Bulletin of Mathematical Biology Vol. 52, No. 3, pp. 431–453, 1990.
Printed in Great Britain.

0092–8240/90$3.00+0.00
Pergamon Press plc
© 1990 Society for Mathematical Biology

MINIMUM MESSAGE LENGTH ENCODING AND THE COMPARISON OF MACROMOLECULES

■ L. ALLISON and C.N. YEE
Department of Computer Science
Monash University,
Australia 3168

(*E.mail:uucp: lloyd@moncsbruce.oz*)

A method of inductive inference known as *minimum message length encoding* is applied to string comparison in molecular biology. The question of whether or not two strings are related and, if so, of how they are related and the problem of finding a good theory of string mutation are treated as inductive inference problems. The method allows the posterior odds-ratio of two string alignments or of two models of string mutation to be computed. The connection between models of mutation and existing string alignment algorithms is made explicit. A fast minimum message length alignment algorithm is also described.

1. Introduction. A theorem in mathematics is something that is true given certain conditions or antecedants. In contrast, a *theory* in physics or chemistry is a hypothesis which explains some aspect of a system more or less well. Theories can only be compared as being better or worse. When doing this informally, terms such as "accuracy", "parsimony" and "simplicity" are often used and Occam's razor may be invoked. We hold the view that such arguments can be formalized in one objective criteria based on minimum message length (MML) encoding. This was proposed as a basis for inductive inference by Solomonoff (1964). Wallace and Boulton (1968) applied MML encoding to spaces of continuous variables and produced a practical classification program. A good discussion of MML encoding and applications is given by Georgeff and Wallace (1984). The work described here applies MML encoding to problems of string comparison in molecular biology. We treat string alignments and models of string mutation as theories to be inferred from data. It is obviously important to have an indication of the posterior probability or the plausibility of the theories inferred. Since the method is novel in this context, an informal description of MML encoding is given below. The relevant concepts are formally defined in appropriate sections of the paper. Wallace and Freeman (1987) describe the mathematical foundations of MML encoding in its fullest generality.

Supported by Australian Research Council grant A48830856.

For illustrative purposes, consider a *transmitter* who must send a message describing a system state (data) to a *receiver*. The transmitter and receiver can previously have agreed on any theory of the system, algorithm, code or code book. The transmitter–receiver paradigm keeps us honest because anything not previously agreed on must be sent in the message. If a theory makes accurate predictions of some parameter value, there is a small expected difference between the predicted and actual value. If the transmitter and receiver share the theory, they can both construct the prediction. There is no need to transmit the actual value in its entirety; instead an efficient (short) code can be devised for the difference. In general a better theory will lead to a more efficient code and to shorter messages on average. An optimal theory will lead to the strict MML encoding of the data; this lower limit is often called the *entropy* of the data; see for example Boulton and Wallace (1969).

Work related to MML encoding concerns the definition of a *random string* (in the case of random number read string of digits). Most definitions of randomness depend on an application and may call for equal frequencies of character values, independence of adjacent characters and so on. The *Kolmogorov complexity* of a finite string, as discussed by Kolmogorov (1965), Chaitin (1966) and more recently by others such as Li and Vitanyi (1988), is the one definition that stands out as being independent of any application. The Kolmogorov complexity of a string is defined to be the size of the smallest program that will generate the string and then stop when run on a universal Turing machine, or general purpose computer. The choice of a particular model of computer makes only a constant factor difference in this complexity and can be ignored. There is at worst a straight-line program, having the same length as the string, which writes the first character, then the second and so on. On the other hand if there is any pattern or structure in the string, a subroutine can be written to reproduce the pattern and can be called whenever needed thus shortening the program. The notion of pattern is very general and includes all common notions—repeat, reversal, syntactic structure, arithmetic progression, in fact anything that is computable. A random string is defined to be one having Kolmogorov complexity equal to its length. Unfortunately the Kolmogorov complexity is not computable for arbitrary strings, not even in principle. Programs can be enumerated in order of increasing size but the *halting problem* demonstrates that there can be no algorithm to determine if an arbitrary program for a Turing machine will halt, let alone produce a given string (see Turing, 1936).

