

An Evaluation of “Ev”

I.G.D. Strachan

June 30, 2003

“Life isn’t like that. Evolution has no long term goal” – Richard Dawkins

Abstract

In this paper we assess the validity of the evolutionary simulation described in the paper “Evolution of Biological Information” [Schneider, 2000], which, it is claimed, demonstrates the evolution of new biological information from scratch with no external intervention, as a set of characteristic patterns developed in nucleotide binding sites. The further claim is made that the amount of information that evolves, a quantity designated $R_{sequence}$, is approximately equal to the amount that is needed to locate the binding sites, given the number of them on the genome. This quantity is designated $R_{frequency}$. We find both these claims to be flawed, firstly that information arises from scratch, when in fact all the simulation is demonstrating is a form of “supervised learning” of a fixed target by a simple form of neural network. Secondly, we demonstrate that for the neural network classifier used in the simulation, that $R_{sequence} < R_{frequency}$ except for the unrealistic exceptional case where all the information is confined to one axis (one base in the binding site). We also show that the result showing $R_{sequence} \approx R_{frequency}$ reported in the Schneider paper is a coincidence, resulting from an incorrect evaluation of the correction for small sample sizes, which fails to take into account the fact that the standard formula for uncertainty (or entropy) is a limiting case that utilizes Stirling’s approximation for $\ln N!$, which is not valid for small values of N . Analytical results are backed up by simulations, including an extensive set of runs using Schneider’s Pascal program `ev.p`.

1 Introduction

The paper “Evolution of Biological Information” [Schneider, 2000] presents a computer simulation of the evolution of information at nucleotide binding sites. The process by which such information arises is one of great interest, as it is observed in nature that the amount of information in the binding sites, denoted $R_{sequence}$, is approximately equal to the amount required for the spliceosome recognizer protein to locate them. The example is given in the introduction to Schneider’s paper of human splice acceptor sites, where there are found to be about 9.4 bits of information per site. The average distance between the sites is 812 nucleotide bases, and hence the amount of information required to locate them (and not make false recognitions of places which are not binding sites), is $\log_2 812 = 9.7$ bits. This quantity is designated $R_{frequency}$. The intention of Schneider’s paper was to demonstrate how this approximate relationship:

$$R_{sequence} \approx R_{frequency} \tag{1}$$

can come about during the evolutionary process.

Results were presented from one run of a Pascal program `ev.p` (referred to as **Ev**). Starting from a random population of 64 “creatures”, each consisting of 266 nucleotide bases, a creature evolved that located the binding sites perfectly with no false recognitions after a short period of mutation and natural selection, (just 704 generations). Furthermore the approximate relationship $R_{sequence} \approx R_{frequency}$ was demonstrated by the simulation. There were 16 binding sites to find, and 256 positions scanned by the simulated “recognizer protein”, meaning that $R_{frequency} = \log_2(256/16) = 4$ bits of information per binding site would be required to locate them. This was illustrated in Figure 2b of [Schneider, 2000], where the information in bits per site was shown to rise steadily up to the required value of around 4 bits. When simulation was continued after a “perfect” creature had evolved (with respect to the designated task of correct recognition of binding sites), then the figure for $R_{sequence}$ was shown to hover around 4 bits (actually 3.983 ± 0.399 bits).

These results look impressive, and apparently justify the claims that information can arise from scratch in an evolutionary process, and that the approximate relation (1) is obeyed. Schneider accepts that the question as to how life gains information is “a valid issue recently raised by creationists” (Truman, R. (1999), <http://www.trueorigin.org/dawkinfo.htm>), and claims that this paper quantitatively addresses that issue.

On careful examination of the paper, however, we find that the claims made on behalf of the simulation are seriously flawed, on three counts, namely:

1. The information that is supposed to arise “from scratch” without any external intervention, is in fact put into the simulation from the start. In order to be able to locate the binding sites, their locations have to be specified before the simulation can run. What follows during the simulation is simply a form of *Supervised Learning*, of a simple type of neural network (called a *Perceptron*). The “Supervision” in a supervised learning procedure alludes to the process where at each round of training, the outputs of the neural network are compared to a *specified target*, and corrections for errors are made accordingly.
2. The calculation of the information content for the binding sites is done in the paper by computing the information content at each of the six locations in the binding site separately, using all 16 sites as a sample, and then summing them. This relies on an important simplifying assumption that the values of the bases at each of the locations are statistically independent. This assumption is acknowledged in [Schneider et al., 1986], where the exact formulae for the information calculations are given, but it appears that in the current paper, the assumption is taken for granted. However, it can be easily demonstrated that this assumption *does not apply* in general for the simple perceptron recognizer used in the simulation. The only case where the perceptron would conform to this independence assumption in fact corresponds to the case where all the information is concentrated onto one axis (i.e. input variable to the perceptron), and the rest are unspecified; a situation that does not, and could not occur in the simulation.
3. Further to this, we find that the actual formula for computing $R_{sequence}(L)$, the information content for one location L in the site, is also flawed, relying on a standard formula for statistical entropy, or uncertainty, which itself can be derived only as an asymptotic limit as the number of samples tends to infinity, after applying Stirling’s approximation for $\ln(N!)$. It can be demonstrated that this approximation is invalid for the small sample sizes ($N = 16$) that are used in the simulation.

The rest of the paper is organized as follows:

- In Section 2, we describe in detail the recognition mechanism used in the **ev** simulation, and show how it corresponds to supervised learning of a pre-specified target, thereby negating the claim that the information has arisen from scratch. We note that the simulation is subject to the same criticism that Richard Dawkins made of his own “Weasel” simulation in “The Blind Watchmaker”, namely that it involves a long-term goal that would not be present in real evolution. We compare and contrast Dawkins’ simulation and Schneider’s, and propose an extension to the Dawkins algorithm that removes the principal difference between the two simulations, and still shows that a fixed long-term goal is involved.
- In Section 3, we analyse in detail the way the quantity $R_{sequence}$ is calculated. We show how the Perceptron recognizer violates the independence assumption stated in [Schneider et al., 1986], and also how the formula for $R_{sequence}(L)$ is also invalid at small sample sizes. We present an alternative formula that can be used, that will give a more valid estimate of $R_{sequence}$.
- In Section 4, we present a set of simulations obtained by using the **ev** program, suitably adapted to perform the original calculations, and the exact formula derived in the previous section. The results obtained are shown to be roughly in line with the theoretical analysis presented in the previous section, and show furthermore that even given the incorrect method of computing $R_{sequence}$ used in the original simulation the relationship $R_{sequence} \approx R_{frequency}$ is a coincidence. Plotting $R_{sequence} - R_{frequency}$ as a function of number of sites shows a clear trend that agrees best with the expected value of zero when the number of sites is equal to 16, coincidentally the number used in the single reported simulation in [Schneider, 2000].
- Finally, in Section 5, we summarize the findings, and discuss the claims of the Schneider paper, which is ultimately based on a research paper from 1982, in the light of more recent research.

2 Supervised learning of pre-specified information

In this section we review the recognition mechanism employed in the **ev** program, and demonstrate that it corresponds to the process of *Supervised Learning*, where a neural network is trained to reproduce a fixed target.

2.1 Review of the Ev program

In the **ev** run reported in [Schneider, 2000], the evolution of a population of 64 “creatures” is simulated. Each creature is represented by a string of nucleotide bases, which may take the values $\{A, C, G, T\}$. The genome of each creature is considered to be separated into two regions, the first of which is a “recognizer gene”, that is made to scan over the whole genome, using a 6-base window of nucleotides. It is a classification system that outputs a logical 1 if the particular 6 base sequence is “recognized” as a binding site, and a 0 if it is not recognized. The remainder of the genome, after the recognizer gene, is where the binding site locations are specified, chosen arbitrarily by the program prior to the start of the simulation. In the simulation run reported in the paper, 256 locations on the

genome are scanned (including the area designated as the recognizer gene, in which there are no binding sites). Since there are six bases in a binding site, there have to be at least $256+6-1 = 261$ nucleotides in the gene, so that when position 256 is scanned, the values at $256 \dots 261$ are available as inputs to the recognizer. In fact the program adds an extra five bases onto the end of the genome in addition to this to assist in the graphical displays of the “sequence logos” [Shaner et al., 1993], to show the growth of information content at the sites.

For the recognizer gene, Schneider uses a simple form of neural network, called a *Perceptron*, consisting of a numeric weight vector \mathbf{w} , and a threshold value θ . The recognition action is performed by computing the dot product of an input vector \mathbf{x} with the weight vector, and subtracting the threshold from it. If the resultant value is greater than zero, then the system is considered to have “recognized” the input, and if it is below zero, then recognition is not deemed to have occurred. Hence the recognition criterion is:

$$\mathbf{w} \cdot \mathbf{x} - \theta > 0 \quad (2)$$

The numerical values of the weight vector and the threshold are encoded onto the recognizer gene, by using sequences of 5 bases for each number (as reported in the paper, but the number of bases can be specified in the input file to the program). By encoding $A = 00, C = 01, G = 10$ and $T = 11$ in binary, the five bases yield a 10-bit number that represents integers in the range $-512 \dots +511$.

The inputs to the perceptron are computed by assigning a 1-of-4 code to each of the six bases in the scanned potential site: $A \rightarrow (1, 0, 0, 0), C \rightarrow (0, 1, 0, 0), G \rightarrow (0, 0, 1, 0)$ and $T \rightarrow (0, 0, 0, 1)$. For a sequence of six bases, there are therefore 24 inputs to the perceptron, in six groups of 4. These are described in the paper as a “weight matrix”, reflecting the 6×4 grouping. Each of the six rows in the matrix can only have a single 1, and the rest are zeros. However, the linear algebra in equation (2) implies no matrix operation, only the vector operation of the dot product (taking the pairwise products of the two vectors and summing the result). Because the inputs to the perceptron in this simulation are always only 0 or 1, the equation reduces to summing the weight matrix row entries corresponding to the base found at each position. The operation is illustrated schematically in Figure 1.

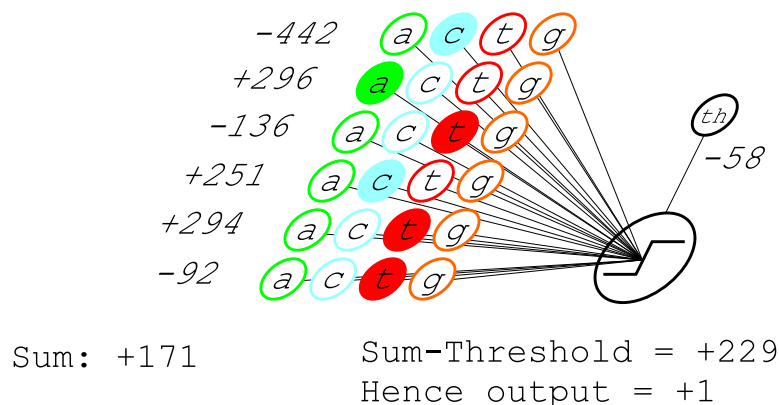


Figure 1: Schematic representation of the action of the perceptron mechanism used in the *ev* program. The sequence *CATCTT* is being tested, and is counted as “recognized” because the final output is above zero (+229).

