 \log_2 615,100 + 2 \log_2 9 + c \approx 179.41 + c. \tag{1.11}$$

Access to the context of the English dictionary allows much better compression than simply the English alphabet as comparing equations 1.4 and 1.11 shows.

Using equation 1.3, we determine

$$A(X, C, p) = -\log p(X) - K(X|C) \tag{1.12}$$

$$\geq 204.46 \text{ bits} - 179.41 \text{ bits} - c \tag{1.13}$$

$$= 25.05 \text{ bits} - c. \tag{1.14}$$

This gives us confidence to say this sentence was not generated by randomly choosing letters from the English alphabet.

It would be possible to adopt a probability model that selected random words from the English language. Such a probability model would explain all of the specification in the sentence. It would also be possible to include more information about the English language such that the specification would increase.

This technique depends on the fact that the numbers of words in the English language is much smaller then the number of possible combinations of letters. If the dictionary contained every possible combination of letters up to some finite length, it would not allow compression, and thus we would not be able to indicate specification. A language where all possible combinations of letters were valid words could still show specification, but another technique would have to be used to allow compression.

But one could also use a much smaller dictionary. A dictionary of 10 words would be sufficient to include all the words in this sentence. The ASC formula would give a much smaller compressed bound:

$$K(X|C) \leq 9 \log_2 10 + 2 \log_2 9 \approx 36.24 \text{ bits.} \tag{1.15}$$

This is a reduction of over 100 bits from equation 1.11. This is because it takes about 16 bits less to encode each word when the dictionary is this small. This is because the sentence is much more closely related to the context. It requires much less additional information to use the context.

But it is possible to include words not included in the dictionary. The program would have to fall back on spelling the word one letter at a time. Only bounds of the ASC can be computed. It is always possible a better compression exists; i.e. the object could be more specified than we realize.

1.3.2 Random Noise

While natural language is an example of something that should be specified, random noise is an example of something which should not. We will calculate the ASC of a random bitstring. This bitstring will contain 1000 bits, where each bit is assigned with equal probability 1 or 0. Since randomness is incompressible, calculating the Kolmogorov complexity is easy. The only way of reproducing a random bitstring is to describe the whole bitstring.

$$K(X) \leq 2 \log_2 1000 + 1000 + c \approx 1020 \text{ bits} + c \tag{1.16}$$

The probability of each bitstring is 2^{-1000} and thus the complexity will be 1000 bits. Calculating the ASC:

$$A(X, C, p) = -\log p(X) - K(X|C) \geq 1000 \text{ bits} - 1020 \text{ bits} - c = -20 \text{ bits} - c. \quad (1.17)$$

As expected, the ASC is negative, there is no evidence of pattern in the string which are not explained by the probability model.

However, we can also consider the case of a biased distribution. That is, 1 and 0 are not equally likely. Instead, a given bit will be one two thirds of the time, while zero only one third of the time. The entropy of each bit can be expressed as

$$H(X_i) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \approx 0.6365 \text{ bits}. \quad (1.18)$$

for any i The entropy of a bit is the number of bits required in an optimal encoding to encode each bit. This means we can describe the whole sequence as

$$K(X) \leq 2\log_2 1000 + 1000 * H(X_i) + c \approx 656.5 \text{ bits} + c. \quad (1.19)$$

If we adopt the uniform probability model, the complexity is still 1000 bits and

$$A(X, C, p) = -\log p(X) - K(X|C) \quad (1.20)$$

$$\geq 1000 \text{ bits} - 656.5 \text{ bits} - c \quad (1.21)$$

$$= 343.3 \text{ bits} - c. \quad (1.22)$$

This random sequence has a high bound of algorithmic specified complexity. It is important to remember that the ASC bound only serves to measure the plausibility of the random model. It does not exclude the existence of another more accurate model that explains the data. In this case, if we use the actual probability model used to generate the message

$$-\log_2(p) = H(X_i) * 1000 \approx 636.5 \text{ bits}. \quad (1.23)$$

and the resulting ASC:

$$A(X, C, p) = -\log p(X) - K(X|C) \tag{1.24}$$

$$\geq 636.5 \text{ bits} - 656.5 \text{ bits} - c \tag{1.25}$$

$$= -20 \text{ bits} - c. \tag{1.26}$$

The bound of ASC gives us reason to reject a uniform noise explanation for this data, but not the biased coin distribution.

Dembski [9] has considered the example of ballot rigging where a political party is almost always given the top billing on the ballot listing candidates. Since the selection is supposed to be chosen on the basis of a fair coin toss, this is suspicious. ASC can quantify this situation. We can describe the outcome by giving the numbers of heads and tails, follow by the same representation as for the biased coin distribution.

$$K(X) \leq 2\log X_h + 2\log X_t + \log \binom{X_t + X_h}{X_h} + c \tag{1.27}$$

where X_h is the number of heads, X_t is the number of tails We assume a probability model of a fair coin

$$-\log_2(p) = X_h + X_t \text{bits}. \tag{1.28}$$

This gives us:

$$A(X, C, p) = X_h + X_t - 2\log X_h - 2\log X_t - \log \binom{X_t + X_h}{X_h} - c \tag{1.29}$$

$$= X_h + X_t - \log \left(X_h^2 X_t^2 \binom{X_t + X_h}{X_h} \right) - c. \tag{1.30}$$

Figure 1.1 shows the result of plotting this equation for varying numbers of head and tails given 20 coin tosses. As expected, for either high numbers of tails or high number of heads, the bound of ASC is high. However, for an instance which looks like a random sequence, the ASC is minimized.

1.3.3 Playing Cards

Another interesting case is that of playing cards for poker. In playing cards if the distribution is not uniform; somebody is likely cheating. For the purpose of

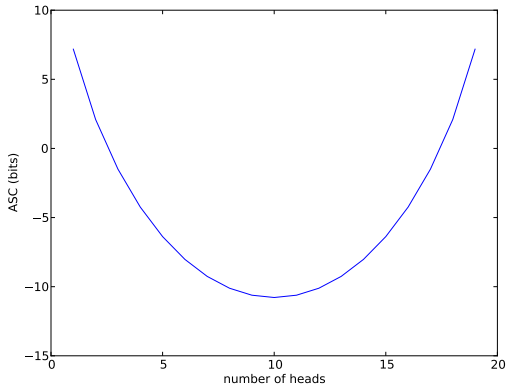


Figure 1.1: ASC for varyingly biased coin sequences and 20 coin tosses.

Table 1.1: Poker hand frequency.

Name	Frequency
Royal Flush	4
Straight Flush	36
Four of Kind	624
Full House	3744
Flush	5108
Straight	10200
Three of a Kind	54912
Two Pair	123552
One Pair	1098240
None	1302540

investigating card hands, we can simply assume a uniform random distribution over all five-card poker hands.

We will consider the hands for the game of poker. A poker hand is made up of 5 cards. Some categories of hands are rarer then others. Table 1.1 shows the frequency of the different hands.

Given a uniform distribution, every poker hand has the same probability, and thus the same complexity. There are 2,598,960 possible poker hands. This gives us

$$-\log_2 p(X) = -\log_2(\frac{1}{2,598,960}) \approx 21.3 \text{ bits.} \tag{1.31}$$

While the probability of every poker hand is the same, the Kolmogorov complexity is not. To describe a royal flush requires specifying that is a royal flush, and which suit it is in. However, describing a pair requires specifying which the paired value as well as both suit in addition to the three cards not involved in the pair. In general, describing hand requires specifying the type of hand, which of all the hands with that type. This gives us

$$K(H_i|C) \leq \log_2 10 + \log_2 |H| + c. \tag{1.32}$$

where 10 is the number of types of hands. H is the set of all hands of a particular type, and H_i is a particular hand in that set.

There are 1,098,240 possible pairs. Putting this in Equation 1.32 gives:

$$K(H_i|C) \leq \log_2 10 + \log_2 |H| + c \approx 23.39 \text{ bits} + c. \tag{1.33}$$

On the other hand, describing a pair without using the context gives

$$K(H_i|C) \leq \log_2 2,598,960 + c \approx 21.3 \text{ bits} + c. \tag{1.34}$$

Single pairs are so common that the space required to record that it was a pair is more then the space required to record the duplicate card straightforwardly. Accordingly, we must take the minimum of the two methods

$$K(H_i|C) \leq min(\log_2 10 + \log_2 |H|, \log_2 2,598,960) + c. \tag{1.35}$$

Table 1.2 shows the ASC for the various poker hands. Rare hands have large ASC, but command hands have low ASC. This parallels what we would expect, because a rare hand might cause us to expect cheating, but a common hand will not.

In other card games, a card is turned over after hands have been dealt to determine trump. The suit of the card is taken to trump for that round of the game. If the same suit is repeatedly chosen as trump, someone may ask what the odds are. This question can be difficult to answer because every possible sequence of trump suits

Table 1.2: The ASC of the various poker card hands.

Name	Frequency	Complexity	Compressed Length	ASC
Royal Flush	4	21.310	5.322	15.988
Straight Flush	36	21.310	8.492	12.818
Four of a Kind	624	21.310	12.607	8.702
Full House	3744	21.310	15.192	6.117
Flush	5108	21.310	15.640	5.669
Straight	10200	21.310	16.638	4.671
Three of Kind	54912	21.310	19.067	2.243
Two pair	123552	21.310	20.237	1.073
One pair	1098240	21.310	21.310	0.000
None	1302540	21.310	21.310	0.000

is equally likely. Yet, it is deemed unusual that the same suit is trump repeatedly. Algorithmic specified complexity allows us to capture this.

We represent the suits as a bit sequence, using two bits for each suit.

$$K(X) = \log_2 4 + \log_2 H + c = 2 + \log_2 H + c \tag{1.36}$$

where 4 is the number of suits, and H is the number of hands played. The complexity of the sequence is

$$-\log P(X) = 4^{\frac{-|X|}{2}} = 2H. \tag{1.37}$$

The ASC is then

$$ASC(X, p) = 2H - 2 - \log_2 H - c. \tag{1.38}$$

Note that this equation becomes $-c$ when $H = 1$. A pattern repeating once is no pattern at all, and doesn't provide specification. Figure 1.2 shows the ASC for increasing numbers of hands. The more times the same suit is chosen as trump, the larger the number of bits of ASC. Given the same trump for many rounds becomes less and less probable.