There is significant evidence that large biological molecules are nearly random strings by the above definition, see for example Gatlin (1974) and Jimenez-Montano (1984). Certainly some short sequences such as TATA boxes recur in DNA strings and repeats, reversals and inversions are not unknown. Shepherd (1981) gives evidence for the echo of a purine-any-pyrimidine (RNY) code in coding regions. However, these effects are weak and there is no evidence as yet for any grammar with the richness of that found in natural languages.

Even if an individual string A is essentially random, it may be obviously similar to string B in some informal sense. B may also be a random string but the relationship between characters of A and B is *not* random! Many methods of comparing two strings or of calculating the edit-distance between them have been proposed to capture this idea. They have been surveyed by Smith *et al.* (1981), Sankoff and Kruskall (1983) and Waterman (1984). Each method has an associated *model* or theory of how strings are *edited* during their evolution. The model specifies how to calculate one or more optimal ways of relating two strings—how they may be *aligned* or edited one into the other. Every such alignment is also a theory—of how the two strings are related—although it is not usually elevated to this status. These theories can be compared on the basis of message lengths. (Although an editing model might not be based on alignments as usually conceived, nevertheless the term "alignment" is used to cover any way of relating two strings.)

Our transmitter now has to send two strings to the receiver. We assume that each individual string is random; this is not necessary in principle but makes calculation and explanation easier. If the strings are unrelated, the transmitter can do no better than send one string and then the other string. On the other hand, if the two strings are related in some way, a shorter message will result from using a good theory of how they are related.

Informally, one often cautions against inferring too much detail from data and MML encoding can quantify this. For example, stating a single alignment is sometimes too detailed and increases the message length inappropriately (see later). Secondly, if a model has any real-valued parameters, their values must be stated in the message. MML encoding implies an optimal degree of precision (see Wallace and Freeman, 1987). Note that most models of mutation have at least one such parameter which is related to the amount or the duration of mutation.

For a fixed model of editing, two alignments of a pair of strings can be compared on the basis of the message lengths required to transmit the strings. The better alignment leads to the shorter message. The difference in message lengths can be interpreted as minus $\log_2$ of the posterior odds-ratio of the alignments. At a higher level, two models of editing can also be compared. If many pairs of strings, $\langle A_i, B_i \rangle$, can be found where A_i and B_i are known to have a common ancestor, the better model is the one which leads to the shorter messages on average.

There can be only one good model of editing that is application independent. It must be based on the size of the smallest program to edit one string into another and stop when run on a universal Turing machine. As before, results for such a theory are uncomputable because of the halting problem. Fortunately there are simpler, computable models which approximate this model and may even perform slightly better if their restrictions are realistic. Many of these models are the basis of familiar alignment and edit-distance methods mentioned above. For example, simple *character editing models* (CEM) permit atomic operations on single characters only. A character may be copied (unchanged), deleted, inserted or substituted when editing string A into string B. This forms the basis of the longest common subsequence (LCS) problem and of Seller's (1974) edit distance. The quality of a CEM or any other model is not absolute; it depends on how closely it fits the evolution of real strings. MML encoding shows how to compare theories. It does not in general show how to form new theories although heuristics may be discovered in certain cases. Any available knowledge, particularly biochemical knowledge, can and should be used to develop new models or theories which then compete with other theories.

In the following sections we make the editing model or theory behind string comparison methods explicit and argue that the methods should be compared using MML encoding. Happily, while different commonly used methods give slightly different results none performs poorly. MML coding allows tuning of parameters for a given model. It is also used to determine the plausibility of alignments and is applied to the problem of embedded strings. Finally a new alignment algorithm is derived. Most of the paper concerns DNA strings over the alphabet $\{A, C, G, T\}$ but the methods are applicable to strings over other alphabets such as the 20 amino acids or the 64 codons. Note that the algorithms decribed do *not* actually encode any messages although for simplicity we usually write as if they do; rather they compute what the message lengths would be.