Here we have represented the weight matrix in a 6×4 array of circular nodes, and the sequence *CATCTT* is being tested. The node corresponding to the base value in each row is shown in solid colour, and weight value (1 of 4 in each row) corresponding to the position of the coloured node is shown to the left of the row. The sum of the weights is 171, which after the threshold value $\theta = -58$ is subtracted off, yields the total +229, which is greater than zero, and hence the sequence is deemed to be “recognized”.

The reason for this complex and sparse representation of the input space (expanding a 6-base sequence to a 24-bit binary vector), may be traced to an earlier paper [Stormo et al., 1982], where Stormo, Schneider, Gold and Ehrenfeucht had used the perceptron technique to classify translational initiation sites in *E. Coli*, and obtained significantly better results than previous rule-based classifier systems. The reason for assigning a 1-of-4 coding to each base as opposed to a 2-bit code, e.g. $A \rightarrow 00, C \rightarrow 01, G \rightarrow 10$ and $T \rightarrow 11$ is due to a fundamental limitation of the perceptron that has been known since the 1960’s [Minsky and Papert, 1969], in that it is only able to solve a restricted sub-class of classification problems that are called “linearly separable”. This is because the weight vector represents an $(N - 1)$ -dimensional hyperplane in the N -dimensional input space that separates the two classes. If a single hyperplane cannot separate two classes of object, then a perceptron will be unable to converge to a solution. A simple example is given by Stormo *et. al.* in this paper, where a 2-bit coding scheme could not separate two classes if the distinguishing rule was an *A* or *T* versus a *C* or *G*. This is illustrated graphically in Figure 2 as a 2-dimensional problem where the separating hyperplane is 1-dimensional (i.e. a straight line).

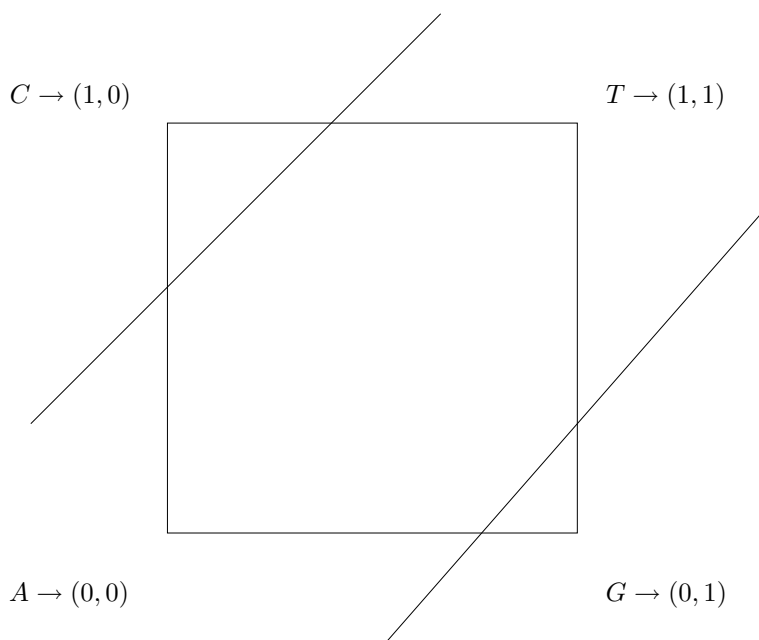


Figure 2: The *Exclusive Or*, or *XOR* problem, which cannot be solved with a single linear decision boundary. *A* and *T* (coordinates 0,0 and 1,1) are a separate class from *C* and *G* (coordinates 1,0 and 0,1).

This is the well-known *exclusive OR*, or *XOR* problem, which was originally discussed by Minsky and Papert in the above citation. The *XOR* function in binary requires the output to be the exclusive *OR* of the two inputs; i.e. only to output a 1 if the inputs differ, otherwise output a zero. This diagram shows that the two classes *A/T* against *C/G* are not linearly separable, because a single straight line cannot be drawn that separates the

region according to class. This kind of problem requires more than one decision plane, which can be provided by the more advanced *Multi-Layer Perceptron*, or MLP [Rumelhart et al., 1986]. The MLP provides extra decision planes by the means of “hidden units”, each of which functions like a perceptron, but a final layer of processing is provided to make a weighted sum of the outputs. (In practice also a smooth sigmoidal function such as $\tanh(x)$ is applied instead of a hard threshold). However, the MLP was not in common use till 1986, and the Stormo *et. al.* paper predates this.

Despite this limitation of the perceptron, it is in fact unnecessary to adopt this coding convention for the **ev** program, because the inputs to the neural network evolve as well as its weights, and so the problem can be avoided. By contrast, in the original work in [Stormo et al., 1982], the neural net was trained by a Perceptron learning rule that worked with a fixed set of inputs (the nucleotide sequence under test), and the output was then compared with a fixed target value (recognized or not-recognized).

In the **ev** simulation, at each stage, the performance of the creature is measured against a (still fixed) target. There are two kinds of mistakes that can be made:

1. A position that is not supposed to be a binding site is falsely recognised.
2. A position that is supposed to be a binding site is not recognised.

At the evaluation stage, the total number of mistakes made by each creature in the population is evaluated. Then natural selection is invoked by sorting the population into ascending order of mistakes made, and the half of the population making the least mistakes is then allowed to wipe out the other half by making copies of itself in their place. In the case of tied mistake counts, various rules are available in the program options for what to do. In the simulations later in this paper, we have used the rule that the creature to survive in the case of a tie is chosen randomly. Following the selection procedure, the whole population is subjected to mutation, where one randomly selected nucleotide per creature is changed to another randomly chosen value (which can be the same). Then follow further rounds of selection and mutation until a creature emerges that makes no mistakes of any kind. In the simulation reported in the paper, such a creature emerges after only 704 generations, and plots of the information content as a function of generation show an inexorable rise in information content up to the required 4 bits per binding site. At this point, no further information can be gained in the simulation, as there are no more binding sites to find (they do not evolve spontaneously, because their locations have to be pre-specified). This is verified in the paper by showing a “noisy stasis” for 1000 generations after the zero mistakes creature had evolved, with the information content varying randomly around a mean value of 3.98 bits.

2.2 Discussion of the Ev program

What has become clear in the preceding section is that the locations of the binding sites have to be pre-specified in order for the simulation to work, otherwise there is no way to determine whether the recognizer has made a mistake or not. This is the basis of our principal objection to the claim that information has arisen from scratch with no external intervention. What is going on is a type of supervised learning, where the target outputs for the neural network (the recognition decisions) are pre-specified for each location in the genome at the outset. The precise location where the “supervision” takes place is shown in the following code fragment (**ev.p** version 3.73 lines 2195-2198).

```

with e.p.creature[bug] do for x := 0 to e.precalc.endscang do begin
  recognized := recognize(w,e.p.width,genome,x,e);
  if e.p.sitelocations[x] <> recognized then m := m + 1;
end;

```

The second line of the code fragment invokes a separate procedure called `recognize` which calculates the perceptron output at position `x` for the creature being evaluated. In the third line, the decision of the perceptron, — the boolean variable `recognized` — is compared to position `x` in a separate boolean array called `sitelocations` which contains the value `true` if position `x` is supposed to be a binding site, and the value `false` otherwise. If the two are different, then the mistake count `m` is incremented. The contents of the boolean array `sitelocations` do not change during the evolutionary part of the simulation, but are instead set up prior to the evolution loop, in a procedure ironically titled `creation`, located at line 1840 in the same file. The action of the recognizer and the mistake counting mechanism is illustrated schematically in Figure 3.

Hence we argue that in fact there is external intervention; the `sitelocations` array has to be set up prior to the simulation, and in doing so, the precise amount of information that is claimed to have evolved from scratch has in fact been pre-specified in the setting up of this array (to specify 16 locations out of 256 requires 4 bits of information per location).

One subtle difference between the `ev` simulation and the Dawkins “METHINKS IT IS LIKE A WEASEL” example, is worth elucidating. In the latter case, the “target” is a fixed sequence of letters (drawn from an alphabet of 26 letters plus a space). It seems that the Schneider illustration has achieved something more than this; many different outcomes in terms of the final sequence of bases in the genome are possible, and it could therefore be argued that there is no pre-specified target. However, there is in fact still a target, in terms of the required outputs of the neural network. The distinction can be expressed succinctly as follows. If we call the genome letter sequence `g`, and the target sequence `t`, then the goal of the Dawkins simulation is to make:

$$\mathbf{g} = \mathbf{t} \tag{3}$$

whereas in the Schneider simulation, the goal is the implicit use of the target:

$$\mathbf{f}(\mathbf{g}) = \mathbf{t} \tag{4}$$

In fact if we choose `f(.)` to be the identity function, then it can be seen that the Dawkins simulation is simply a special case of the form used by the Schneider simulation. In the Schneider simulation, `f(g)` is the perceptron output function. Hence both the Dawkins and the Schneider simulations use a specified target. Note that `f(.)` is a many-to-one mapping, allowing multiple outcomes, whereas the Dawkins simulation is a one-to-one mapping, with a fixed outcome. However, this does not alter the fact that a fixed target has to be specified in order to make the simulation work.

Another way of looking at the distinction between Dawkins and Schneider is to say that the Schneider simulation in fact has a *biochemical target* which is to find an organism that correctly recognizes the binding sites, wherever they might be. This target might be argued to be unspecified, and hence that information arose from scratch.

The problem with this argument is that the *biochemical target*, which corresponds to a vast number of different possible outcomes to the simulation, is still a *fixed subset of the total number of possible genomes*, and is entirely dependent on the *mathematical target* that is defined in `sitelocations` at the outset. Furthermore, in order for the simulation to work, the evolution algorithm has to have sight of the mathematical target, in order to be able to compute the number of mistakes. Therefore `ev` is subject to the same criticism that Richard Dawkins made of the “WEASEL” simulation in “The Blind Watchmaker”.

Dawkins writes:

Although the monkey/Shakespeare model is useful for explaining the distinction between single-step selection and cumulative selection, it is misleading in important ways. One of these is that, in each generation of selective ‘breeding’, the mutant ‘progeny’ phrases were judged according to the criterion of resemblance to a *distant ideal target*, the phrase METHINKS IT IS LIKE A WEASEL. Life isn’t like that. Evolution has no long-term goal.

Although Dawkins’ model collapses the biochemical target and the mathematical target into one, the separation of the two does not achieve anything extra, or reduce the dependence on a distant ideal target. To illustrate this, we propose a small extension to the original Dawkins program that makes the two types of target separate.