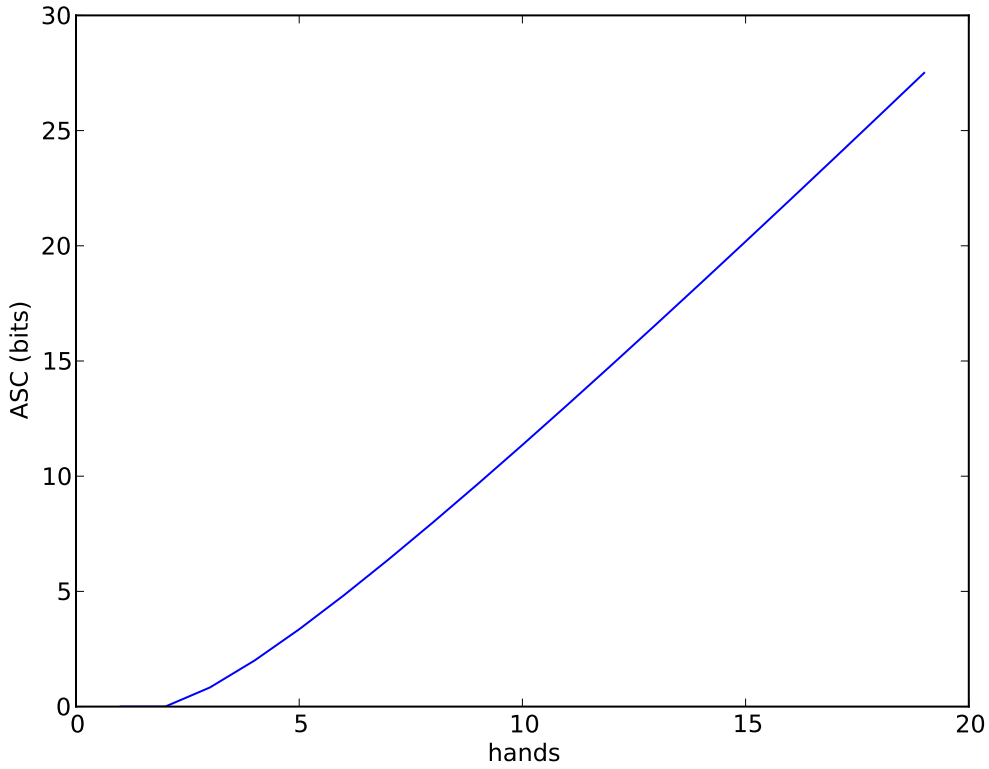


Figure 1.2: A plot of ASC for getting the same suit repeatedly.

1.3.4 Folding Proteins

In biology, an important prerequisite to a protein being functional is that it folds. The number of all possible proteins folding has been estimated.

> the overall prevalence of sequences performing a specific function by any domain-sized fold may be as low as 1 in 10^{77} [15]

We can create a program which outputs a particular protein, given the laws of physics.

```
for all proteins of length L
    run protein in a physics simulator
    if protein folds
        add to list of folding proteins
```

`output the Xth protein from the list`

This program given different choices of L and N will output any folding protein that we choose. This means that we can describe the protein by providing those two numbers.

$$K(X|C) = 2\log_2 L + \log_2 F_L + c \tag{1.39}$$

where C is the context, in this case the law of physics. F_L is the number of folding proteins of length L. Taking Axe's estimate [15], an assuming simplistically, that it applies for all lengths of proteins:

$$F_L = 10^{-77}4^L \tag{1.40}$$

$$\log F_L = -77\log 10 + L\log 4 \tag{1.41}$$

so

$$K(X|C) = 2\log_2 L + \log_2 F_L + c = 2\log_2 L + -77\log 10 + L\log 4. \tag{1.42}$$

For our probability model, we will suppose that each base along the DNA chain is uniformly chosen. It should be emphasized that according to the Darwinian model of evolution, the bases are not uniformly chosen. This supposition only serves to test a simplistic chance model of protein origin. We can calculate the probability as

$$-\log_2 \Pr(X) = -\log_2 4^{-L} = L\log_2 4. \tag{1.43}$$

Caution should be used in apply this formula. It assumes that proportion of functional proteins is applicable for all lengths, and implies that a fractional number of proteins fold.

Finally calculating the ASC,

$$ASC(X,p) = L\log 4 - 2\log_2 L + 77\log_2 10 - L\log_2 4 - c \tag{1.44}$$

$$= -2\log_2 L + 77\log_2 10 - c. \tag{1.45}$$

The final bound for ASC depends little on the length of the protein sequence which only comes to play in the logarithmic term. The significant term is the $77 \log_2 10 \approx 255.79$ bits. This means that we have good reason to believe that folding sequences were not generated randomly from a uniform distribution.

1.3.5 Functional Sequence Complexity

Kirk Durston *et. al.* have defined the idea of *functional sequence complexity* [16]. Functional sequence complexity is related to a special case of algorithmic specified complexity.

A protein is made from a sequence of amino acids. Some sequences have functionality and some do not. The case considered in section 1.3.4 above of folding is one particular case. But perhaps more interesting is considering the case of various proteins which perform useful biological functions.

Let Ω be the set of all proteins. Let F be the set of all proteins which pass a functionality test. Let $f(x)$ be a probability distribution over F. Both F and $f(x)$ can be produced by a simple algorithm using a functionality test on each element of Ω. Consequently, F and $f(x)$ can be described using a constant program length.

Consider the average for ASC over all elements in F.

$$\sum_{x \in F} f(x) A(x, C, p) = \sum_{x \in F} f(x)(-\log p(x) - K(x|C)) \tag{1.46}$$

$$= \sum_{x \in F} -f(x) \log p(x) - \sum_{xinF} f(x) K(x|C)) \tag{1.47}$$

We can describe any element x given the probability distribution and $\log f(x)$ bits. Given that we can calculate $f(x)$ and F with a constant program gives:

$$K(x|C) \leq \log -f(x) + c. \tag{1.48}$$

Place this into equation 1.47.

$$\sum_{x \in F} f(x) A(x, C, p) \geq \sum_{x \in F} -f(x) \log p(x) - \sum_{x \in F} -f(x) \log f(x) - \sum_{x \in F} c \tag{1.49}$$

The middle term is recognized as the Shannon entropy.

$$\sum_{x \in F} f(x) A(x, C, p) \geq \sum_{x \in F} -f(x) \log p(x) - H(f) - c \sum_{x \in F} f(x) \tag{1.50}$$

If we assume that the distribution p is uniform, $p(x) = \frac{1}{|\Omega|}$,

$$\sum_{x \in F} f(x) A(x, C, p) \geq \log_2 |\Omega| \sum_{x \in F} f(x) - H(f) - c \sum_{x \in F} f(x). \tag{1.51}$$

The two summations over F, are summations over a probability distribution and therefore 1.

$$\sum_{x \in F} f(x) A(x, C, p) \geq \log_2 |\Omega| - H(f) - c \tag{1.52}$$

Equation 5 in Durston's work, adjusting for notation is

$$\log |\Omega| - H(f). \tag{1.53}$$

This equation derives from making the same uniformity assumption that we have made here. Thus, for the uniform probability distribution case,

$$\sum_{x \in F} (f(x) A(x, C, p)) + c \geq \log |\Omega| - H(f). \tag{1.54}$$

This establishes the relationship between ASC and FSC. The difference is that the ASC is a lower bound, and includes a constant. This is the same constant as elsewhere: the length of the program required to describe the specification.

1.4 Objections

1.4.1 Natural Law

We have argued that compressibility in the presence of context is a necessary condition for information. This is in contrast to other who have argued that lack of compressibility is a necessary condition for information [17]. But compressible objects lack complexity. Because a compressible object is describable as some simple pattern, it is amenable to being produced by a simple process. Many objects in the real world follow simple patterns. Water tends to collect at lower elevations. Beaches

follow a slopping pattern. Sparks fly upwards. But these patterns are the result of the operation of simple law-like processes. Even if the explanations for these patterns were unknown, we would suppose due to the simplicity of the pattern shown that some simple explanation existed.

The premise behind this use of compressibility is that it identifies what human would see as simple patterns. Abel writes:

> A sequence is compressible because it contains redundant order and patterns. [17]

The problem is that algorithms are very versatile and allow the description of many patterns beyond that which humans would see as patterns. As has been shown by the various examples in this paper, many objects which do not exhibit what humans typically identify as redundant order and patterns are in fact compressible. Significantly, we have argued that functionality actually allows compressibility. Contrary to what Abel states, functional sequences are compressible by virtue of the functionality they exhibit. All of the sequences that Abel holds to be mostly incompressible are actually compressible.

But are compressible objects amenable to explanation by simple processes? Do all compressible objects lack complexity? If this were true, it would be problematic for algorithmic specified complexity because all specified objects would also be not complex, and no object would ever be both specified and complex. But many compressible objects do not appear to amenable to explanation by a simple process.

As discussed, English text is compressible given a knowledge of the English language. This does not somehow make it probable that English text will appear on a beach carved out by waves. Ninety degree angles are very compressible; yet, they are not typically found in nature. The existence of an explanation from the laws of nature does not appear to follow from compressibility.

Kolmogorov complexity deliberately ignores how long a program takes to run. It is only concerned with the length of the program's description. A program may be short but take an astronomical amount of time to run. Many of the specifications considered in this paper fall into that category. These objects are compressible, but that compression does not give an practical way to reproduce the object. But if there is no practical way to reproduce the object, we have no reason to suggest law-like processes as a plausible explanation.

1.4.2 Context is Subjective

The ASC of any object will depend on the context chosen. Any object can be made to have high ASC by using a specifically chosen context. But this appears to be the way that information works. If the authors, who do not understand Arabic, look at Arabic text, it appears to be no better then scribbling. The problem is not that Arabic lacks information content, but that we are unable to identify it without the necessary context. As a result, this subjectivity appears to capture something about the way information works in the human experience.

As with specification, it is important the context be chosen independent of object under investigation. While a specification will rarely be independent of the object under investigation, we believe it is much easier to maintain this independence in the case of a context.

1.4.3 Incalculability

It is not possible to calculate the Kolmogorov complexity of an object. However, it is possible to upper bound the Kolmogorov complexity and thus lower-bound the algorithmic specified complexity. This means that we can say that something is at least this specified, although we cannot rule out the possibility that it is even more specified. This means that we cannot mechanically detect that something has specification, although we can objectively identify it when we see it.

1.5 Conclusions

Dembski argued that information can be detected by looking for specified complexity. We propose that all or most forms of specification can be represented as algorithms, using Kolmogorov complexity. The shorter the algorithm, the more specified the object is. In order to measure a broader range of specification, we include the context and thus make use of conditional Kolmogorov complexity. We have defined the concept of Algorithmic Specified Complexity which takes into account the probabilistic complexity as well as the Kolmogorov complexity. We have presented a number of examples showing how this can represent the specification in a variety of cases. We hope that this paper introduces discussion on the use of conditional Kolmogorov complexity as a method for measuring specification as well as the use of Algorithmic Specified Complexity.