Reichert *et al.* (1973), Cohen *et al.* (1975) and Wong *et al.* (1974) have applied information theory to string comparison and theirs appears to be the only closely related work. In fact they present a single editing model and give informal arguments as to why it should be good; differing models are not compared. It is now recognized in coding theory that data modelling and coding are separate activities and we take advantage of this. The coding problem has been solved by the arithmetic coding method described by Langdon (1984) and by Witten *et al.* (1987) which is optimal for a given model. Reichert *et al.* were primarily interested in direct calculation of MML alignments for their particular model. They were unable to do this efficiently and their algorithms are $O(n^4)$ or worse. Here the importance of fast algorithms is recognized and we therefore compare alignments and models that have efficient algorithms.

Information theory has also been applied in molecular biology by Jimenez-Montano (1984) to the classification of amino acids, by Staden and McLachlan

(1982) to the detection of coding regions, by Hasegawa and Yani (1975) to studies of the genetic code and by Shepherd (1981) to the search for the primeval genetic code. Smith (1969) has studied the information density of the genetic code and Smith and Waterman (1980) have extended this approach to overlapped coding regions. Such studies refine our knowledge of the properties of strings. Theories based on this knowledge can be compared on the basis of MML coding and the knowledge can and should be built into string comparison algorithms.

2. Character Editing Models (CEM). The simplest *character editing models* (CEM) are based on character operations. The atomic operations allow a character to be copied, inserted, deleted or changed. A copy is sometimes called a match and a change is sometimes called a mismatch. A *mutation* is a change, an insert or a delete operation. These models are reversible so inserts and deletes are often collectively called *indels*. It is convenient to introduce a null or blank character '$-$' to describe indels. A distance '$d(a, b)$' on characters can be interpreted as a cost on edit operations and can be extended to a distance '$D(A, B)$' on strings in the following way. Summing the costs of individual character operations gives a cost to a sequence of edit operations. The distance '$D(A, B)$' on strings is defined to be the minimum cost over all edit sequences that transform A into B. The well known *dynamic programming algorithm* calculates $D(A, B)$.

Boundary conditions:

$$D(-,-)=0$$

$$D(-, B[1..j])=D(-, B[1..j-1])+d(-, B[j])$$

$$D(A[1..i], -)=D(A[1..i-1], -)+d(A[i], -)$$

General step:

$$\begin{aligned} &D(A[1..i], B[1..j])\\ &\quad=\min(D(A[1..i-1], B[1..j-1])+d(A[i], B[j]), && \text{match/mismatch}\\ &\qquad\quad D(A[1..i-1], B[1..j]) \quad +d(A[i], -), && \text{insert in } A\\ &\qquad\quad D(A[1..i], \quad B[1..j-1])+d(-, \quad B[j])) && \text{insert in } B \end{aligned}$$

Note that '$-$' is used to denote both the null or blank character and the null or empty string and that D is a metric if d is. In practice, the algorithm uses a matrix $D[,]$ and $D[i, j]$ is written for $D(A[1..i], B[1..j])$. It is clear that the elementary costs are applied independently of position and context.

The choices made in the general step of the dynamic programming algorithm determine a path or *alignment*, denoted 'A/B', from $D[0, 0]$ to $D(|A|, |B|])$. This alignment is a way of representing a sequence of edit operations that achieves the minimum cost.

Two simple form of $d(,)$ are popular.

	match	indel	mismatch	
CEM(1, 1):	$d(a,a)=0$	$d(a,-)=d(-,a)=1$	$d(a,b)=1$	– Sellers' edit distance
CEM(1/2, 1):	$d(a,a)=0$	$d(a,-)=d(-,a)=1/2$	$d(a,b)=1$	– ~LCS

The former stems from molecular biology and the latter from computer science applications. Various sets of weights, based either on physical and chemical properties or on inferred rates of substitution, are used for strings over amino acids (see Chs 22 and 23 of the atlas by Dayhoff, 1978).