Instead of a simple genome that must match the target text, we use instead a composite genome, where the first part contains an encryption key, and the remainder of the genome represents text. In assessing the fitness of each of the progeny at each generation, we apply Vignere decoding to the text segment, using the encryption key in reverse to decrypt the text. Once decoded, like the Dawkins program, the fitness is decided according to the resemblance to an arbitrarily chosen piece of text (in our case, the phrase `THE ANSWER IS FORTY TWO`).

Details of how the decoding is done, and a sample output from a MATLAB simulation of this evolutionary process are shown in Appendix A.

The extended Dawkins type simulation is now very much more similar to `ev`, in that there are now many possible outcomes (in the simulation given, there are 8 letters in the encryption key, giving $27^8 = 2.82 \times 10^{11}$ possible outcomes). For each of these, the text segment of the genome will be different, but the entire set of successful (zero mistakes) genomes will always completely be defined by the target string. We still “told it the answer”¹.

Our simulation also mimics `ev` in that the encryption key is analogous to the recognizer gene, the text segment is analogous to the binding site region of the DNA, and the target text string is analogous to the pre-specified pattern in `sitelocations`.

The only significant detail missing in our simulation is the calculation of information content. Even this would be possible if the target text were much longer, allowing frequency distributions of the individual letters to be calculated. For example, in the case

¹The number 42 was given as the answer to “the ultimate question about life, the universe and everything” in Douglas Adams’ radio series “Hitchhiker’s Guide To The Galaxy”. The answer was the outcome of a 7.5 million year calculation by a computer named “Deep Thought”. Our calculation took rather less time than this, because “Deep Thought” had to work out the answer from first principles, rather than being told what it was at the outset.

where there were 2 letters in the encryption key, one could calculate frequency distributions of the letters separately on the odd and even numbered letters in the text segment, and these would be similar to the overall frequency distribution of the letters in the original text, but cyclically shifted according to the corresponding letter in the encryption key. Hence the simulation would show an information gain, equivalent to the information put in (the distribution of letters in English text is of course non-uniform, and will have a lower entropy than a uniform random distribution).

2.3 Binding sites forming next to pre-existing functional proteins

It might be argued that the pre-specification of the binding site locations is “as in nature”, or that the exons in the genome are already present, thereby defining the binding site locations. In the paper, Schneider notes that:

An advantage of the **ev** model over previous evolutionary models, such as biomorphs, [Dawkins, 1986], Avida, [Lenski et al., 1999], and Tierra, [Ray, 1994], is that it starts with a completely random genome, and no further intervention is required. Given that gene duplication is common and that transcription and translation are part of the housekeeping functions of all cells, the program simulates the process of evolution of new binding sites from scratch.

It is clear that if the process in nature started with a completely random genome, that no binding sites could be evolved using the process described in the **ev** program because the locations of the binding sites would be unspecified. However, we might argue that in fact the binding site locations are fixed because they are the positions immediately before valid coding regions to define some functional protein.

There are two problems here, which mean that it cannot be said that the **ev** simulation represents this case (of a pre-existing functional protein).

The first objection is that the program makes no attempt to preserve any information in the exon regions. Indeed, these regions are required to evolve as well as the binding site sequences, in order to avoid false recognitions. To illustrate this situation, we performed a modified **ev** type simulation, where coding regions were specified, and at the outset, information was introduced into the sequences. In order to do this, we considered each coding region to be divided into “codons” of three nucleotides, but set it up so that one of 8 of the 64 possible values (initially randomly chosen), were selected for each simulated codon. Thus, the initial information content is $\log_2(64/8) = 3$ bits of information per codon. We constructed a genome of 605 nucleotides, where the coding regions consisted of 24 nucleotides, and the binding sites of 6 nucleotides. The rest of the genome consisted of the recognizer gene. During the simulation, we monitored the information content of the binding sites, and of the coding regions (ignoring for simplicity the small sample error correction used by Schneider; which does not affect the general pattern of the results). The results are shown in Figure 4.

Not surprisingly (because the evaluation function of the simulation does not allow for it), the information content starts at 3 bits per codon, and rapidly decays, reaching its equilibrium level before any significant gain in information has arisen in the binding sites.

The second reason why **ev** does not represent this situation has to do with the information calculations. In order for the Shannon information content at the binding sites to

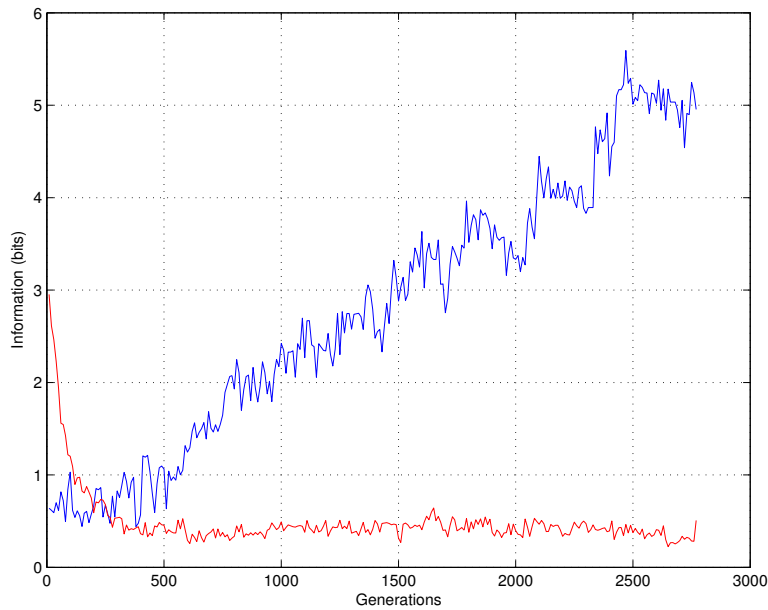


Figure 4: Ev-type simulation showing information content at binding sites (blue) and the information per codon at simulated coding regions, (red) of the best individual. At the end of the simulation, a creature which makes no mistakes has evolved, but the information in the coding regions has disappeared.

be equal to $R_{frequency}$, there must in fact be a uniform random distribution of the input points given to the perceptron, which is a hyperplane that divides the input (hyper)space into two regions, one with $1/16^{\text{th}}$ of the total volume. But the problem here is that if the distribution is uniform, then when the genome is considered *as a whole*, there will in fact be *no information at all*, because a uniform distribution of points will give the highest value of entropy possible. If there is pre-existing information in the genome, which defines where the binding sites are (just the location of the intron-ends), then we should not expect a uniform distribution of points, and hence the information content would not be expected to be equal to $R_{frequency}$.

The conclusion is inescapable. The reason the information arose at the simulated binding sites is *because it was put in right at the beginning*. There was not just a “completely random genome” at the beginning. There was a random genome plus an externally supplied specification of where the sites were supposed to be².

3 Information calculations

So far, we have not discussed the methods used to compute information content in the Schneider simulation. For this, Schneider uses the established methods of information theory [Shannon, 1948]. However, although we do not question the basic principles of Shannon’s information theory, we believe that they have been incorrectly implemented in **ev**, both in the calculation of the information content for a single location in the binding

²Given that one of the objectives of the paper is to answer creationist issues concerning how life gains information in an evolutionary process, it is ironic that this specification gets set up in a routine called **creation**, but there really needs to be a much clearer explanation of where this specification is supposed to have come from if the claims of the paper are to be taken seriously.

site, and in the case for a complete binding site. In doing so, two key assumptions have been overlooked, which we shall examine. These two oversights combine fortuitously to give the “expected” result that $R_{sequence} \approx R_{frequency}$, but as we shall see, this is not expected to be the case for the perceptron recognizer system used in the simulation. First we shall review the basic method of calculating $R_{sequence}$.

3.1 Basic method for calculating $R_{sequence}$

The fundamental idea behind the calculation of information content is reduction in uncertainty (or entropy). As an illustration, consider a game between two players A and B where A thinks up a number, say between 0 and 255, and B attempts to guess the value of the number. Given a straight guess, B has only a $1/256$ chance of getting the right answer. However, if B is allowed to ask a series of Yes/No questions about the value of the number, then the value of A 's number can be known with absolute certainty after 8 Yes/No questions, by the familiar procedure of “binary chopping”, where the first question is something like “Is the number greater than 127?”, which will determine which half of the range it is in. Further questions of the same kind will halve the available possibilities each time, and after 8 questions, the range will be only of size one and the number known with probability 1. As each question halves B 's uncertainty about the value of the number (or doubles the probability that a straight guess will be right), we say that B gains 1 *bit* of information with each question, and by the end, 8 bits of information have been gained. To represent a number in the range 0-255 in binary requires 8 bits, and the answer to each question reveals in succession the identity of each of the bits in the binary representation of the number, starting at the most significant bit.

Now let us suppose that A and B play this game 1,000 times, each time with A thinking of the number and B deducing the identity of the number by a series of Yes/No questions. But suppose in addition, B records for the sake of interest the value of the number supplied by A . Suppose that after 1,000 repetitions, B notices that actually A has only supplied 16 distinct values out of the possible 256. Armed with this information (and assuming that A is not going to change the number picking strategy arbitrarily after 1,000 games), B can now get the identity of the number after only 4 questions. All that is needed is to divide the range into 16 partitions, each of which contains only one of the 16 numbers. B 's questions are still of the form “Is it greater than x ?”, but the x 's are now chosen to be on a partition boundary, so as to halve the number of possible partitions each time. After four questions, there is one partition left, and B knows the value of the number. Hence, B 's action of recording the numbers each time has gained 4 bits of information and hence only 4 further questions are required.

In the general case, if B can infer the probability distribution of the values chosen by A , $\{p_i, i \in 0, \dots, 255\}$, then, if that distribution is significantly non-uniform, information has been gained, and B can use this information to reduce the expected number of Yes/No questions necessary to locate the value of the number. The general formula for the uncertainty, or entropy, in bits of this probability distribution is derived in [Shannon, 1948], and has the following form:

$$H = - \left(\sum_i p_i \ln p_i \right) / \ln 2 \quad (5)$$

This can also be written as:

$$H = - \sum_i p_i \log_2 p_i \quad (6)$$

It is conventional in information theory to express the uncertainty in bits, though the constant is arbitrary, and dropping the factor of $\ln(2)$ results in H being expressed in units of *nats*.

This formula shows that the greatest uncertainty results from a uniform distribution where the p_i are all the same. For the 256 numbers problem, it results in the uncertainty being 8 bits, as we expect. If, however, the probabilities are all zero, except for one value, where the probability is unity, then the uncertainty comes to zero (because $p \ln p \rightarrow 0$ as $p \rightarrow 0$).