CHAPTER TWO

The Improbability of Algorithmic Specified Complexity

This chapter published as: Ewert, W., Dembski, W. A., & Marks II, R. J. (2013). On The Improbability of Algorithmic Specified Complexity. 2013 IEEE 45th Southeastern Symposium on System Theory: SSST 2013. Waco, TX.

2.1 Introduction

Low probability events are often claimed to not happen. But this is fallacious because low probability events take place all of the time. Any snowflake's pattern is highly improbable, but this does not prevent low probability snowflakes from existing. The common occurrence of low probability events seems paradoxical within the rubric of the probability paradigm. The paradox is resolved after recognizing there are often very many improbable events such that the total probability of such low probability events can actually be quite large.

To illustrate, assume that there exists 10^{1000} possible snowflakes each of which is equally likely. This means that any given snowflake pattern has a probability of 10^{-1000}. Thus

$$\Pr[\Pr[X] \leq 10^{-1000}] = 1. \tag{2.1}$$

where X is a random variable corresponding to a particular snowflake pattern. The equation states that we have a high probability (actually a certainty) of obtaining a very low probability event.

However, it is commonly assumed that no two snowflakes are alike. This is because such an event has dramatically lower probability than the occurrence of of a single specified snowflake. The probability of a second specified snowflake being identical to the first specified snowflake, however, is

$$\Pr[\text{First snowflake } = x, \text{ and the second snowflake } = x]$$

$$= 10^{-1000}10^{-1000} = 10^{-2000}.$$

But this is the same as the second snowflake having some other specification.

$$\Pr[\text{First snowflake } = x, \text{ and the second snowflake } = y]$$

$$= 10^{-2000}$$

where $x \neq y$. We revisit this example is Section 2.3.2.

Another example is the specified arrangement of sand on a beach. Any one particular arrangement of the sand is highly improbable. But an arrangement of the sand to spell words is not less probable then an arrangement without words. We would say, however, that the forming of such words through wind and water would be next to impossible.

How, then, do we resolve the seeming paradox that improbable events happen frequently? One resolution of the paradox is through the viewpoint of *specified complexity* [9]. An object with specified complexity is, as the name states, both specified and complex. For an object to be complex means that it is improbable. Specification means the object exhibits some independent pattern. The identical snowflakes exhibit a particular pattern: one snowflake is an exact replica of the other. This is what sets the pair off as distinct from all other pairs of snowflakes. Words written in the sand also exhibit a pattern: they form English letters.

Improbability gives us a good way to quantify the complexity of an object, but methods of measuring specification are less obvious. One method uses Kolmogorov complexity [2, 3, 4]. Kolmogorov complexity is defined to be shortest computer program length required to reproduce a specified bitstring description of an object. For identical snowflakes, the first snowflake can be described in detail followed by the computer command DUPLICATE. In contrast, describing two distinct snowflakes would require a longer program because each snowflake would have to be described separately. The program would be about twice as long as the program for the identical

snowflakes. The identical snowflakes require a shorter program to describe them because they follow more of a pattern. Short programs and hence smaller Kolmogorov complexity corresponds to objects which follow a pattern.

Kolmogorov complexity suffers from the property of being unknowable [18]. There is no method to compute the Kolmogorov complexity of an object with arbitrary length. However, we can give upper bounds for the Kolmogorov complexity. If a given bitstream of 1000 bits can be compressed without loss to 200 bits, we are assured that the Kolmogorov complexity of the 1000 bits on the operating system equals or exceeds 200 bits. Consequently we can show from this bound that there is a pattern to the input, but we cannot determine whether there is a pattern we are missing. Additionally, Kolmogorov complexity quantities contain an unknown additive constant that allows it to be applicable to any computer language. The constant can be thought of as the length in bits of one computer language translating into another. As a result of any modeling using Kolmogorov complexity, the quantity is a useful theoretical construct [13].

Using conditional Kolmogorov complexity [12] we define *algorithmic specified complexity* (ASC) [19] as

$$ASC(X, C, P) = -\log_2 P(X) - K(X|C)$$

where

- X is the object or event or under consideration
- C is the context– the presumed information which can be used to describe the object
- $K(X|C)$ is the Kolmogorov complexity of X given C. This quantity can not be computed exactly but can be bounded.
- $P(X)$ is the probability of the occurrence of X.

By taking into account both the probability and the Kolmogorov complexity of an object, the ASC measures the degree to which an event fits the presumed probability

distribution. The $\log_2 P(x)$ term measures the complexity of the object, whereas $-K(X|C)$ measures the specification. If an event happens which has a high ASC, we should conclude that since it has a low probability and the rare property of compressibility, it gives us strong indication to believe that the assumed probability distribution is incorrect.

The usefulness of this definition depends on the wide variety of constructs that are compressible. This includes for example simple pattern, such as "01" repeated 32 times. It also includes valid English text, which given a knowledge of the English language can be compressed. Its also include complex functioning systems because they can be described by their functionality rather then the system that produces that functionality. Thus Kolmogorov complexity captures a wide variety of objects that we deem "special." Thus we can usefully apply this metric to a wide variety of objects.

2.2 A Bound on the Probability of ASC

The following theorem quantifies the unlikelihood of obtaining a high ASC event.

Theorem 1. *The probability of obtaining an object exhibiting α bits of ASC is less then or equal to $2^{-\alpha}$.*

$$\Pr[ASC(X, C, P) \geq \alpha] \leq 2^{-\alpha} \tag{2.2}$$

Proof.

$$\Pr[ASC(X, C, P) \geq \alpha]$$

$$= \Pr[-\log_2 P(X) - K(X|C) \geq \alpha]$$

$$= \Pr[P(X) \leq 2^{-\alpha - K(X|C)}]$$

Let β be the set of all events in the domain of X such that $P(X) \leq 2^{-\alpha-K(X|C)}$.

$$\Pr[ASC(X, C, P) \geq \alpha] = \sum_{x \in \beta} P(x).$$

The definition of β is such that we have an upper bound on $P(x)$. Thus

$$\begin{aligned} \Pr[ASC(X,C,P) \geq \alpha] &\leq \sum_{x\in\beta} 2^{-\alpha-K(x|C)} \\ &= 2^{-\alpha}\sum_{x\in\beta} 2^{-K(x|C)}. \end{aligned}$$

Since Kolmogorov complexity can assume prefix free code [13], a distribution over all programs is defined by

$$\Pr[X = x] = 2^{-K(x|C)}.$$

$\sum_{x\in\beta} 2^{-K(x|C)}$ is a summation over this distribution for some subset of the values, thus it less then or equal to one as dictated by the Kraft inequality [13].

$$\Pr[ASC(X,C,P) \geq \alpha] \leq 2^{-\alpha} \tag{2.3}$$

This proves the theorem. □

From the main result of Theorem 1 in (2.2),

$$-\log_2 \Pr[ASC(X,C,P) \geq \alpha] \geq \alpha$$

It is therefore unlikely to obtain a high value of ASC. Low probability events commonly occur, but high ASC events do not.

2.3 Examples

The definition of ASC uses both complexity and specification. We can look at various cases of these parameters to see how ASC is affected.

2.3.1 Uniform Specification and Complexity

2.3.1.1 *Compressible Sequences* Suppose that we have 256 items each with equal probability of occurring and each of the same compressed length. The only way

for 256 items to all have the same minimum length is to use 8 bit codes for all of them. For any item in the collection we then have

$$\begin{aligned} ASC(X, C, P) &= -\log_2 P(X) - K(X|C) \\ &= -\log_2 \frac{1}{256} - 8 \\ &= 8 - 8 \\ &= 0 \text{ bits of ASC} \end{aligned}$$

And the bound on 0 bits of ASC is

$$\Pr[ASC(X, C, P) > 0] \leq 2^{-0} = 1$$

The probability in this case is clearly 1 because all objects will have the same ASC.

2.3.1.2 *A Rare Compressible Sequence* Suppose we have a single sequence that can be compressed into 2 bits but has a probability of 2^{-256}. Then we calculate the ASC

$$\begin{aligned} ASC(X, C, P) &= -\log_2 P(X) - K(X|C) \\ &= -\log_2 2^{-256} - 2 \\ &= 256 - 2 \\ &= 254 \text{ bits of ASC.} \end{aligned}$$

The bound on this is

$$\Pr[ASC(X, C, P) > 254] \leq 2^{-254}$$

which is 4 times as probable as the actual value because there can be only up to 4 bit sequences of 2 bits in length.

2.3.1.3 *A Common Compressible Sequence* Suppose that we have a single sequence that can be compressed into 2 bits and this happens half of the time. We

calculate the ASC

$$\begin{aligned} ASC(X, C, P) &= -\log_2 P(X) - K(X|C) \\ &= -\log_2 \frac{1}{2} - 1 \\ &= 1 - 2 \\ &= -1 \text{ bits of ASC.} \end{aligned}$$

This gives the bound

$$\Pr[ASC(X, C, P) > -1] \leq 2^1 = 2.$$

While the sequence is highly compressible, the high probability prevents it from having a large measure of ASC.

2.3.2 Snowflakes

Consider again the case of the snowflakes. There are, by our assumption, 10^{1000} possible snowflakes. We'll assume that we can describe each snowflake using a compact bit pattern taking $\log_2 10^{1000} = 1000 \log_2 10 \approx 3322$ bits. If we have to describe two distinct snowflakes, it will take

$$\begin{aligned} K(X|C) &= 3322 \text{ bits} + 3322 \text{ bits} + c \\ &= 6644 \text{ bits} + c \end{aligned} \tag{2.4}$$

where c is some constant number of bits. The log-probability is

$$\begin{aligned} -\log_2 P(X) &= -\log_2 10^{-2000} = 2000 \log_2 10 \\ &= 6644 \text{ bits.} \end{aligned} \tag{2.5}$$

So the ASC is

$$\begin{aligned} -\log_2 P(X) - K(X|C) &= 6644 \text{ bits} - 6644 \text{ bits} - c \\ &= -c. \end{aligned} \tag{2.6}$$

Using Theorem 1 we obtain a bound of 2^c which, since $c > 0$, produces a bound above 1 for the probability of obtaining this pair of snowflakes. There is nothing unusual about an arbitrary pair of snowflakes.

However, if the two snowflakes are identical, we can describe them by a shorter program.

$$K(X|C) = 3322 \text{ bits} + c.$$

where c is some constant number of bits. The probability is the same, so the ASC is

$$\begin{aligned} -\log P(X) - K(X|C) &= 6644 \text{ bits} - 3322 \text{ bits} - c \\ &= 3322 \text{ bits} - c. \end{aligned}$$

Using the theorem, we bound the probability at $2^{-3322+c}$. Assuming that the constant c is sufficiently small, this is a vanishingly small probability and thus we can conclude that obtaining two identical snowflakes would be absurdly improbable. If we did find two identical snowflakes, we would have to conclude that our original assumed probability distribution was incorrect.