Some work is phrased in terms of "similarity" of strings rather than edit distance. A similarity measure 'S' (often normalized) can be defined in terms of a distance and *vice versa.*

$$S(A,B)=(D(A,-)+D(B,-)-D(A,B))/2$$

$$D(A,B)=D(A,-)+D(B,-)-2.S(A,B)$$

This means that computational techniques for a similarity, for example Hirschberg's (1975) space efficient technique and Allison and Dix's (1986) word-parallel technique for the LCS problem, can be applied to the corresponding edit distance and *vice versa.*

The CEM(1, 1) model above gives Sellers' (1974) metric; it attempts to minimize the number of mutations in an alignment. CEM(1/2, 1) corresponds to the longest common subsequence (LCS) similarity measure; it attempts to maximize the number of matches (or copies) in an alignment. Alignment algorithms for these two models have differing but similar objectives and often give similar alignments. A copy and a change operation each account for two characters but an indel accounts for only one character. Therefore it may be possible to reduce the number of mutations in an alignment at the price of a (slight) decrease in the number of copies or to increase the number of copies at the price of an increase in the number of mutations. A interesting pair of models, apparently untried, is CEM$(1-\delta, 1)$ and CEM$(1/2+\delta, 1)$ where $\delta<1/(|A|+|B|)$.

3. Coding Theory. There is a considerable body of work on encoding data compactly for the purposes of transmission or storage; Hamming's (1980) book gives a good introduction. For our purposes, the main result is that given characters (or events) c_i, with probabilities of occurrence p_i, an optimal code assigns a code word of length $-\log_2(p_i)$ to c_i. Concatenating code words gives an optimal code for a sequence of independently generated characters of known length. The arithmetic coding method described by Langdon (1984) and by Witten *et al.* (1987) is a way of achieving this limit. If the characters in a string are not independent, then probabilities conditional upon preceding characters are used.

For example, if the characters $\{A, C, G, T\}$ are all equally probable it is not possible to do better than encode each character in $2 = -\log_2(1/4)$ bits. If on the other hand the characters have probabilities 1/2, 1/4, 1/8 and 1/8, a Huffman code will give them code words of length 1, 2, 3 and 3 bits respectively. This code gives an expected message length of 1.75 bits per character. Note that it is optimal and that the message length is minus $\log_2$ of the probability of the event described. If strings of length 2 are considered, 'AA' is coded in 2 bits which corresponds to its probability of 1/4 and so on. When a string of arbitrary length is transmitted, its length must also be included and this is considered in the next section. In general, the MML for a string is minus $\log_2$ of the probability of the string appearing.

For finite problem spaces, the MML inference is equivalent to arguments employing Bayes' theorem. As a simple example, consider pairs of strings of length one. Assume each letter has probability 1/4. We entertain two possible theories about such a pair of strings—the null-theory that the strings are unrelated and the r-theory that they are related. If unrelated, we give a prior probability of 1/16 to each possible pair. If related, we give a higher prior probability, say 1/8, to a matching pair and a lower probability, 1/24, to each non-matching pair. Given $\langle A, A\rangle$ as data, the null-theory codes this in 4 bits, the r-theory in 3 bits. The odds-ratio is 2 in favour of the r-theory. Given $\langle A, C\rangle$ as data, the null-theory is more plausible.

The problem of optimal encoding is intimately connected with the probabilities of occurrence of the characters. If the probabilities are not known in advance, it is possible to use a *dynamic coding* method which uses the frequencies of observed characters to approximate these probabilities. The receiver only knows the characters so far transmitted and the frequencies to that point are used to adjust the code by an algorithm shared by transmitter and receiver. An assumption such as equal probability is made initially to start the process. As the message grows, the code tends towards the optimal code for the actual, but unknown, probabilities. This method is optimal for this situation.

The coding methods described are optimal if the model of the data is correct and coding research is now largely concerned with this modelling problem. For example, are the probabilities fixed, how much context influences the next character and so on.

There are several options to choose from when encoding DNA strings. The simplest assumptions is that all four DNA-characters are equally probable which implies that a two bit code is optimal. It is well known that the characters in the body of *known* strings do not quite appear equally often and a fixed code could be based on the observed frequencies. In some classes of strings the frequencies of characters are significantly skewed. When working on such a class a fixed code could be designed for it. Alternatively, a dynamic code could be used. In addition, adjacent characters are not altogether independent in coding regions and there is evidence of an ancient RNY code. Conditional probabilities could be used, perhaps dynamically, to account for this. The simplest option is taken in this paper and it is assumed that characters are independent and that all characters are equally probable. There are several reasons to do this. Firstly, it is a very good choice; our tests on the Blur sequences and a large virus indicate that it is very difficult indeed to encode DNA in much less than 2 bits per character. See Bains (1986) for the Blur sequences. Secondly, the body of strings may be biased by the current technology and interests of molecular biologists. We want to discuss string comparison as generally as possible. Lastly, the general methods described are independent of the particular choice made and simplicity makes for a minimum length paper.