It therefore follows that knowledge of the probability distribution of the values reduces the uncertainty with respect to a uniform distribution, and this reduction in uncertainty is the information gained in bits³. It is this principle that is used to compute the information content in the binding sites in Schneider's simulation. Here, instead of 256 values, there are only four values possible at each location L , namely A, C, G and T . Hence if we knew the probabilities of the four bases at position L in the binding site, (p_A, p_C, p_G, p_T) , then we could compute the uncertainty at that location by using equation (6). However, a subtlety arises in that we don't know the exact probabilities at each location, we can only estimate them by the *frequencies*, in other words the number of occurrences of each base (N_A, N_C, N_G, N_T) divided by the total number of binding sites, denoted γ in the paper. Hence our estimated uncertainty is given by the formula:

$$H = - \sum_{B=A,C,G,T} \left(\frac{N_B}{\gamma} \right) \log_2 \left(\frac{N_B}{\gamma} \right) \quad (7)$$

This gives us the formula for computing the uncertainty at one location L in the binding site, from our sample of the values of location L over the γ binding sites (and in the simulation described in the paper, $\gamma = 16$).

In order to get the information content for the whole binding site, in the **ev** program, the values for each of the six locations are simply summed. Thus if we write $N_B(L)/\gamma = f_{B,L}$, for the frequency of base B at location L then the total uncertainty of the binding site is given by the expression:

$$H(L) = - \sum_{L=1}^6 \sum_{B=A}^T (f_{B,L}) \log_2 (f_{B,L}) \quad (8)$$

and the frequency table $f_{B,L}$ is the inferred representation of the joint probability distribution of all the bases in the binding site.

In order to compute the information content, one would normally subtract this from the uncertainty of a uniform distribution of bases, which would be 2 bits per base. How-

³More generally, in the modern Bayesian formalism we infer a *posterior distribution* from the data, given a *prior distribution* which reflects our prior knowledge in the absence of data, using Bayes' Theorem. The difference in entropy between the posterior and prior distributions represents the information gained. In this discussion, we treat the prior as a uniform distribution. Such a prior is termed "uninformative" because it gives no information as to the value of the data.

ever, because of the uncertainty in the probability estimate due to the small sample size, Schneider estimates the expected uncertainty in a sample size of γ by a technique described in Section 3.3, and uses this to calculate the prior uncertainty, and hence the information content. But first, we shall discuss in more detail the validity of summing the information contents at each of the bases.

3.2 $R_{sequence}$ for the whole binding site

As shown in equation (8) the total uncertainty in the binding site is obtained by summing the uncertainty calculated in each of the six base locations. What is not mentioned in [Schneider, 2000], is an important assumption that is discussed in the earlier paper on information content at binding sites [Schneider et al., 1986]. The assumption here is simply that the values of the nucleotides at the individual bases are statistically independent. What this means in practice is that the joint probability distribution of variables can be factorized as the product of the distributions of each of the variables. If we represent the base value at position L as b_L , then this implies that:

$$p(b_1, b_2, \dots, b_6) = p_1(b_1)p_2(b_2) \dots p_6(b_6) \quad (9)$$

The implication of this is that if the probability distribution is not factorizable, or in other words, if the variables are *not* independent, then $R_{sequence}$ is not a correct measure of the information content.

To return to our example of the number guessing game, for example, it is clear that the set of 16 numbers chosen by A consistently out of 256 can be *any* set of 16, and that $R_{sequence}$ can therefore come to just about anything between 0 and 4 bits. This is illustrated in Figure 5 for the case where the 16 numbers happen to be consecutive, and the binary form of the numbers is chosen to compute the information content (i.e. only two symbols instead of four).

For a start index of 120 (i.e. a sequence from 120 to 135), the binary codes yield an $R_{sequence}$ value of zero. It is left as an exercise to the reader to verify that this is so. All that is needed is to construct a 16×8 table of the 8 bit binary representations of the numbers from 120 to 135 inclusive. This will show that in each column (bit position), there are always 4 ones and 4 zeros, meaning that $R_{sequence}(L)$ always evaluates to zero. A similar plot may be obtained for base-4 coding of the numbers, as if the numbers were represented as the nucleotides on a genome, in a similar fashion to the numerical values in the **ev** weight matrix. The base-4 representation plot is similar in form, but with different values, and it does not drop to zero at any point. Again, this may easily be constructed from the first table, by combining pairs of bits to make a value from 0 to 3. Then it will be found that there are not equal frequencies of the four values in each column. What is this telling us? Simply that $R_{sequence}$ is not necessarily a good representation of the information content (the real Shannon information content is always 4 bits because 256 possibilities were reduced to 16 by B 's observations over a period of time). Specifically it tells us that the bit positions are *not* statistically independent and that equation (9) does not apply. It also shows us that the value of $R_{sequence}$ depends on how the information is encoded. In both cases, the actual information content was the same, but the different encoding schemes (binary, or base-4) led to markedly different values for $R_{sequence}$.

But it could be argued that this is a silly counterexample. What about the perceptron classifier used in **ev**? It turns out that the independence assumption is equally invalid

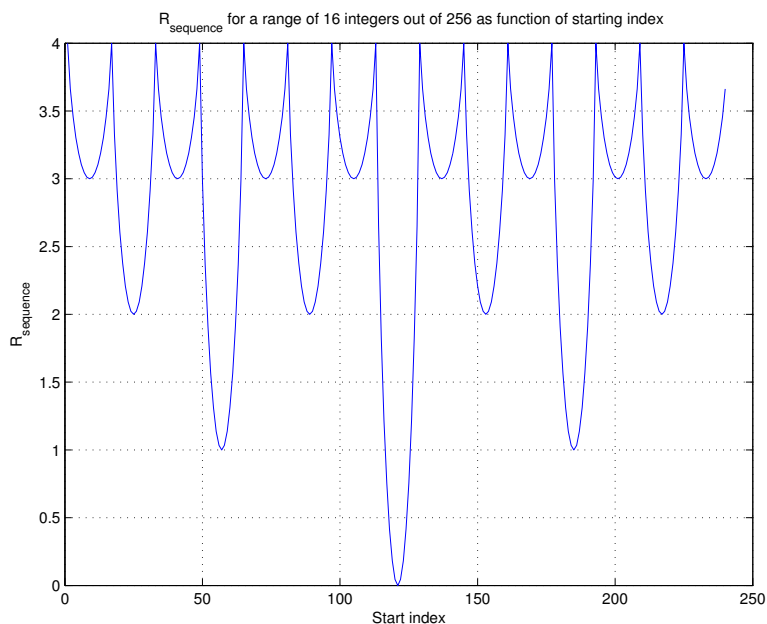


Figure 5: $R_{sequence}$ plotted using the binary codes for a set of 16 consecutive numbers out of 256, as a function of the starting index.

for this type of classifier. In order to understand why this is so we need to consider the geometry of how the perceptron works. As stated earlier, a perceptron with N inputs has an N -dimensional weight vector \mathbf{w} that defines an $N - 1$ dimensional separating hyperplane, where one side of the hyperplane is all one class (e.g. binding site) and the other side is all the other class (i.e. non-binding site). The perceptron in the \mathbf{ev} simulation has 24 binary inputs, and the entire data space is a sparse lattice on a 24 dimensional hypercube. Since this is difficult to visualize, we shall consider the case where the input space has only 2 dimensions, each of which is a number from 0 to 63 (which can be derived from 3 bases, giving 6 bits each). In this case the separating plane is a straight line. We show a typical case in Figure 6.

This example was constructed artificially by generating a random weight vector \mathbf{w} , and calculating the dot product $\mathbf{w} \cdot \mathbf{x}$ for all 4096 points on the discretized 2-d grid. The threshold value was then chosen so that 256 points were below the threshold value, so the area of the “recognized” region was 1/16th of the total area. Information content should therefore once again be 4 bits. However, if we calculate the value of $R_{sequence}$ for all 256 recognized points, we find that it is in fact only 3.45 bits⁴. This calculation was performed by computing the frequency table $f_{B,L}$ for all 256 points that are recognized by the classifier, and applying the standard formula. Each (x, y) point was considered as a sequence of six bases, each of which has four values, giving $x = 16b_1 + 4b_2 + b_3$ and $y = 16b_4 + 4b_5 + b_6$.

What happened to the extra information? The answer lies in the fact that the variables (the x and y coordinates) are not independent, and that the frequency table estimate of the probability distribution $f_{B,L}$ is inappropriate to represent the joint probability distribution of the region. It is quite clear from the figure that the distribution of each of the variables depends on the value of the other; e.g. if $x = 17$ then there is only one possible value of y being zero. By contrast, when $x = 10$, then y can take any value from 0 to 11.

⁴Note we have ignored the small sample correction here, which will be negligible for 256 samples

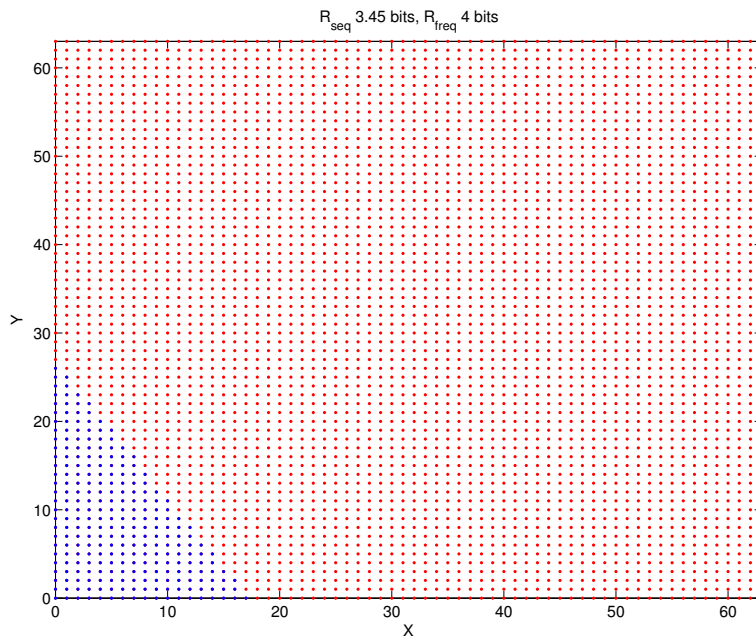


Figure 6: Two dimensional perceptron example. The linear decision boundary produces two classes; the binding sites (blue) and the non-binding sites (red). There are 256 points in the binding sites class.

We can find an analytic form for the implied probability if the information content is the sum of values of $R_{sequence}(L)$, because x is given by the first three bases, and y by the last three, and hence:

$$\begin{aligned} \sum_L R_{sequence}(L) &= \left\{ \sum_{i=1}^3 R_{sequence}(i) \right\} + \left\{ \sum_{i=4}^6 R_{sequence}(i) \right\} \\ &= R_{sequence}(x) + R_{sequence}(y) \end{aligned} \quad (10)$$

implying that the probability distribution would be $p(x)p(y)$. If the decision boundary crosses the x axis at x_0 and the y axis at y_0 , and we take the input space to be the unit square, then we have $p(x) \propto y_0(1 - x/x_0)$ and $p(y) \propto x_0(1 - y/y_0)$, resulting in an implied joint probability distribution by taking the product of the two, of:

$$p(x)p(y) \propto x_0y_0 - x_0y - y_0x + xy \quad (11)$$

which is illustrated as a surface plot in Figure 7, where $x_0 = y_0 = 1$ and the function has been plotted on a 20×20 discretized grid on the unit square.