2.4 Conclusion

The algorithmic specified complexity is a theoretical quantification measuring how well a probability distribution explains a given event. By using the bound in Theorem 1, we can establish the probability of obtaining particular amounts of ASC. We conclude that an object exhibiting high ASC is unlikely to arise. Given a high ASC object, we have evidence that the assumed probability distribution was incorrect.

Additional examples of ASC are available [19]. We are currently exploring the capabilities and limitations of the ASC measure.

CHAPTER THREE

Visual Algorithmic Specified Complexity

3.1 Introduction

It is critical to be able to distinguish patterns and information from noise. As humans we readily do this. But what is our theoretical basis for doing so? If we look at a picture of a sunset, we readily identify it as not being a random assortment of pixels, but why? A random image which looked like a sunset would be astronomically improbable. However, so would any random image. Thus probability alone does not seperate the pattern from the noise.

In order for an image to be distinguishable from random noise, it must follow some independant pattern or specification. If an object follows a specification or a pattern we say that is specified. The image of the sunset follows a pattern by virtue of the fact that it looks like other sunsets. Any image containing content rather than random noise fits some pattern. Naturally, any image looks like itself, but the requirment is that the pattern must be independent, and therefore the image cannot form a pattern for itself. If an object is both improbable and specified we say that it exhibits *s*pecified complexity[9].

The question that remains is how to measures this specification. Measuring complexity is straightforward as it is simply the probability of the image. Specification can be quantified using *K*olmogorov Complexity [2, 3, 4]. Kolmogorov complexity or variations thereof have been previously proposed as a measurement method for specification[9, 10, 11].

Kolmogorov complexity is defined as the length of the shortest program required to reproduce a result, in this case the pixels in an image. The more the image can be described in terms of a pattern, the more compressible it is, and the more specified.

For example, a black square is entirely described by a simple pattern, and a very short computer program sufficies to recreate it. As a result we conclude that it is very specified. In contrast, a completely random image cannot be compressed at all, and thus we conclude that it is not specified at all. Images with content such as sunsets take more space to describe than the black square but are more specified than random noise.

However, it is only uniform random noise which defies compression. Stochastic processes which follow some other distribution will be compressible. For example, the black square seems very simple, i.e. not complex. It would seem problematic to classify such a simple image with the images of sunsets or other content. To account for this, we have to model a stochastic process which can produce such simple images. Which images might be considered simple depends on the stochastic process being modelled.

Given a particular stochastic process, we would like to be able to measure how well a given image is explained by that process. The goal is seperate those images which look like they were produced by the stochastic process from those which were not. Towards this end we use Algorithmic Specified Complexity, [19]

$$ASC(X, C, P) = -\log_2 P(X) - K(X|C) \tag{3.1}$$

where

- X is the object or event or under consideration
- C is the context, given information which can be used to describe the object
- $P(X)$ is the probability of X under the given stochastic process.
- $K(X|C)$ is the Kolmogorov complexity of X given context C.

By taking into account the Kolmogorov complexity and the probability assigned by the stochastic process the ASC measures the degree to which image fits the hypothesized stochastic process. Given high ASC, we have reason to believe that the image is

unlikely to be produced by that process. In fact, we can conclude that [20]

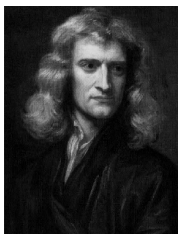
$$\Pr[ASC(X, C, P) \geq \alpha] \leq 2^{-\alpha}, \tag{3.2}$$

thus bounding the probability of obtaining high ASC images when sampled according to a given distribution. For example, we have a one in 1024 chance of obtaining 10 bits of ASC. Large amounts of ASC give strong indication that the image was not produced by the proposed stochastic process.

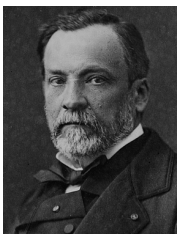
ASC is defined based on the conditional kolmogrov complexity, taking the context as a parameter. The context enables the compression to take advantage of known information. A picture of a house defies explanation by a simple stochastic process because it looks like other houses which have previously been seen. If we take the context to be a library of known images, then the similarity should allow us to describe the new image by making use of details from the library images. Without the context, images with simple patterns like shapes or fractals could be deemed compressible, but it is difficult to see that an image of a house would be compressible. Including a context lets us take into account prior experience and area of knowledge.

A solid black square may be assigned a high probability by a reasonable stochastic process. It is very compressible and thus specified, but does not have a level of ASC due to its low complexity. A random image will be assigned a low probability by a stochastic process, but it is not compressible and therefore not specified. As a result, it will not have a high value of ASC either. A sunset will be given a low probability by a stochastic process (excluding those designed to produce images of sunsets), and it is also specified, because it can be described by a shorter computer program. Consequently, the ASC of the sunset image will be high. The ASC allows us to distinguish between these various categories of images.

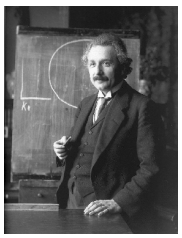
This paper's contribution is to consider the application of ASC to images. By using a library of images in a number of scenarios we demonstrate ASC's ability to distinguish images from noise. We will show that it can work under noise, algorith-



(a) Newton



(b) Pastuer



(c) Einstein

Figure 3.1: Images of scientists.

mic transformations, and different camera shots. The paper will investigate these different examples and show ASC in each one. Thus we demonstrate, by example, the applicability of ASC.

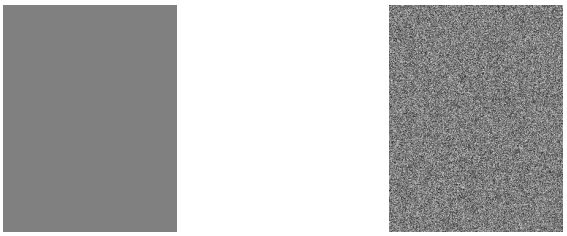
3.2 Prior Work

The Kolmogorov Complexity of images has been used as a method of computing image similarity, [21, 22]. These are based on the notion of information distance, [8], which computes the similarity of two binary sequences (or anything mapped to binary sequences) using conditional Kolmogorov complexity. The idea is that if two images are similiar, there should be a set of algorithmic transformations to convert one image into the other such that it requires less space to describe the transformations than to simply encode the image directly. Others have worked on the problem of compressing similiar images, [23, 24]. The idea is that we should be able to take advantage of image similarities to compress them better. The compressibility of similiar images is the basis of the work considered here. Without it, it would be impossible to use the library of images to compress related images.

3.3 Results

3.3.1 Image Library

Figure 3.1 shows three pictures of famous scientists which makes up the library of images for our context. For contrast, see Figure 3.2 which shows a solid square



(a) Solid Gray Square

(b) Random Image

Figure 3.2: Comparison images not included in the library.

and a random image. These are not in the library, but are provided for comparison. The square is very compressible because of its single solid color, whereas the random image is not due to the incompressible noise that it contains.

In the simplest case, we want to compress an image exactly identical to one in the library. We can easily describe such an image merely by its index in the library and thus:

$$K(X|C) = \lceil \log 3 \rceil + c \tag{3.3}$$

The images are 284 x 373 pixels in grayscale, with 256 levels of gray. The raw grayscale image encoded directly would require $8 * 284 * 373 = 847,456$ bits. Initially, we will postulate the images were generate by randomnly choosing the grayscale for each pixel uniformly across all possible values. This would mean that every possible grayscale image has an equal probability.

$$\Pr[X] = 2^{-847456} \tag{3.4}$$

where X is the random variable constiting the image. The complexity of the image is then:

$$-\log_2 \Pr[X] = -\log_2 2^{-847456} = 847456 \text{ bits.} \tag{3.5}$$

Using the formula for ASC, we obtain for any of the library images

$$\begin{aligned} ASC(X, C, P) &= -\log_2 \Pr[X] - K(X|C) \\ &= 847456 - 2 - c \\ &= 847454 - c \text{ bits.} \end{aligned}$$

Recall that $\Pr[ASC > 847454] \leq 2^{-847454}$ which renders the probability of generating these images through such a stochastic process as absurdly improbable.

How does the process fair for a simple pattern such as the solid square? The solid square can be described by its particular shade of gray, taking 8 bits. Thus the complete description of the solid square is:

$$K(X|C) = 8 + c \tag{3.6}$$

Thus the ASC for the solid square of the same size as the scientists' pictures would be $847,456 - 8 + c = 847,448 - c$ bits. The square is only slightly less likely to be produced by the stochastic process then the detailed images of the scientists. This is because a stochastic process picking uniformly over all random images of this size is extermely unlikely to produce a solid image. The stochastic process we are using does not assign higher probability to simple patterns.

However, we can define a stochastic process which is more likely to do so. We will adopt an approximation of complexity based on length of *Portable Network Graphic (PNG)*[25] files. The PNG format is designed to take advantage of redundancies present in typical images to produce better compression. Thus the modelled stochastic process will produce images containing these sorts of redundancies, and such redundancies will not be a basis for high ASC. The first 8 bytes of a PNG image are always the same, so we've excluded these from the length calculation. We assume that the probability of an image is thus

$$\Pr[X] = 2^{-l(X)-8} \tag{3.7}$$

Table 3.1: Details on the various images.

Image	Complexity (Uniform)	Complexity (ASC)	KC	ASC (Uniform)	ASC (ASC)
Newton	847456	520224	2	847454	520222
Pasteur	847456	543000	2	847454	542998
Einstein	847456	513064	2	847454	513062
Square	847456	6224	8	847448	6216
Random	847456	848808	847456	0	1352

where $l(X)$ is the length in bits of the PNG file required to produce the image. Naturally, this gives a complexity of $l(x) - 8$.

Table 3.1 shows the complexity and ASC for the various images under the two different stochastic models. The pictures of the scientists all compress to similiar lengths in PNG and are thus deemed similiarily complex. The random image is significantly more complex, while the solid square is much less complex. Using the PNG complexity, the square image has two orders of magnitude smaller less ASC than the other images. The square image is much better explained than any of the library images. It still has a large amount of ASC, this is because it still takes many bits to describe the image using PNG. It is still exceedingly unlikely to create solid image by randomnly generating PNG files.

A somewhat surprising result is the quantity of ASC found in the random image under the PNG complexity. As might be expected, under a uniform distribution, the complexity and specification cancel each other out leaving absolutely no indication of specified complexity. However, the PNG-based stochastic model would assign lower probabilities to images lacking any sort of redundancy. The absence of redundancy means that the image does not fit the stochastic process. This is exactly what should be the case as it was not produced by that stochastic process.