4. Coding Lengths and Integers, or How Long is a Piece of String? When transmitting a string it is necessary to include its length to make the message intelligible. Some alignment methods also require the transmission of other lengths and integers. If two rival theories transmit the same length information it can be ignored in comparisons as being a constant overhead. It is also possible that such overheads become a small fraction of the total message for long strings and can then be ignored. Otherwise, to design a good code it is necessary to use a good prior probability distribution, $P(k)$, for lengths. If some particular distribution fits the body of data well, a code can be designed for the distribution, using $-\log_2(P(k))$ bits to encode a value k. The priors for lengths are an important part of a model of mutation.

For example, if the range of an integer k is $[1..n]$ and the distribution is uniform, an optimal code uses $\log_2(n)$ bits to encode k. The length of the largest known gene or the biggest chromosome might be used as an upper bound on the length of strings but it is more natural to consider unbounded distributions.

Given a geometric distribution, $P(k)=(1-p)^{k-1}p, k \geq 1$, with mean $m=1/p$, $-\log_2(P(k))=-\log_2(p)-(k-1)\log_2(1-p)$ bits are used to transmit k. Note that this is of the form $a+b\times k$ which appears again later. It is approximately $(k-1)/m+\log_2(m)$ if p is small and k is not much larger than m. If k is of the same order as m, the message length is close to $\log_2(m)$.

Given a Poisson distribution, $P(k)=e^{-m}m^k/k!$ with mean m, and using Stirling's approximation for $k!$, approximately $(m-k)\log_2(e)+k\log_2(k/m)+0.5\log_2(2\pi)+0.5\log_2(k)$ bits are used to transmit k. When $k=m$ this reduces to $0.5\log_2(2\pi)+0.5\log_2(m)$.

Reichert *et al.* (1973) favour $P(k)=1/k(k+1)$ giving a message length of $\log_2((k(k+1))$ bits or $\sim 2\log_2(k)$ bits.

Rissanen (1983) argues persuasively that a colourless or universal prior distribution for the integers should be derived from a prefix code for integers. He gives a continuous function, $r(k)$, as the length of a code word for k. Unfortunately $r(k)$ is not concave which makes it unsuitable for some of our purposes. A concave approximation, $U(k)$, giving $P(k)=2^{-U(k)}$, is used in this paper; details are given in an appendix. $U(k)$ is $O(\log_2(k))$. Note that regardless of whether Reichert's method, Rissanen's method or a method based on geometric or Poisson distributions with mean m is used, the number of bits used to transmit a value $k \simeq m$ is of the same order as $\log_2(m)$. Most DNA strings of interest are between 100 and 1000 characters long and no method has a marked advantage in this range. The distribution used is not crucial to the MML approach; it is part of a theory which is open to test given enough data.

There seems to be no widely accepted prior probability distribution for the lengths of strings of interest to molecular biologists. Because of this and unless otherwise stated, we code an integer variable $k \geq 1$ in $U(k)$ bits. If the lower limit of the variable is not 1, a constant is added as a correction before coding. In particular, a variable $i \geq 0$ is coded in $U(i+1)$ bits.

5. Coding CEM Alignments. A string comparison or alignment method is associated with a model of string evolution or string editing. The model fixes the set of edit operations and gives their prior probabilities. The model may have one or more parameters. An alignment $A|B$ embodies two strings and a theory of how they are related—how one may be edited into the other.

Example

A:	$ACGTAGT$
B:	$A\text{–}GTACT$
A; edit (A, B):	$ACGTAGT$; copy, delete, copy, copy, copy, change(C), copy
alignment:	$\langle A, A\rangle\ \langle C, -\rangle\ \langle G, G\rangle\ \langle T, T\rangle\ \langle A, A\rangle\ \langle G, C\rangle\ \langle T, T\rangle$

We might transmit this alignment in many ways, for example as A followed by instructions to edit A to B, or as a string of pairs of the form $\langle a, b\rangle$. Since all methods convey exactly the same information—two strings plus their assumed relationship—all should have the same MML. In practice it is easiest to transmit the alignment as a string of pairs. The alignment can be thought of as a program for a two-tape machine which produces A on one tape and B on the other tape.