This is evidently not correct, as it gives finite values for the probability at points that are outside the decision boundary. The true probability distribution is uniform, inside the triangle $(0, 0), (x_0, 0), (0, y_0)$, and zero outside it. This proves that $p(x, y) \neq p(x)p(y)$ and hence the true information content is not given by summing the information content in each location of the bases.

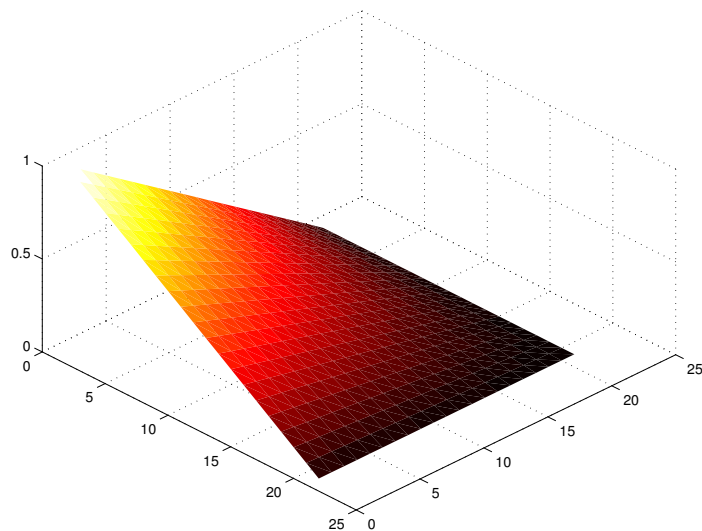


Figure 7: Surface plot of the function $p(x)p(y)$.

Hence the frequency table is an inadequate representation of the probability distribution of the binding site acceptance region of input space, for the perceptron recognizer, and so we cannot just sum up the information content at each base.

Can we ever get the situation where $R_{sequence} = R_{frequency}$, as we did in the 16 numbers case? Yes, we can, in the case where the decision plane is parallel to one of the axes. This is shown in Figure 8.

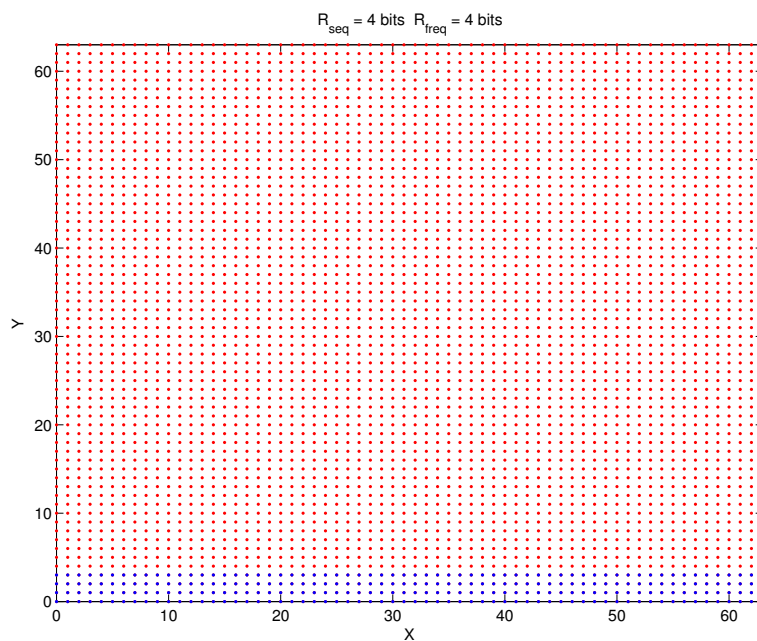


Figure 8: Two dimensional perceptron example, where $R_{sequence} = R_{frequency}$. This can only occur where one the decision boundary is parallel to one of the axes.

Here, the variables are uncoupled; because of the rectangular geometry of the acceptance region, we note that the allowed range of values of y is unaffected by the value of x , and vice versa. We also note here that all the information is carried by just one variable, namely y , which has values only in the range $(0, \dots, 3)$ in the blue “recognized” region, giving rise to $\log_2(64/4) = 4$ bits of information, whereas x is unrestricted and hence has zero contribution to the information.

This argument can be extended to higher dimensions. If there are three dimensions, then we shall only have $R_{sequence} = R_{frequency}$ if the decision plane is parallel to one of the faces of the cube. We have verified this in simulation, choosing random weight vectors and setting the threshold so that 256 points are classified as recognized. We found that $R_{sequence}$ came to 4 bits about 1 in every 300 attempts, and each time the corresponding weight vector happened to be very close to $(1, 0, 0)$, or $(0, 1, 0)$ or $(0, 0, 1)$, which define planes parallel to the faces of the cube. A slight deviation is allowable because of the discretization of the $16 \times 16 \times 16$ grid of points evaluated.

Hence, we have shown that for a perceptron classifier of the type used in the `ev` program, that $R_{sequence}$ will always be strictly less than $R_{frequency}$, because it is computed using an inappropriate model of the probability distribution of the points that lie in the classification region. In order to demonstrate this, we conducted an experiment using the same type of perceptron classifier as in `ev`. A 24-dimensional weight vector was constructed using a random number generator, and the dot product $\mathbf{w} \cdot \mathbf{x}$ for all 4096 possible 6-base binding site candidates was computed, generating the inputs by the same 1-of-4 coding scheme for each base. The threshold value θ was then chosen so that 256 of the points were “recognized” and the remainder were not. Note that although \mathbf{w} was chosen by generating random numbers in the range $-1, \dots, +1$, the action of the `ev` weight matrix can be accommodated in this, as both \mathbf{w} and θ can be scaled by an arbitrary constant factor without changing the result. A small MATLAB script was written to compute $R_{sequence}$ from the resultant sets of 256 points, and the experiment was repeated 1,000 times. The resultant distribution of values for $R_{sequence}$ is shown in Figure 9.

Over the 1000 runs, $R_{sequence}$ was 3.114 ± 0.137 bits. This is in fact worse than the two-dimensional case, and again confirms that the assumptions behind the simple calculation of $R_{sequence}$ are not valid for this kind of recognizer system.

However, we are left with a puzzle. The simulation reported in [Schneider, 2000] showed $R_{sequence}$ very close to 4 bits. We need to explain the discrepancy between the result Schneider obtained, and our result obtained via construction from a randomly selected weight matrix. We address this in the next section.

3.3 $R_{sequence}(L)$ for an individual location

In this section we examine the validity of the formula used for small sample correction for the information calculations in Schneider’s paper.

The general formula given for the uncertainty H of a set of probabilities $\{p_i\}$ is:

$$H = - \sum_i p_i \ln p_i \quad (12)$$

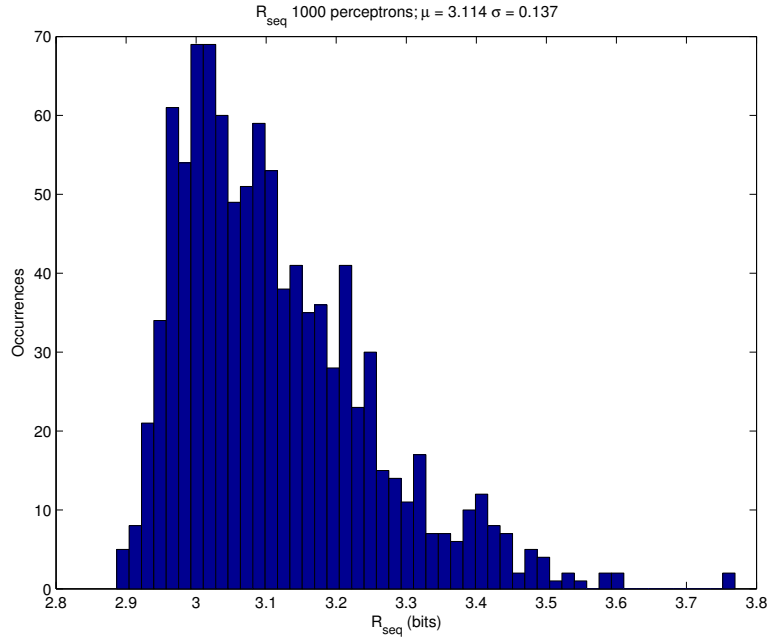


Figure 9: Distribution of values of $R_{sequence}$ obtained for 1000 randomly generated perceptrons of the same form as in the Ev program. In all cases $R_{frequency} = 4$.

where H is here expressed in *nats*. To express it in the conventional *bits* one divides by the natural logarithm of 2. This formula was derived in [Shannon, 1948] with the assumption that the probabilities p_i are already known. However, with a finite and small sample size, we do not know the probability, and can only estimate it from the sample. Hence, in order to calculate the uncertainty for a particular position L in an aligned set of N binding sites, the frequency of each base b is used to estimate the probability, to give the formula:

$$H_s = - \sum_{b=A}^T \left(\frac{N_b}{N} \right) \ln \left(\frac{N_b}{N} \right) \quad (13)$$

where s denotes a particular configuration of bases $\{N_A, N_C, N_G, N_T\}$ that sum to N .

It is recognized that this is an approximation, and a small sample correction is calculated for the information content by computing the expected value of H_{nb} for a sample size of N . This is computed by summing H_{nb} over all nb weighted by the probability of the configuration nb occurring, hence:

$$E(H_{nb}) = \sum_{\text{all } nb} p_{nb} H_{nb} \quad (14)$$

where p_{nb} is given by the multinomial theorem, where the probabilities of A, C, G, T over the whole genome are given by (p_A, p_C, p_G, p_T) :

$$p_{nb} = \frac{N!}{N_A! N_C! N_G! N_T!} p_A^{N_A} p_C^{N_C} p_G^{N_G} p_T^{N_T} \quad (15)$$

This gives what is described in [Schneider et al., 1986] as the “Exact” formula for the entropy, and the information content can then be computed as:

$$I = E(H_{nb}) - H_s \quad (16)$$

It might be thought that to compute the average would involve a combinatorially large number of computations (4^N), but it can be calculated quite efficiently for small N by noting that all permutations where there are fixed values of (N_A, N_C, N_G, N_T) have the same probability, and so we can compute it as a series of nested loops, by considering the combinations that are in “alphabetical order”, according to the following formula:

$$E(H_{nb}) = \sum_{b_1=A}^T \sum_{b_2=b_1}^T \cdots \sum_{b_N=b_{N-1}}^T p_{nb} H_{nb} \quad (17)$$

In order to avoid the overhead of a deeply nested recursion, an elegant non-recursive algorithm is given to compute this in [Schneider et al., 1986]. Either way, this results in $(N + 1)(N + 2)(N + 3)/6$ evaluations. This is not prohibitively expensive with only 16 sites, requiring only 969 calculations. However, the cost rises as the cube of N .