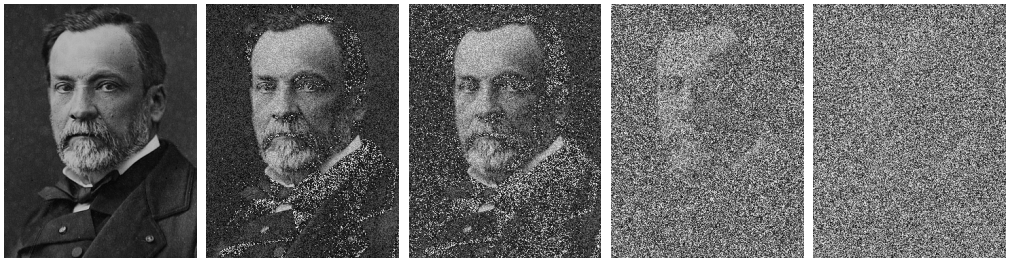


Figure 3.3: Picture of Louis Pastuer with increasing levels of added noise.

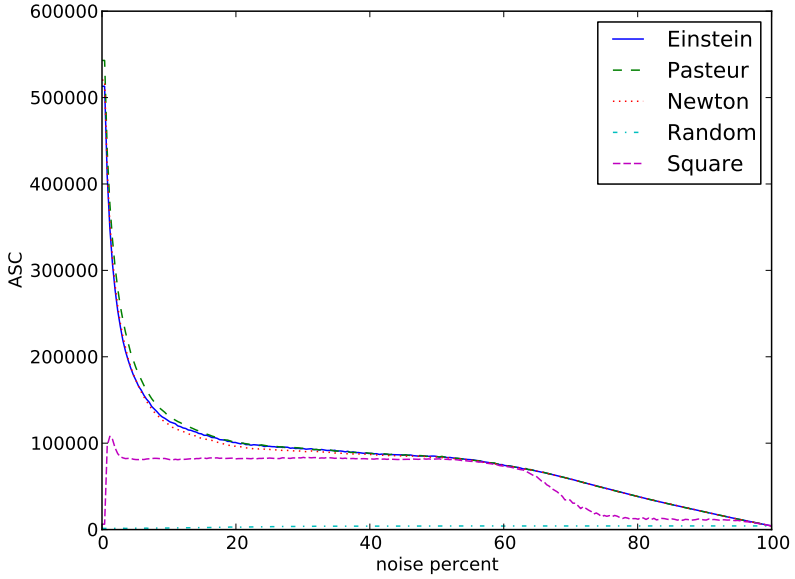


Figure 3.4: ASC for varying levels of noise.

3.3.2 Noise

Not all images will be exactly identical to those in the library. For a simple case consider a noisy copy of an image. It is the same as the library version, except that noise has been added to it. In order to compress that image, we need to specify both the image in the library as well as the noise.

$$K(X|C) = \lceil \log_2 3 \rceil + pH(N) + c \tag{3.8}$$

where p is the number of pixels, N is the random variable modelling the noise, and $H(N)$ is the entropy of that random variable. The entropy of a random variable is

the average numbers of bits required to describe that result of that random variable. Note that only the entropy of the random variable affects the description length. The mean of the variable could be shifted without forcing the image to use any additional space. The square image cannot be described as similiar to one in the library, but it can be described as its base color with the noise

$$K(X|C) = 8 + pH(N) + c \tag{3.9}$$

Adding noise to a random image producing another random image, leaving us with no way of compressing it:

$$K(X|C) = 8p + c \tag{3.10}$$

We modelled uniformly random noise added to each pixel. Figure 3.3 shows the picture of Pastuer as increasing levels of noise are added. Figure 3.4 shows the plot of the varying images as levels of noise are increased. At 0% noise, the image is exactly identical to the one in the library. At 100% noise, the image is indistinguishable from random noise. The three scientit's images follow each other closely. There is initially a great deal of ASC, but this decreases as the noise is increased. The square actually has an initial increase in ASC as noise is added. This is because the PNG file format works very well to compress a solid square, but does a relatively poor job of compressing that square with just a small amount of noise.

There is a relatively flat period between twenty and sixty percent. This is caused by a closely matched increase in the png length of the images and the kolmgorov complexity of those images. The noise increases both the complexity of the image, as well as decreasing the specification. These two changes cancel out leaving a slow change. All of the methods tend towards zero ASC as the noise reaches 100%. The random image is flat always exhibiting very low amounts of ASC.

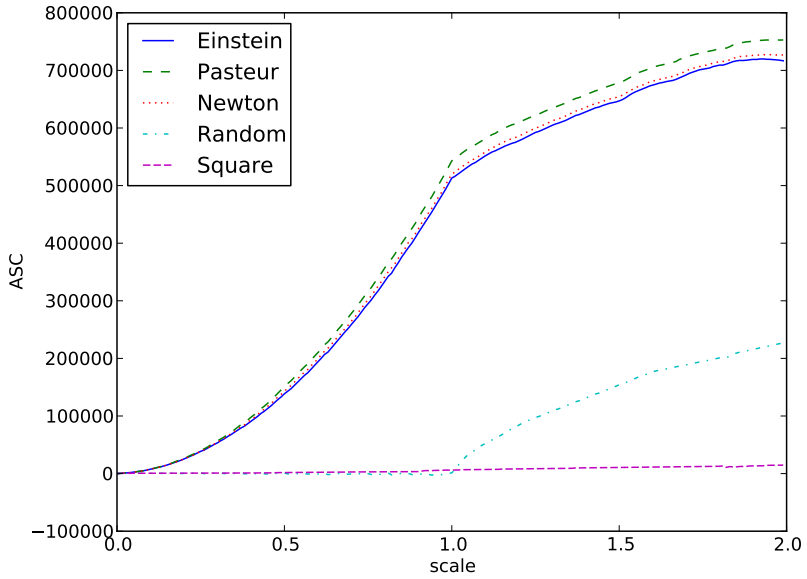


Figure 3.5: ASC for different resizings.

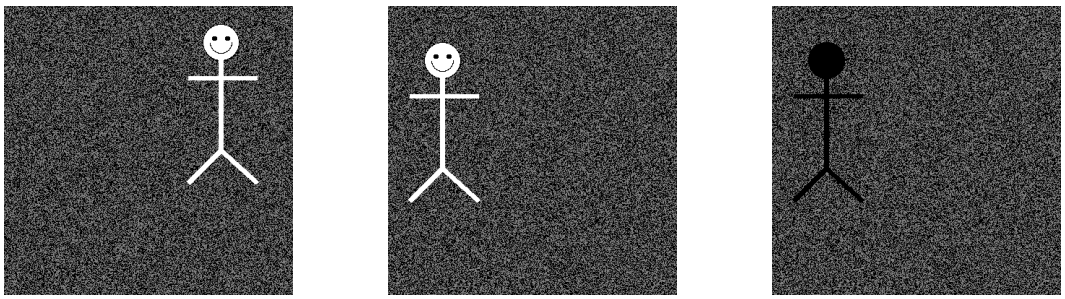
3.3.3 Scaling

Another possibility is that the image is of a different size then the image in the library. In this case, we should be able to resize the image from the library to match the one we are compressing. As long as the image has been resized in an algorithmic way we can describe the image by specifing the value from the library along with the scaling factor. In this case we'll represent the scaling factor as $\frac{x}{2000}$, and allow scaling factors from $\frac{1}{2}$ to 2. Thus we will encode each scientists image as the index from the library along with the scaling factor.

$$K(X|C) = \lceil \log_2 3 \rceil + \lceil \log_2 2000 \rceil + c \tag{3.11}$$

For the case of the solid square, it has to be described as the color and the scaling factor:

$$K(X|C) = 8 + \lceil \log_2 2000 \rceil + c \tag{3.12}$$



(a) Context Stick Man (b) Stick Man Image (c) Difference Image

Figure 3.6: The library of images.

For the random image, scaling up can be describe as the original random image and the scaling factor:

$$K(X|C) = 8p + \lceil \log_2 2000 \rceil + c \tag{3.13}$$

where p is the number of pixels in the pre-scaled image. However, Kolmogorov complexity is defined as the shortest program that produces the result, and this is not the most efficient method to describe a scaled down random image. Rather we can encode the image directly:

$$K(X|C) = 8s \tag{3.14}$$

where s is the number of pixels in the scaled image. Note that when $s = p$ both methods will be approximately equal in length. Figure 3.5 shows the ASC for the images and varying resizes. For the scientists, the ASC increases as the scale does. It increases quickly for scales below one, whereas it only increases slowly for scales above 1. This is because scaling up the original images introduces redundancy into the images which PNG compresses very well. Thus the complexity increases slowly. Random noise shows ASC after passing the 1.0 point as well because while the base image is random, redundancy is introduced by the scaling process.

3.3.4 Repeated Element

Figure 3.6 shows two images which both share a stick man figure. Otherwise the images are random noise. We will consider the image on the left to be our context,

Table 3.2: The PNG complexity length for the various man images.

Name	PNG Complexity
Context	216536
Image	216656
Difference	211832

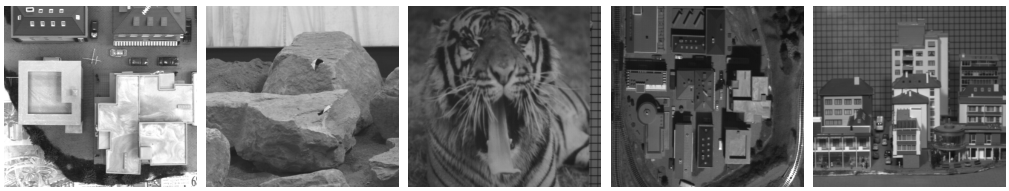


Figure 3.7: A collection of images.

and attempt to compress the image on the right. The second image can be described as the stick figure from the first image together with the difference encoded as an image. The difference is shown in Figure 3.6(c). Note that the noise in the bounding box of the stickman in Figure 3.6(c) is calculated such that adding it to the noise around the stick figure in the library image will produce the noise from the target image. Table 3.2 shows the number of bits required to describe the images by PNG. To actually describe the image then requires specifying the bounding box of the stickman in the original image, 4 coordinates, as well as the target in the current image, 2 coordinate. Since the images are 400x400 pixels, this will require:

$$6 \log_2 400 \approx 52\text{bits}. \tag{3.15}$$

Thus

$$K(X|C) = 52 + l + c \tag{3.16}$$

where l is the length of the png compression of the difference image. In this case, $l = 211,576$ so $K(X|C) \leq 211628 + c$ and $ASC \geq 216496 - 211628 - c = 4868 - c$ bits.