Starting with the DNA-alphabet, $\{A, C, G, T\}$, plus the null character '$-$', the full alignment-alphabet contains 24 characters ($\langle -, -\rangle$ is not used).

$$\begin{array}{cccccc} & - & A & C & G & T \\ - & & * & * & * & * \\ A & * & = & \neq & \neq & \neq \\ C & * & \neq & = & \neq & \neq \\ G & * & \neq & \neq & = & \neq \\ T & * & \neq & \neq & \neq & = \end{array}$$

$=$ match or copy
$\neq$ mismatch or change
$*$ indel, insert or delete

CEMs imply that alignment-characters in an alignment are independently chosen. Each alignment-character must have some probability. For simplicity only, we assume that all four matches are equally probable. Similarly, all twelve mismatches are equally probable and all eight indels are equally probable. This leaves three parameters. The actual probabilities of matches, mismatches and indels are parameters of a particular CEM. In general we do not know these probabilities *a priori*. We therefore encode an alignment using a dynamic code for just three characters, match, mismatch and indel, each followed by a fixed code for extra parameters.

match(DNA character 1 . . 4)
mismatch(two different DNA characters 1 . . 12)
indel(string A or B, DNA character 1 . . 4)

$p(\text{match}) + p(\text{mismatch}) + 2 \times p(\text{indel}) = 1$

$p(\langle A, A\rangle) = p(\text{match})/4$ etc.
$p(\langle A, C\rangle) = p(\text{mismatch})/12$ etc.
$p(\langle A, -\rangle) = p(\text{indel})/8$ etc.

Note that the independence assumption of CEMs implies a prior expectation that *run lengths* of matches, mismatches and indels each have a geometric distribution. The geometric distribution originates from the run lengths of independently chosen characters.

It is clear that Seller's metric and the LCS problem are based on instances of CEMs which we call CEM(1, 1) and CEM(1½, 1). Their indel (mismatch) weights apply equally to each indel (mismatch). The weights are independent of context, implying independent choices. What the actual probabilities involved in these models are is not yet clear.

The frequency of an alignment-character and therefore the length of its code word depends on the alignment. Closely related strings lead to alignments with many matches. In the extreme case of two identical strings, a good alignment consists only of matches and effectively transmits two DNA characters for a cost very close to 1 bit per character (bpc). The alignment description can be separated into two parts. The sequence of match, mismatch, insert and delete characters form the theory proper of how the two strings are related but does not state what the strings are. Many pairs of strings could be related in the same way. The parameters form a coding of the strings, the data proper, optimized for the theory.

We may consider several theories about the relatedness of two strings. The *null-theory* is that string A and string B are unrelated. It is encoded by encoding A and then B. An *A/B-theory* is that A and B are related in some particular way as represented by an alignment that achieves the minimum edit cost. Note that the null-theory and an $A|B$-theory are exclusive but not exhaustive. There may be several alignments that have the same edit cost. The *$r0$-theory* is that A and B are related by any one of the alignments achieving the minimum edit cost; it is the union of all $A|B$-theories. Note that once both strings are transmitted, by whatever means, the receiver can then reconstruct all possible alignments using any algorithm he or she chooses. The *rn-theory*, $n \geq 0$, is that A and B are related by some alignment with edit cost at most n plus the minimum possible. The sequence $r0, r1, r2, \ldots$ consists of weaker and weaker theories. The limit, the *r-theory*, is that A and B are related in some unspecified way. It is the complement of the null-theory. Wallace (1989) has recently devised a method of encoding the r-theory which will be the subject of a future paper.