This formula allows for the fact that $f_b = N_b/N$ is an approximation for the true probability p_b , and the averaging of all the possibilities gives the expected value H_{nb} given random sampling of 16 sites. We expect this uncertainty to be somewhat lower because of the small sample size. In the case of 16 bases, and with equal probabilities of all four bases, the uncertainty comes to around 1.855 bits per base, as opposed to 2 bits per base with an infinite sample size, which we would need to know the probabilities exactly.

However, a key point has been overlooked in this derivation that makes a significant difference to the results. The problem is that the formula $-\sum_i (N_b/N) \ln(N_b/N)$ is itself only an approximation for large values of N when Stirling’s formula:

$$\ln N! \approx N \ln N - N \quad (18)$$

can be used. The derivation given here loosely follows [Bishop, 1995, pps 240–241]. We consider N measurements and represent the probability density function as a histogram of discrete bins. (In the case of nucleotides, we have 4 bins). We shall define the number of measurements that occupy bin i to be N_i . The entropy (or uncertainty; the two terms can be used interchangeably) is defined as (a constant times) the logarithm of the *multiplicity*, which is the number of ways of arranging the system. If there are N objects, then there are N ways of choosing the first object, $N - 1$ ways for the second, and so forth, giving $N!$ ways in total. However, rearrangements that simply re-order the items in one bin are redundant, and should not be counted. Therefore for every bin i , we must divide by $N_i!$, giving the formula for the multiplicity W to be:

$$W = \frac{N!}{\prod_i N_i!} \quad (19)$$

Therefore, choosing the arbitrary constant to be $(1/N)$ so that the uncertainty is given as the average value per sample, the exact formula for entropy with a small sample of N objects is given by:

$$H = \frac{1}{N} \left\{ \ln N! - \sum_i \ln N_i! \right\} \quad (20)$$

Note that this formula is straightforward to compute for relatively small values on N , as $\ln N! = \sum_{i=1}^N \ln i$, which can be stored as a table. In fact such a table is already stored in the `ev.p` program, as the value for p_{nb} of equation (15) is computed by taking logs, summing and then exponentiating the result.

In order to obtain the general formula for entropy, we have to take the limiting case as $N \rightarrow \infty$, and make use of Stirling's approximation, which gives $\ln N! \approx N \ln N - N$ for large N . Hence, substituting this formula for all the terms involving the log of a factorial in equation (20), gives:

$$H \approx \frac{1}{N} \left\{ N \ln N - N - \sum_i N_i \ln N_i + \sum_i N_i \right\}$$

Note that this is already an approximate formula for H before we have taken into account the approximation of the probability by the frequency. We denote this approximate value H_{approx} .

Now, noting that $\sum_i N_i = N$, the second and fourth terms in the bracketed expression cancel:

$$H_{approx} = \frac{1}{N} \left\{ N \ln N - \sum_i N_i \ln N_i \right\}$$

We now substitute $\sum_i N_i$ for N to give

$$H_{approx} = \frac{1}{N} \left\{ \sum_i N_i \ln N - \sum_i N_i \ln N_i \right\}$$

and rearranging terms, and combining the difference of logs as the log of a quotient, we have:

$$H_{approx} = - \sum_i \left(\frac{N_i}{N} \right) \ln \left(\frac{N_i}{N} \right) \quad (21)$$

Finally a *second approximating step* is required in order to arrive at the $-\sum_i p_i \ln p_i$ formula, which is the limit as $N \rightarrow \infty$, because the frequency will approach the true probability for an infinite sample size.

$$H_{approx} \approx - \sum_i p_i \ln p_i \quad (22)$$

Taking into account the $(1/N)$ constant factor, the difference between the Stirling approximation and the exact formula is shown in Figure 10:

As can be seen, the difference is significant at $N = 16$ (0.144 nats or 0.208 bits), falling to 0.04 bits at $N = 100$ and 0.026 bits at $N = 200$. If we compute $E(H_{nb})$ for the case

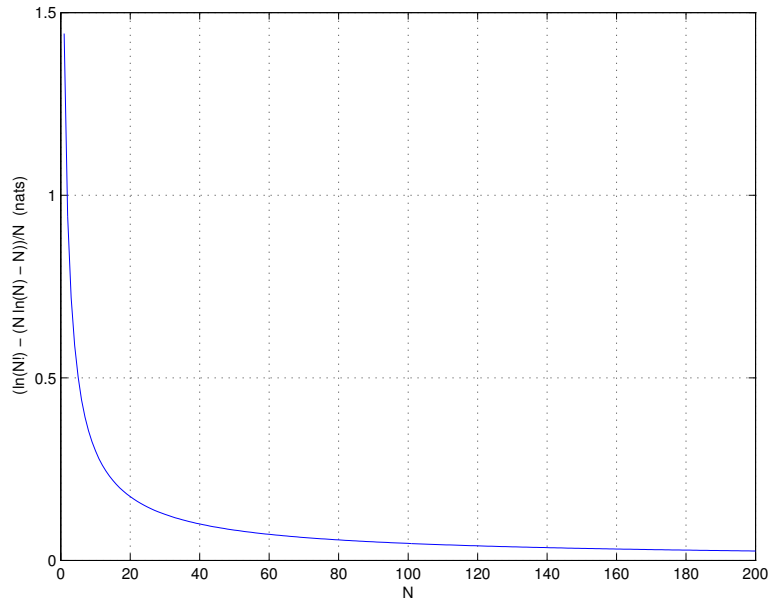


Figure 10: Difference between the exact value of $\ln(N!)/N$ and the approximate value $(N \ln(N) - N)/N$ for different values of N .

$N = 16$, and $p_A = p_C = p_G = p_T = 0.25$, we get the value 1.495 (bits). This contrasts with the so-called “exact” formula of [Schneider et al., 1986] which gives 1.855 bits. In addition, we find that for each configuration nb the exact formula for the entropy is on average 0.191 bits lower than the approximate formula. This means that the information content computed should be $1.855 - 1.495 - 0.191 = 0.169$ bits lower for each base in the binding site, or around 1 bit lower for the site. Hence the original estimate of around 4 bits per site should in fact be revised down to 3 bits.

We suggest therefore that the arrival at the figure of 4 bits per site, which is equal to $R_{frequency}$ as required, is in fact fortuitous. We demonstrate that this is the case in the simulation results presented in the next section.

4 Simulation Results

In Figure 11, we show the results of an extended set of simulations using Schneider’s program `ev.p`⁵.

Each data point on the graph was the average value of $R_{sequence} - R_{frequency}$ over 50000 generations after an organism committing no errors had evolved. The value of $R_{sequence}$ was sampled every 10 generations. The values calculated according to Schneider’s original formula are plotted in blue, and those computed with the exact formula of equation (20) are plotted in red. Ten simulations, each with a different random number seed, were performed for each value of γ , the number of sites per “creature”. All simulations were

⁵`ev.p` version 3.70 was originally downloaded from the Schneider web-site <http://www.lecb.ncifcrf.gov/~toms/delila/ev.html> and adapted to run under Borland Delphi. The current version is 3.73, but the only differences from 3.70 appear to be the removal of a redundant feature, and one or two speed optimizations, which will not affect the results produced by the program. On a 2.4 GHz Pentium IV laptop, the set of 200 runs described here was in an overnight run, so ultimate speed was not a problem.

performed with a recognizer width of 6 bases, and a weight matrix precision of 6 bases per weight. This was increased from the original value of 5 for the simulation reported in Schneider's paper to compensate for any possible artifacts due to lack of available precision in the weights matrix. As more of the genome is now taken up with the recognizer gene, we have compensated for it by increasing the overall length of the genome by 25 (the number of weights plus the threshold) to 281.

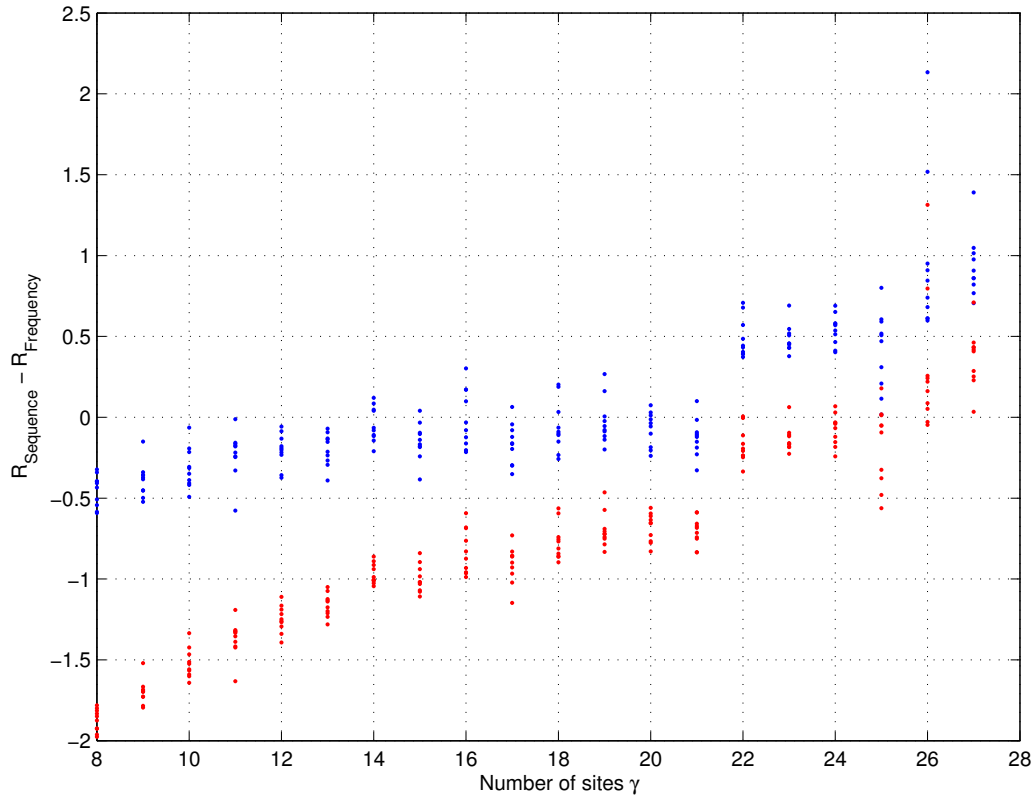


Figure 11: $R_{sequence} - R_{frequency}$ plotted for 200 runs of Schneider's Pascal program for varying numbers of sites per organism. 10 runs were performed using a different random number seed each time, for each number of sites. The plotted result is the average value of $R_{sequence}$ for 50000 generations after a zero mistakes creature had evolved (every 10th generation sampled). Blue points correspond to $R_{sequence}$ computed via Schneider's original formula, and red points give $R_{sequence}$ computed according to the new combinatorial formulae.

Note that according to the thesis of [Schneider, 2000], all the values for $R_{sequence} - R_{frequency}$ should be clustered around the zero mark as what the program is supposed to show is that $R_{sequence}$ evolves up to $R_{frequency}$.