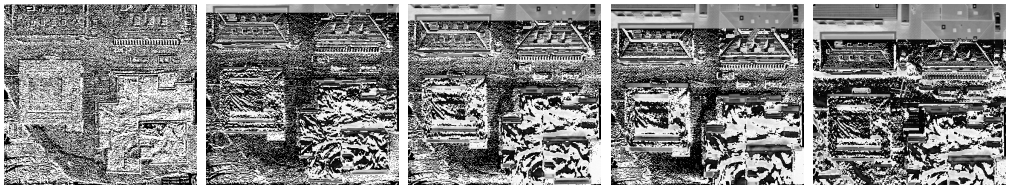


Figure 3.8: Aeriel city shot difference images.

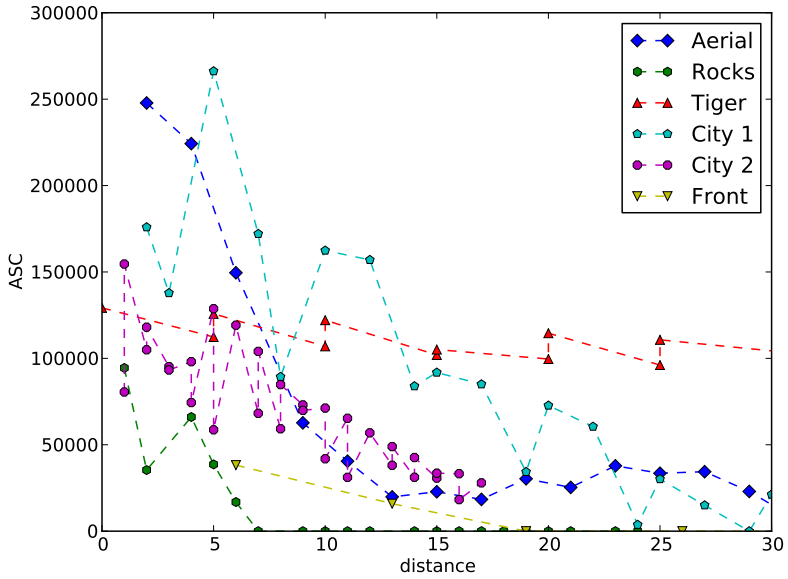


Figure 3.9: ASC plotted by image collection and offset distance.

3.3.5 Photos

Two photographs taken of the same object will differ slightly in all sorts of ways. For example, the picture may shifted, and the noise will be different. Figure 3.7 shows a collection of images[26]. Each image is representative of a collection of photos taken of the same object from slightly varying positions. This images can be made to line up by shifting the image by an offset. We take these representative images as our context, and attempt to compress other images in the collection. We do this by recording the needed offset as well as a difference image, samples of which are shown in Figure 3.8. Each image can be described as:

$$K(X|C) = \log_2 |L| + \log_2 w + \log_2 h + l + c \tag{3.17}$$

where L is the set of images in the library, w and h are the height of the image, and l is the png length of the difference image. The $\log_2 |L|$ term is to determine which image from the library should be used. The w and h are present to specify the offset between the library image and the current one.

Figure 3.9 shows a scatter plot of the ASC. Each point is a single image's ASC using the context of the images shown in Figure 3.7. The x axis, distance, is the manhattan distance of the shift required to line up the two images. For most of the collections, the ASC moves towards zero as the shift required increases. An exception is the tiger images which maintain most of their ASC value. However, images with small shifts contain significant amounts of ASC. This means that we can conclude that the other images are not simply random noise. They share too much similairty with the random image to be generated by a stochastic process, even one that introduces redundancies into images.

3.4 Conclusion

We have sought to demonstrate the applicability of ASC to differentiating noise from patterns. Given a context of known images, we have demonstrated they can be used to compress related images. This indicates specification of those related images.

We have estimated the probability of various images by using the number of bits required for the PNG encoding of the image. This allows us to approximate the ASC of the various images. We have shown hundreds of thousands of bits of ASC in various circumstances. Given the bound established on producing high levels of ASC, we can conclude that the images contain information are not simply noise. We have shown in a variety of circumstances that random noise does not produce ASC. Additionally, the simplicity of an image such as the solid square also does not exhibit ASC, as it is account a high probability. Thus we have demonstrated the theoretical applicablity of ASC to the problem of distinguishing information from noise.

CHAPTER FOUR

The Game Of Life

4.1 Introduction

A machine is an arrangement of parts that exhibit some functionality. 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. The whole is greater than the sum of the parts. Almost any other arrangement of the same parts would not produce anything interesting.

Arranging a large collection of parts into a working machine is highly improbable. However, any particular arrangement would be improbable regardless of whether that arrangement had any functionality whatsoever. Functional machines are specified, they follow some independent pattern. When something is both improbable and specified, we say that it exhibits *specified complexity*[9]. A functional machine is an example of the idea of specified complexity.

To analyze physical machines in depth would be intractable due to the complexity of both physics and machines in the real world. However, we can ameliorate this problem by using a simplified model of reality which nonetheless has many machines operating in it. An example is Conway's *Game of Life* [27]. This is a cellular automata, a two-dimensional grid of living and dead cells that develop based on simple rules. 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, many patterns (essentially machines) have been discovered or invented. These are particular arrangements of living and dead cells that when left to operate by the rules of the game, exhibit some sort of functionality. 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 [28]. Our goal is to differentiate these different categories of patterns. We would like to be able to quantify what separates a simple glider, readily produced from almost any randomly configured soup from Gemini, the product of much careful work.

Specified complexity is the mark of non-randomness, of design. A highly probable object can be explained by randomness, but it will lack complexity and thus not have specified complexity. Random noise will be improbable, but will lack specification and thus also lack specified complexity. In order to have specified complexity, both parts must be present. The object must exhibit a pattern while being improbable.

How does one measure specification? One possibility is to use Kolmogorov complexity [2, 3, 4]. Kolmogorov complexity or variations thereof have been previously proposed as a way to measure specification [9, 10, 11]. Kolmogorov 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.

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

Conditional Kolmogorov complexity [12] allows programs to have input, Y, which is not tallied in the final compression.

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

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 [29]. 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 to an entry in the alphabet list, or indexing an entry in the concordance. Shakespeare

used 31,534 different words [30]. Although both characterizations only bound the conditional Kolmogorov complexity (additional compression is possible), we would expect

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

The more specific the context, the smaller the conditional Kolmogorov complexity. 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 Kolmogorov complexity even further. Small conditional Kolmogorov complexity can be caused by 1) placing X in the context of Y, and/or 2) a small (unconditional) Kolmogorov complexity, $K(x)$.

Algorithmic specified complexity (ASC) [19] is defined as,

$$ASC(X, C, P) = I(X) - K(X|C) \tag{4.1}$$

where

- X is the object or event or under consideration
- C is the context, given information which can be used to describe the object
- $K(X|C)$ is the Kolmogorov complexity of object X given context C.
- $P(X)$ is the probability of X under the given stochastic process.
- $I(X) = -\log_2(P(X))$ is the corresponding self information.

ASC is probabilistically rare in the sense that [20]

$$\Pr[ASC(X, C, P) \geq \alpha] \leq 2^{-\alpha}. \tag{4.2}$$

ASC provides evidence that a stochastic outcome modeled by a particular distribution, $P(X)$, does not explain a given object. ASC is incomputable because Kolmogorov complexity is incomputable [13]. However, the true Kolmogorov complexity is always equal to or less than any estimate. This means that the true ASC is always equal to or more than an estimate. We will refer to known estimate as *observed algorithmic*

specified complexity (OASC). We know that

$$ASC(X, C, P) \geq OASC(X, C, P). \tag{4.3}$$

ASC can be nicely illustrated using various functional patterns in Conway's the *Game of Life*. The Game of Life and similar systems allow a variety of fascinating behaviors [31]. 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 calculuate 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 non-trivial. In order to approximate the probabilties, 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 don't 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 pattern exist, but it is not possible to conclude that they were not produced by random configurations.

Section 4.2 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 4.3 uses this methodology to calculate ASC for a variety of patterns found in the Game of Life.

Table 4.1: The 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-Diference	$\diagdown$	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	$+, -, /, *$	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

4.2 Methods

4.2.1 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 neighbours dies.

(2) *Family.* A living cell with two or three live neighbours lives on to the next generation.

(3) *Overcrowding.* A living cell with more than three living neighbours dies.

Figure 4.1: The block, a simple still life.

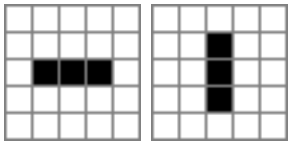


Figure 4.2: The blinker, a simple period-2 oscillator.

(4) *Reproduction.* A dead cell with exactly three living neighbors becomes a living cell.

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 will devise a mathematical formulation for describing this functionality. 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:

$$X = X \oplus . \tag{4.4}$$

This says that a pattern does not change from one iteration to the next. This is known as a *still-life* [27], and an example is presented in Figure 4.1 A more interesting

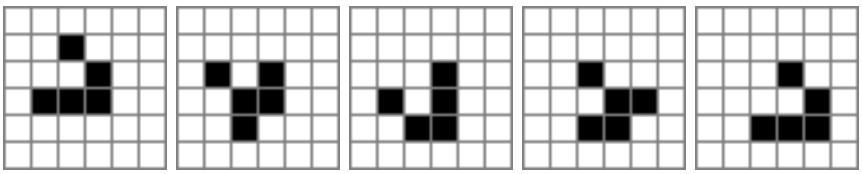


Figure 4.3: The glider, a simple spaceship.

pattern can be described as:

$$X = X \oplus \oplus \tag{4.5}$$

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 completely define the two iteration flip-flop, we will actually require two equations:

$$X \neq X\oplus \tag{4.6}$$

$$X = X\oplus^2 \tag{4.7}$$

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

$$X = X\oplus^i \tag{4.8}$$

to mean a pattern that repeats in i iterations, but not in less than i iterations. An example for $i = 2$, shown in Figure 4.2, is a period-2 oscillator[32] or a flip-flop [27].

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 Figure 4.3. In order to represent movements we introduce arrows, so $X \uparrow$ is the pattern X shifted up one row. Since four iterations regenerates the glider shifted one unit to the right and one unit down, we can write

$$X \downarrow\rightarrow = X \oplus^4 . \tag{4.9}$$

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,

$$X = \tag{4.10}$$

Note that to the right of the equals sign here is a small picture of the glider in Figure 4.3. We can also combine patterns, for example taking the union:

$$ = \cup , \tag{4.11}$$

or the intersection:

(4.12)

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:

(4.13)

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

$$Y := X\oplus^{32} \tag{4.14}$$

$$Y = Y\oplus^{32} \tag{4.15}$$

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

$$X\oplus^{32} = X \oplus^{64} . \tag{4.16}$$

Table 4.1 provides a listing of all of the supported operations.

More than one pattern X will generally satisfy any given equation. In fact, some equations will admit an infinite set of patterns that satisfy the constraints. We will resolve this by defining a total ordering on the set of patterns. Any given pattern can be identified by a set of equations and a non-negative integer identifying which satisfying pattern is wanted.

In theory, a Game of Life pattern could have an infinite number of live cells. This poses a problem for attempting to assign an integer to each living cell. An infinite number of living cells requires an initialization of an infinite number of living cells. We will constrain initialization to a finite number of living cells cells to avoid such occurrences.

Another issue is that there is an infinite number of patterns for any given number of living cells. For example, two living cells could be seperated by any amount of space. However, because a cell is only affected by its immediate neighbors, cells cannot affect

the state of other cells which are sufficiently far away. 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.

The full ordering can be defined by a set of rules with lower-numbered rules having priority:

(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 Figure 4.3 and reading left to right then down gives $010001111 = (143)_{10}$.

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:

$$X \downarrow\rightarrow = X\oplus^4, \#0 \tag{4.17}$$

as the smallest pattern which fits the description.

4.2.2 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

Table 4.2: Binary Encodings.

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
$*$	10010
$/$	10011
$:=$	10100
$=$	10101
$\neq$	10110
(repeat)	10111

Literals

Type	Encoding
Number	11001
Pattern	11010

Special

Type	Encoding
Stop	11111

in Table 4.2. 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 4.2. Such operations are simply encoded using their 5 bit sequence. Since they have no arguments, their sequence is completed directly after the five bits. 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 the by representation of the subexpression. Thus $X\uparrow$ can be represented as 0100100000. Since the subexression 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 five 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 4.2 are denoted by the five 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[33]. See Section 4.2.3 for a discussion of binary encodings for arbitrary patterns.

To declare there are no more operations to be had, we will use a 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:

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

4.2.3 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 4.2.2 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

$$\alpha(p) = l(p_w) + l(p_h) + p_w p_h \tag{4.18}$$

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.

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

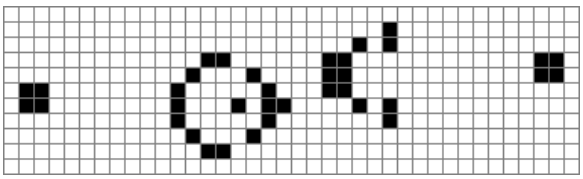


Figure 4.4: The Gosper gliding gun.

number of live bits and then we can encode the actual pattern using less bits:

$$\beta(p) = l(p_w) + l(p_h) + l(p_a) + \left\lceil \log_2 \binom{p_w p_h}{p_a} \right\rceil \tag{4.19}$$

where p_a is the number of alive pixels in pattern, p We will call this the *compressed encoding*.

To demonstrate these methods, consider the *Gosper gliding gun* in Figure 4.4. Using the standard encoding this requires

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

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:

$$P(p) = 1 + \min(\alpha(p), \beta(p)) \tag{4.20}$$

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.

$$\begin{aligned}\Pr[X = p] &= \Pr[X = p|C] \Pr[C] \\ &+ \Pr[X = p|\overline{C}] \Pr[\overline{C}]\end{aligned}$$

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 each method is equally probable:

$$\begin{aligned}\Pr[X = p] &= \frac{2^{-\alpha(p)}}{2} + \frac{2^{-\beta(p)}}{2} \\ &= \frac{2^{-\alpha(p)} + 2^{-\beta(p)}}{2}\end{aligned} \tag{4.21}$$

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 (4.20) 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.

4.2.4 Computability

The mathematical formulation 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 [34]; however, our equations using the components in Table 4.2 describing the Game of Life are not. There are concepts that cannot be described using the operations we have defined. However, the proof on the bound of ASC only requires that the language used to

describe the pattern is prefix-free. Thus the theoretical results regarding the bound ASC 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. 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 langer than the OASC results presented here.

The language used here is used in part for the simplicity in understanding. It allows the comparison of the complexity of various specifications without constants which is difficult in standard Kolmogorov 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 towards the patterns found in the Game of Life.

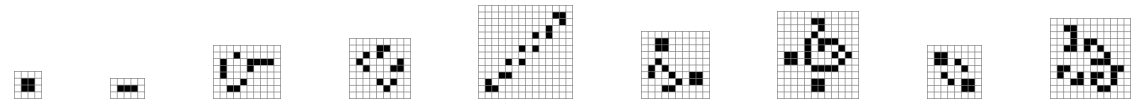
4.3 Results

4.3.1 Oscillators

The simplest oscillator is one which does not actually change, that is a still life. An example is depicted in Figure 4.1. This object can be described as:

$$X = X\oplus, \#0 \tag{4.22}$$

as this is the smallest pattern that can fit the test. There are four symbols taking twenty 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 $l(2) + l(2) + 2 * 2 + 1 = 4 + 4 + 4 = 13$. Thus the



(a) block (b) blinker (c) caterer (d) mazing (e) pseudo-barberpole (f) unix (g) burloaferimeter (h) figure eight (i) 29p9

Figure 4.5: The smallest known oscillators for each category.

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 Figure 4.2 can be described as:

$$X = X\oplus^i, i = 2, \#0 \tag{4.23}$$

This takes 6 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 modelled stochastic process well.

However, the same pattern could be described as:

$$X = \text{}, \#0 \tag{4.24}$$

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 4.2.4. Thus the length is $3 * 5 + 5 + l(0) + 11 = 20 + 1 + 11 = 32$ 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. Figure 4.5 shows the smallest known oscillators for periods up to nine. Smaller oscillators

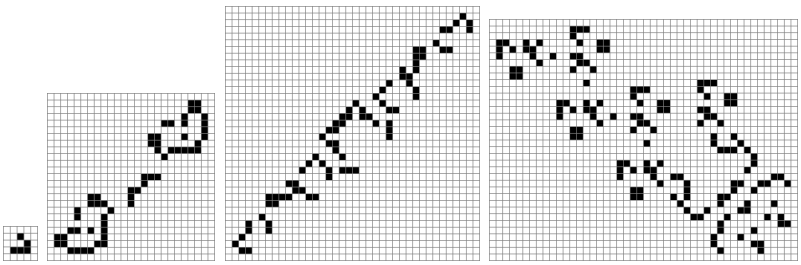
Table 4.3: ASC for the smallest known oscillators in each category.

Name	Period	$I(X)$	$K(X\|C)$	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	$1.742 * 10^{-21}$	

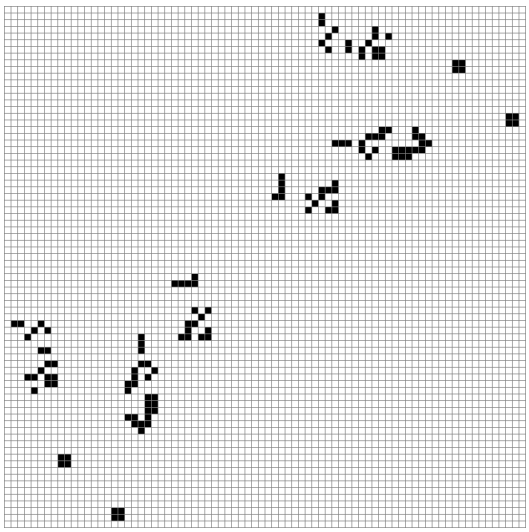
than these may exist, but for now we believe these to be the ones described by the formulation. Table 4.3 shows the calculated values of OASC for the various oscillators. The $\Pr[X]$ column derives from experiments on random soups [35]. The missing entries do not appear to have been observed in random 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 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 isn't captured by our metric.

Any pattern in the Game of Life can be constructed by colliding gliders [32]. 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 much less 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.



(a) glider (b) 58P5H1V1 (c) 77P6H1V1 (d) 83P7H1V1



(e) Four engine cordership

Figure 4.6: The smallest known spaceships for each speed moving diagonally.

4.3.2 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 Figure 4.3. We previously showed in Equation 4.9 that it could be described as

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

this has 8 symbols so the length will be $5 * 8 + 5 + l(4) + l(0) = 45 + 6 + 1 = 52$. The complexity is 20 and the ASC is at least $20 - 52 = -32$ bits. As previously noted, any

Table 4.4: 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

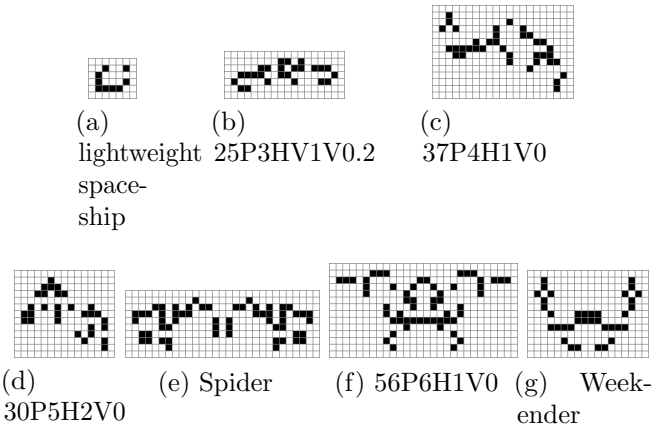
pattern can be described such that it has at least -21. This matches the observation that glider often arise from random configurations.

As with oscillators we can readily describe the smallest version of a spaceship. In addition to varying with respect to the 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 gilder. In general to travel with a speed of c/s with period p can be described as

$$X \downarrow^{\frac{p}{s}} \rightarrow^{\frac{p}{s}} = X \oplus^{p}, \#0 \tag{4.25}$$

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 $5 * 12 + 5 + l(\frac{p}{s}) + l(p) + l(0) = 66 + l(\frac{p}{s}) + l(p)$.

Figure 4.6 shows the smallest known diagonally moving spaceships for different speeds. If we assume that these are the smallest spaceships for these speeds, then Equation 4.25 describes them. Table 4.4 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 Figure 4.6 where the remaining patterns are much larger than the glider.



(a) lightweight spaceship (b) 25P3HV1V0.2 (c) 37P4H1V0

(d) 30P5H2V0 (e) Spider (f) 56P6H1V0 (g) Weekender

Figure 4.7: The smallest known spaceships for each speed moving orthogonally.