The graph of results shows that this is not the case, and that there is a clear trend from $R_{sequence}$ being lower than expected at the least number of bases, and increasing thereafter. There is a sudden jump in the trend at $\gamma = 22$, which is almost certainly due to the fact that this is the point where the binding sites begin to overlap. It is notable that with the original program's formula for $R_{sequence}$ the best agreement with the approximate relation $R_{sequence} \approx R_{frequency}$ seems to occur at around $\gamma = 16$, which is the result reported in the paper, and the agreement is less good elsewhere, even when the sites do not overlap. However, using the exact formula (20), results are uniformly lower than for the original formula used by Schneider. The average value when $\gamma = 16$ actually comes to around

0.15 bits higher than in the simulations of the previous section. We have not investigated this discrepancy in detail, but the reason for it, and the general trend may be due to interaction between the sites. The way the program decides where to put the sites is to determine a region in which they can be placed. This explicitly excludes the recognizer gene region, and also leaves a gap of 6 bases after it (6 being the window size for the binding site). The reason for the gap, according to the comments in the source code for `ev.p`, is to assure that the first site is not constrained by the recognizer gene (i.e. no inputs to the perceptron will overlap the recognizer gene and the first site). This leaves only around 125 sites to place the binding sites. If there are 16 binding sites, then the spacing between their starting locations comes to 8 nucleotides, and hence in between two of the sites, there are in fact only 2 nucleotides that do not belong to a binding site⁶. As the recognizer scans a location starting at position 5 in a site, it will have a window of 6 nucleotides consisting of the last two bases in one site, two non-binding-site bases, and the first two from the next site. This may constrain the allowable combinations for the bases in a binding site in a way that was not modelled in the simulation of the last section. The extra constraints could give rise to the additional information found.

In summary, these results, which were obtained from Schneider's original program, show that when the number of sites is varied over a region, $R_{sequence}$ does *not* in general come to be approximately equal to $R_{frequency}$, even given the original calculation method, which fails to take into account Stirling's approximation. The reason for the apparent agreement in the result presented in [Schneider, 2000] is because of two effects:

1. The perceptron, as we showed, is an inappropriate recognizer model to use, as it will violate the independence assumption of the bases. Hence $R_{sequence}$ should be in fact lower than $R_{frequency}$.
2. The neglect of the Stirling approximation in the formula for $R_{sequence}(L)$ at one location, leads to an inflated estimate for the information.

It is purely fortuitous that the two effects happen to cancel each other out when there are 16 binding sites.

5 Discussion

In this paper, we have analyzed the simulation of [Schneider, 2000], and consider it to be flawed with respect to the claims made of it. Our three chief objections are as follows.

1. The information that arises has to be pre-specified in the `sitelocations` array.
2. $R_{sequence}$ will not in general be equal to $R_{frequency}$, unless strict conditions are applied to the probability distribution. These conditions do not apply to the perceptron in `ev` and hence $R_{sequence}$ is less than $R_{frequency}$.
3. The reason $R_{sequence}$ appears to be the same as $R_{frequency}$ is because it has been calculated incorrectly for the small sample sizes.

⁶In this simulation the program option for equally spaced sites was chosen. An alternative option would give randomly spaced sites, but the average gap between the end of one site and the beginning of another would still be 2 bases.

The simulation does *not* demonstrate the evolution of information from a “completely random genome” (Schneider’s words), with no further intervention, because it is a form of supervised learning, given a simple form of neural network (the perceptron), and a *fixed, pre-specified target*, contained, in the program in the array `sitelocations`. The information content of this array, which specifies 16 numbers out of 256, is precisely equal to the information that is supposed to have evolved from scratch, paid for by the deaths of half the population at each generation.

As we discussed, this means that the simulation falls foul of the same criticism that Richard Dawkins gave of his “Weasel” model in “The Blind Watchmaker”, namely that the model required a long term goal to be specified, which would not be available in nature.

We discussed how the Schneider model differs from the Dawkins model by having a separate *biochemical target*, which may have many outcomes, which is different from the *mathematical target* used in the program. However, this does not gain anything significant, as the mathematical target must be specified at the outset. We illustrated this with a simple extension to the Dawkins simulation, using Vignere encoding.

So the `ev` model judges the progeny according to how many binding sites of pre-specified location are correctly recognized, and how few non-binding sites are mistakenly recognized. This also is a long term idealized goal, and hence, according to Dawkins, is misleading in that evolution has no long term goal.

In the case of a spliced protein where there are multiple exons, they must all be located correctly for the protein to be expressed; two out of three being located correctly is not an improvement on one out of three, because the protein will only be expressed if all three are located. In this situation, the long-term goal would not be visible at the start; there would be no smooth path of improvement from the random start to the case where all three were located correctly.

In a simulation we performed early in this work, an experiment was performed to investigate the effect of this; of the 16 binding sites, 3 groups of 3 were specified that had to be all recognized to gain a selective advantage. The remaining 7 sites were allowed to be recognized singly. The effect of this was to increase the time to solution to 80,000 generations. The first set of three were all correctly located after 44,000 generations. No further experiments were performed on this because of the prohibitive amounts of CPU time that would have been required; we expect the time to rise exponentially with the number of sites that have to be simultaneously located. Although this was not part of the main thrust of this work, this answers the point made late in Schneider’s paper that the simulation demonstrates “irreducible complexity” by forming a “Roman Arch” consisting of the recognizer and the binding sites, both of which have to be in place for the organism to survive. It is presumed that the “scaffolding” needed to hold up the Roman Arch in the simulation is the `sitelocations` array. While this is undoubtedly true, the Roman Arch consists of only two components. Where it has to consist of N components, then the assembly of all of them can only occur via a fortuitous sequence of neutral mutations (neutral because no selective advantage can be obtained till all N “stones” in the arch are assembled). The expected number of generations for this to occur rises exponentially with N . Since the case for $N = 3$ increased the time from a several hundred to several tens of thousands of generations, it is clear this cannot be extended very far.

Secondly, the attempt at demonstrating that $R_{sequence}$ evolves to be approximately equal to $R_{frequency}$ is also flawed on two accounts. As the simulations in Section 3.2 have demonstrated, the perceptron model for the recognizer violates the independence assumption of the bases in the site, and hence $R_{sequence}$ should not give the true information

content, but will in fact be less than it. This is because the implied probability distribution is inappropriate, and will assign a finite probability to points outside the recognition region. The fact that in the simulation, the expected result was obtained is a coincidence, caused by the incorrect formula for $R_{sequence}$ for a single base and a small sample size.

This result negates the following claim in [Schneider, 2000]:

A twos complement weight matrix was used to store the recognizer in the genome. At first it may seem that this is insufficient to simulate the complex process of transcription, translation, protein folding and DNA sequence recognition found in cells. However the success of the simulation, as shown below, demonstrates that the form of the genetic apparatus does not affect the computed information measures.

The findings of this paper show that the form of the recognizer definitely *does* effect the information calculation, because it affects the joint probability distribution of the nucleotide bases in the recognized sites. In particular, the relation:

$$p(b_1, b_2, b_3, b_4, b_5, b_6) = p(b_1)p(b_2)p(b_3)p(b_4)p(b_5)p(b_6) \quad (23)$$

is violated by the perceptron recognizer. We have given geometric arguments as to why this is so, and demonstrated it empirically via simulation.

It should be noted that we are *not* claiming that the amount of information arising (the true Shannon Information) is not the same as $R_{frequency}$; we are only pointing out that $R_{sequence}$ does not represent the true information content for this recognizer system. The only proper way to compute the information content would be along the lines of the simulations of Section 3.2, where every possible input is classified, and the size (hyper-volume) of the binding site class is compared to that of the non-site class. However, this calculation cannot be performed in practice, with real DNA, because we do not have an exact mathematical model of the recognition mechanism. The only way it can be done is to estimate the relative volumes (given an assumed uniform distribution in input space), by counting the number of sites and bases on the genome. But to do this is to calculate precisely the value of $R_{frequency}$, and hence does not demonstrate anything useful.

Can anything positive be taken out of this discussion? We believe that something can point to possible avenues of future research. Despite the negative conclusions of this paper, we are still left with the puzzle that inspired it in the first place; the fact that *in nature* we observe, on a number of occasions, that $R_{sequence} \approx R_{frequency}$ (several examples are given in [Schneider et al., 1986]). The results of this paper point to two possible explanations of this phenomenon:

1. The recognition system used by the spliceosome induces a probability distribution that is factorizable on a base-by-base basis, as in equation (23). This then should be telling us something about the nature of the binding and the recognition surface (maybe it is a piece of RNA, for example).
2. If the above is not true, then we know that the true information content at the binding sites is *greater* than $R_{frequency}$. This would effectively rule out an evolutionary process that started with a random genome and evolved up to the observed amount, because there is no reason for it being greater than $R_{frequency}$. However, it might be

explained if the binding sites got inserted intact, via a duplication process in the first place. If this were the case, then initially the binding sites would all be identical, and the information content would be 2 bits per base. However, the recognizer might be robust enough to tolerate mutations and still have the binding capability, in which case, the information content might decay away, and might still be in the process of decay (and it would never fall below $R_{frequency}$).

It would normally be the case at this point in a discussion to point to further methods of research that could be used to determine which of the possibilities above was true; the principal idea being to suggest more sophisticated types of recognizer model, such as Multi-Layer Perceptrons [Rumelhart et al., 1986], or Hidden Markov Models [Rabiner, 1989]. However, it appears from a simple search of the relevant literature that the subject has progressed by large amounts from the simple perceptron technology at the heart of the **ev** simulation, and it is therefore puzzling to find this research being presented in a paper published as late as the year 2000. For example in [Reese et al., 1997], improved methods are given for automated splice site detection for data produced by the Human Genome Project. Reese *et. al.* use a Multi-Layer Perceptron and employ a coding scheme that takes into account sequential correlation between the letter values at successive bases. This is done by assigning a 1-of-16 code to a pair of successive nucleotides, as opposed to a 1-of-4 coding for a single base. This produced significantly more accurate detection of splice sites, and hence of exons (they report 8 percent less missed exons). Their work was based on earlier work using Hidden Markov Models [Henderson et al., 1996], which had reported strong correlations in neighbouring nucleotides at the splice site.

These two papers are a small sample of the work that has gone on since the early work of using perceptrons in [Stormo et al., 1982]. They indicate clearly that more sophisticated probabilistic models give better recognition systems. These more sophisticated models are bound to violate the independence assumption of bases that is taken for granted in computing the quantity $R_{sequence}$. For example if one nucleotide is *A* or *T* with equal probability, and the neighbouring one is *C* or *G* with equal probability, then under the independence assumption there are four equally likely possibilities for the two nucleotides, namely (*A, C*), (*A, G*), (*T, C*) and (*T, G*). However, a Hidden Markov Model, might well show that *A* is always followed by *C* and *T* always by *G*. This will halve the number of possibilities, meaning that the information gained by using this model is 1 bit more than the simplistic assumptions behind the calculation of $R_{sequence}$. Therefore we suspect that the true information content of binding may in fact be higher than $R_{frequency}$, and that something much more sophisticated than **ev** is going to be needed to demonstrate how this information can arise in an evolutionary process.