Table 4.5: 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

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:

$$X \uparrow^{\frac{p}{s}} = X \oplus^{p}, \#0 \tag{4.26}$$

The length of this is $5 * 9 + 5 + l(\frac{p}{s}) + l(p) + l(0) = 51 + l(\frac{p}{s}) + l(p) + l(0)$. 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. Figure 4.7 shows the smallest known spaceship for a number of different speeds. Table 4.5 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 [36]. The remaining spaceships exhibit significant amounts

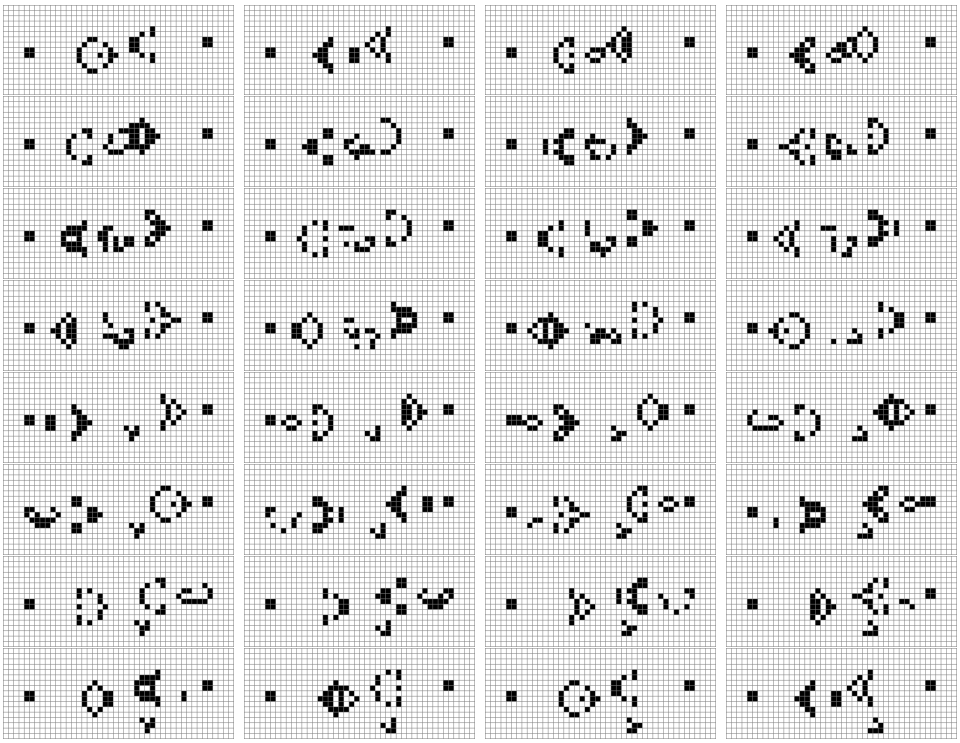


Figure 4.8: 31 iterations of the Gosper gun.

of ASC, although not as much as the diagonal spaceships, and are not reported to have been observed arising at random.

4.3.3 Guns

Figure 4.8 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:

$$X\oplus^{30} = X \cup \blacksquare \rightarrow^{24}\downarrow^{10}, \#0 \tag{4.27}$$

That is, the configuration after thirty 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

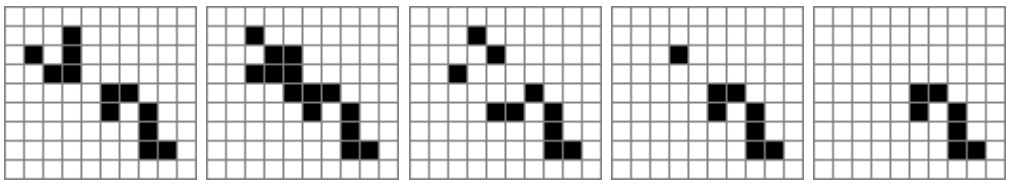


Figure 4.9: The glider being eaten by the eater.

us $196 - 111 = 85$ bits of OASC. At a probability of 2^{-85}, we conclude the Gosper gun is unlikely to produced by a random configuration.

4.3.4 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 Figure 4.9, 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:

$$X = X\oplus \tag{4.28}$$

Secondly, it must recover from eating the glider:

$$(X \cup \blacksquare\uparrow^3\leftarrow^4)\oplus^4 = X \tag{4.29}$$

The two equation have a total of 18 symbols, and the glider will require 20 bits to encode. Thus the total length will be $5 * 18 + 5 + 20 + l(3) + l(4) + l(4) + l(0) = 5 * 18 + 5 + 20 + 4 + 7 + 7 + 1 = 134$ 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.

4.3.5 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[32]. Experiments have been performed to measure the

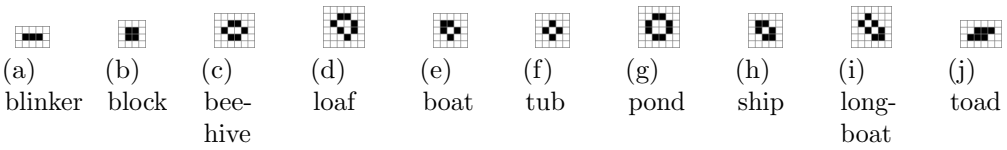


Figure 4.10: The ten most common Game of Life ash objects.

frequencies of various objects arising from this soup [35]. Figure 4.10 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 x 4 which will require at most $1+l(4)+l(4)+16 = 1+7+7+16 = 31$ 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.

4.4 Conclusions

We have demonstrated the ability to describe functional Game of Life pattern using a mathematical formulation. This allows us to compress, theoretically if not practically, Game of Life patterns which exhibit some functionality. Thus ASC has the ability to capture functionality.

We made a simplifying 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. For the most part, this appeared to work, with the exception of the unix pattern. However, even that pattern was less then an order of magnitude more probable then the bound suggested. This suggests for the most part that the approximation was reasonable, although it could be improved.

We conclude that many of the machines built in life do exhibit 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 our method.

BIBLIOGRAPHY

[1] C. E. Shannon, W. Weaver, and N. Wiener, "The Mathematical Theory of Communication," *Physics Today*, vol. 3, no. 9, p. 31, 1950.

[2] G. J. Chaitin, "On the length of programs for computing finite binary sequences," *Journal of the ACM (JACM)*, vol. 13, 1966.

[3] R. J. Solomonoff, "A preliminary report on a general theory of inductive inference," Zator Co. and Air Force Office of Scientific Research, Cambridge, Mass, Tech. Rep., 1960.

[4] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *International Journal of Computer Mathematics*, 1968.

[5] G. J. Chaitin, *Conversations with a mathematician : math, art, science, and the limits of reason : a collection of his most wide-ranging and non-technical lectures and interviews.* New York, New York, USA: Springer, 2002.

[6] L. A. Levin, "Various Measures of Complexity for Finite Objects. (Axiomatic Description)," *Soviet Math*, vol. 17, no. 522, 1976.

[7] C. H. Bennett, "Logical depth and physical complexity," *The Universal Turing Machine A Half-Century Survey*, pp. 227–257, 1988.

[8] C. H. Bennett, P. Gács, M. Li, P. M. Vitányi, and W. H. Zurek, "Information distance," *Information Theory, IEEE Transaction on*, no. 8556, pp. 1–29, 1998. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=681318

[9] W. A. Dembski, *The Design Inference: Eliminating Chance through Small Probabilities.* Cambridge University Press, 1998, vol. 112, no. 447.

[10] ——, *No Free Lunch: Why Specified Complexity Cannot Be Purchased Without Intelligence.* Lanham MD: Rowman & Littlefield, 2002.

[11] ——, "Specification: the pattern that signifies intelligence," *Philosophia Christi*, vol. 7, no. 2, pp. 299–343, 2005.

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

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

[14] O. E. D. Online, "Oxford English Dictionary Online," 2012. [Online]. Available: http://dictionary.oed.com

[15] D. D. Axe, "Estimating the prevalence of protein sequences adopting functional enzyme folds." *Journal of molecular biology*, vol. 341, no. 5, pp. 1295–315, Aug. 2004.

[16] K. K. Durston, D. K. Y. Chiu, D. L. Abel, and J. T. Trevors, "Measuring the functional sequence complexity of proteins." *Theoretical biology & medical modelling*, vol. 4, p. 47, Jan. 2007.

[17] D. L. Abel and J. T. Trevors, "Three subsets of sequence complexity and their relevance to biopolymeric information." *Theoretical biology & medical modelling*, vol. 2, p. 29, Jan. 2005.

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

[19] W. Ewert, W. A. Dembski, and R. J. Marks II, "Algorithmic Specified Complexity," in *Engineering and Metaphysics*, Tulsa, OK, 2012.

[20] ——, "On The Improbability of Algorithmic Specified Complexity," in *2013 IEEE 45th Southeastern Symposium on System Theory: SSST 2013*, Waco, TX, 2013.

[21] N. Nikvand and Z. Wang, "Generic image similarity based on Kolmogorov complexity," *Image Processing (ICIP), 2010 17th IEEE ...*, pp. 309–312, 2010.

[22] S. Supamahitorn, "Investigation of a Kolmogorov Complexity Based Similarity Metric for Content Based Image Retrieval," 2004.

[23] M. Kramm, "Image group compression using texture databases," *Electronic Imaging 2008*, 2008. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=811112

[24] J.-D. Lee, S.-Y. Wan, C.-M. Ma, and R.-F. Wu, "Compressing sets of similar images using hybrid compression model," *Multimedia and Expo, 2002. ...*, no. l, pp. 9–12, 2002.

[25] T. Boutell, "PNG (Portable Network Graphics) Specification Version 1.0," 1997.

[26] C.-C. Wang, "Vision and Autonomous Systems Center's Image Database." [Online]. Available: http://vasc.ri.cmu.edu/idb/html/motion/index.html

[27] M. Gardner, "Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life"," *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.

[28] A. P. Gouche, "Oblique Life spaceship created," 2010. [Online]. Available: http://pentadecathlon.com/lifeNews/2010/05/oblique_life_spaceship_created.html

[29] "Concordance of Shakespeare's complete works," *Open Source Shakespeare*. [Online]. Available: http://www.opensourceshakespeare.org/concordance/

[30] J. Bennett, *Statistical Reasoning for Everyday Life*, 2nd ed. Allyn & Bacon, 2002.

[31] S. Wolfram, *A New Kind Of Science*. Champaign, IL: Wolfram Media, 2002.

[32] "LifeWiki." [Online]. Available: http://www.conwaylife.com/wiki/Main_Page

[33] D. Salomon, *Variable-Length Codes for Data Compression*, 2007.

[34] A. Adamatzky, Ed., *Collision Based Computing*. Springer Verlag, 2002.

[35] A. Flammenkamp, "Top 100 of Game-of-Life Ash Objects," 2004. [Online]. Available: http://wwwhomes.uni-bielefeld.de/achim/freq_top_life.html

[36] ——, "Spontaneous appeared Spaceships out of Random Dust," 1995. [Online]. Available: http://wwwhomes.uni-bielefeld.de/achim/moving.html