Acknowledgements

I would like to thank my colleague Dr. W. Worraker helpful discussions during the writing of this paper, and also Dr. P. Rüst for valuable insights on the biological matters being discussed, and for a valuable discussion on what constitutes a pre-specified target, which has helped to clarify some issues. Thanks also to John Bracht and Micah Sparaccio for their comments.

Appendices

A Vignere Encoded Text Evolution Program

We present here a result from one run of the MATLAB program we wrote to perform the Vignere Encoded Text Evolution discussed in the paper. First we give a simple example of how the text would be decoded. Vignere encoding is a simple extension to Caesar shift encoding, where each letter in a text is shifted through a fixed number of letters (with a “wrap round” occurring at the letter Z). In our simulation, we treat the space as a 27th “letter”. In the Vignere cipher, a different shift occurs on each letter in the encoded text, with the shift repeating in a cycle according the length of the encryption key. In the program, we make the encryption key a string of letters (or spaces) and the shift is given by the ordinal position of the letter in the alphabet (or by zero for a space). This is illustrated in the example below, where the encryption key is the string “ABC”, and we adopt the convention that a shift of 1 position from ‘Z’ produces a space:

Input	F	O	R	T	Y	T	W	O	
Key	A	B	C	A	B	C	A	B	C
Shift	1	2	3	1	2	3	1	2	3
Encoded Text	G	Q	U	U		C	U	Y	R

The following is a sample output from the MATLAB program. An interactive Javascript version is available at <http://www.iscid.org/vignere/vignere-text-evolution.php>

Vignere encoded text evolution demonstration

```

Target text      : THE ANSWER IS FORTY TWO
Cipher key length : 8 letters
Maximum generations : 100
Offspring each gen : 70
Print output every : 1 generations
Random number seed : 42
Initial cipher key : DJXM KVB
Initial encoded text : BCZXXUVBJXMGMDKRFG OOZT
Initial decoded text : FMWJXEQDNGJTMOFTJQXAOJO

```

Press any key to start demo

```

Gen 0001: Key: RJXM KVB  decoded text: TMWJXEQDAGJTMOFTXQXAOJO
Gen 0002: Key: REXM KVB  decoded text: THWJXEQDABJTMOFTXLXAOJO
Gen 0003: Key: REXM KVB  decoded text: THWJAEQDABJTMOFTXLXAOJO
Gen 0004: Key: REXM KVB  decoded text: THWJAEQDABJTMOFTXLXATJO
Gen 0005: Key: REXC KVB  decoded text: THW AEQDABJJMOFTXLXRTJO
Gen 0006: Key: REXC KVB  decoded text: THW AEQDABJJMOFTRLXRTJO
Gen 0007: Key: REFC KVB  decoded text: THE AEQDABSJMOFTRLFRTJO
Gen 0008: Key: REFC KVB  decoded text: THE AEQDARSJMOFTRLFRTJO
Gen 0009: Key: REFC KVB  decoded text: THE AEQDARSJMOFRRLFRTJO
Gen 0010: Key: REFC XVB  decoded text: THE ARQDARSJMAFRRLFRTWO
Gen 0011: Key: REFC XVE  decoded text: THE ARQGARSJMAFURLFRTWO

```



```

Gen 0066: Key: REFC XVE  decoded text: THE ANSQER IS FORTY TWO
Gen 0067: Key: REFC XVE  decoded text: THE ANSQER IS FORTY TWO
Gen 0068: Key: REFC XVE  decoded text: THE ANS ER IS FORTY TWO
Gen 0069: Key: REFC XVE  decoded text: THE ANSOER IS FORTY TWO
Gen 0070: Key: REFC XVE  decoded text: THE ANSMER IS FORTY TWO
Gen 0071: Key: REFC XVE  decoded text: THE ANSAER IS FORTY TWO
Gen 0072: Key: REFC XVE  decoded text: THE ANSPER IS FORTY TWO
Gen 0073: Key: REFC XVE  decoded text: THE ANSWER IS FORTY TWO

```

Run statistics

```

Target achieved after          : 73 generations
Cipher key last changed at gen : 11
Final cipher key                : REFC XVE
Final encoded text              : BCZXAQXRNMUFCKJ OSXTZT

```

B List of modifications applied to Ev.p

The following modifications were made to the program to obtain the results presented in this paper:

1. The program was ported to run under Borland Delphi. This necessitated changes to some of the file input/output handling, and to the way the files were pre-assigned. No changes to the computational routines were required.
2. The routines `dorseq` and `calcerror` were cloned to new routines `dorseqnew` and `calcerrornew`, and the new versions changed to calculate the information, and the average expected uncertainty $E(H_{nb})$ according to the exact combinatorial formulae presented in the last section, using equation (20). The output routine printed both values of $R_{sequence}$ to the output file.
3. A bug was fixed in the `creation` routine, that resulted in the incorrect placing of the last site on the genome. The initial set of runs showed similar trends to the results presented here, except that for certain numbers of binding sites, the results for $R_{sequence}$ were much lower than the general trend. (See Fig. 12).

On examination of the program, it was found that these particular cases were when it had erroneously placed the locations of the last site in a region of the genome that did not get scanned. As a result, it did not feature in the mistake count, and no characteristic pattern developed for that site. The calculation of $R_{sequence}$, however, did take the unscanned site into account, and as it did not follow the pattern of the other sites, it led to a drop in the amount of information computed. This is probably due to a sign error in the line:

```
potential := genome - site[1] + 1;
```

(line 1915 in `ev.p`). This line was intended to calculate the number of places the `creation` routine could possibly place a site, given that no sites were to be placed over the recognizer gene or within one binding site width of it. The `+1` was changed to a `-1` in order to get the intended result where all the sites were scanned.

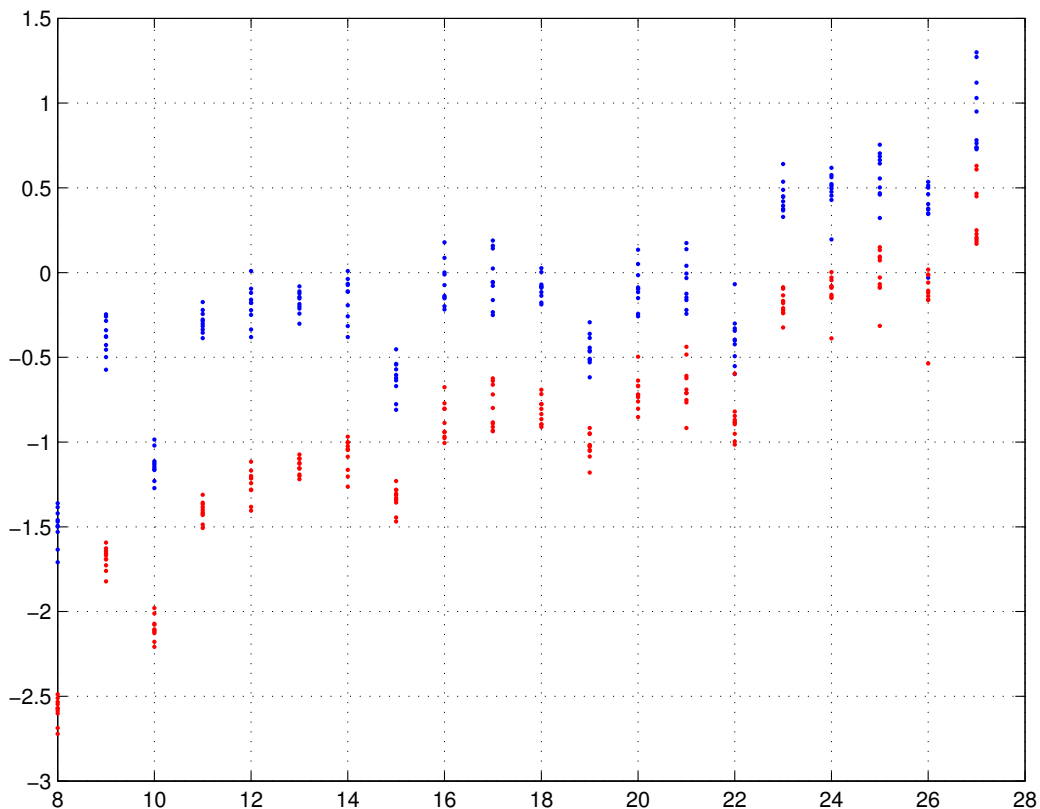


Figure 12: Initial results obtained for the main simulation. The anomalous results for 10, 15, 19 and 22 binding sites are due to a bug in the program that meant that when the last binding site was in the last possible position, it did not get scanned.

References

- Bishop, C. (1995). *Neural Networks for Pattern Recognition*. OUP, Oxford, U.K.
- Dawkins, R. (1986). *The Blind Watchmaker*. W. W. Norton and Co., New York.
- Henderson, J., Salzberg, S., and Fazman, K. (1996). Finding genes in human DNA with a Hidden Markov Model. In *Proceedings, 4th International Conference on Intelligent Systems for Molecular Biology*. AAAI press.
- Lenski, R., Ofria, C., Collier, T., and Adami, C. (1999). Genome complexity, robustness and genetic interactions in digital organisms. *Nature*, (400):661–664. <http://www.krl.caltech.edu/avida/>.
- Minsky, M. and Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. MIT Press, Cambridge, Mass.
- Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2).
- Ray, T. (1994). Evolution, complexity, entropy and artificial reality. *Physica D*, (75):239–263. <http://www.hip.atr.co.jp/~ray/pubs/oji/ojih.html>.
- Reese, M. G., Eeckman, F. H., Kulp, D., and Haussler, D. (1997). Improved splice site detection in genie. *Journal of Computational Biology*, 4(3):311–324.

- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.
- Schneider, T. D. (2000). Evolution of biological information. *Nucleic Acids Res.*, 28:2794–2799. <http://www.lecb.ncifcrf.gov/~toms/paper/ev/>.
- Schneider, T. D., Stormo, G. D., Gold, L., and Ehrenfeucht, A. (1986). Information content of binding sites on nucleotide sequences. *J. Mol. Biol.*, 188:415–431. <http://www.lecb.ncifcrf.gov/~toms/paper/schneider1986/>.
- Shaner, M. C., Blair, I. M., and Schneider, T. D. (1993). Sequence logos: A powerful, yet simple, tool. In Mudge, T. N., Milutinovic, V., and Hunter, L., editors, *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences, Volume 1: Architecture and Biotechnology Computing*, pages 813–821, Los Alamitos, CA. IEEE Computer Society Press. <http://www.lecb.ncifcrf.gov/~toms/paper/hawaii/>.
- Shannon, C. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423 and 623–656.
- Stormo, G. D., Schneider, T. D., Gold, L., and Ehrenfeucht, A. (1982). Use of the ‘Perceptron’ algorithm to distinguish translational initiation sites in *E. coli*. *Nucleic Acids Res.*, 10:2997–3011.