A problem with an $A|B$-theory is that it is usually too precise. For example, the strings *CCC* and *CCCC* are probably related but it is impossible to be sure of one alignment. Similarly, *CGC* and *TTATA* are probably unrelated but a classical alignment algorithm will arbitrarily choose some alignment. When these examples are embedded in longer strings the problem compounds. People interpreting such alignments are well aware of this problem but we want to deal with it algorithmically. An $r0$-theory is one step in the direction of optimal precision. We omit the details of how to code an $r0$-theory, although it roughly consists of saving one bit each time two alternative optimal paths are recognized. Using a standard coding technique, the message length of the $r0$-theory can be estimated by subtracting $\log_2$ of the number of optimal alignments from the MML of an $A|B$-theory. It is not at all clear how to code an rn-theory for $n > 0$.

The difference in the message lengths of two theories can be interpreted as minus $\log_2$ of their posterior odds ratio (see Wallace and Freeman, 1987). The point at which two theories are equally plausible can be calculated for special cases or can be estimated by Monte-Carlo experiment. Note that the null-theory must include the lengths of A and of B. An $A|B$-theory can include just one, slightly larger, length—that of $A|B$.

Some approximate results can be calculated for very long strings for which the overheads of stating lengths become a vanishing fraction of the total message costs. Assuming for example that matches have probability p in total and that mismatches, and likewise indels, have probability $(1-p)/2$ in total, the expected message length per alignment-character is given by L.

$$L = -p*\log_2(p/4) - (1-p)/2*\log_2((1-p)/2/12) - (1-p)/2*\log_2((1-p)/2/8)$$

The three terms give the contributions from matches, mismatches and indels respectively. The expected number of DNA-characters per alignment-character, d, can be calculated, noting that matches and mismatches account for two DNA-characters and indels for just one.

$$d = 2*p + 2*(1-p)/2 + (1-p)/2$$

The homology of two strings, h, is the proportion of DNA-characters in A and B that are matched.

$$h = 2*p/d$$

Solving $L/d = 2$ gives $p = 0.67$, $h = 0.73$. At this point, the null-theory and the $A|B$-theory have equal message lengths and are equally plausible. If A and$B$ have higher homology, the $A|B$-theory gives a shorter message and is more plausible.

Deken (1983) calculates bounds for the length of an LCS divided by the string length for random strings of the same length. For an alphabet of size 4 these bounds are 0.55 and 0.72. Our Monte-Carlo experiments on long strings give $h = 0.65$ as an average figure. Although LCSs are based on CEM(1/2, 1), the homology figures for CEM(1, 1) are similar and our Monte-Carlo experiments give $h = 0.59$ for random strings. One would expect the $r0$-theory to require homology a little above that of random strings for support. Two weakly related strings will have $k \gg 1$ equally plausible (weak) alignments. The $r0$-theory is the union of these alignments and in this case its MML should be $\log_2(k)$ bits less than the MML of any one of the alignments as it is not supporting one above the others. Therefore an $A|B$-theory needs homology higher than the $r0$-theory for support and the lower limit of 0.73 is reasonable. As a rough guide this may be used to judge if an alignment is more plausible than the null-theory although message length should be the final arbiter.

A simple variation on the dynamic programming algorithm was used to count the number of optimal alignments; note that it is not necessary to enumerate them. Monte-Carlo experiments show that two random strings of length 100 have 5×10^7 alignments on (geometric) average that achieve the minimum edit cost. The $\log_2$ of this figure is approximately 26 and dividing by 200 gives an extra cost of 0.13 bits per DNA-character for transmitting any one alignment rather than the $r0$-theory, their union. Using the equations above, we find that $h = 0.68$ gives a cost of 2.13 bits per DNA-character for transmitting an alignment. For homology above 0.68 an MML encoding of the $r0$-theory is shorter than for the null-theory. In other words, for homology between 0.68 and 0.73 it is possible to say confidently that the two strings are related by one of the optimal alignments. These figures also suggest that the method of coding alignments is very good if not optimal. Monte-Carlo tests were performed and the code length per character for an *r*0-theory was plotted against homology for strings of various lengths (Fig. 1). The critical figures, at which the null and *r*0 theories are equally plausible, is $h=0.85$ for strings of length 20 decreasing to $h=0.7$ for length 2000. This is consistent with the previous estimates. It must be recognized that the random strings used were generated using the same model that the coding methods are based on.

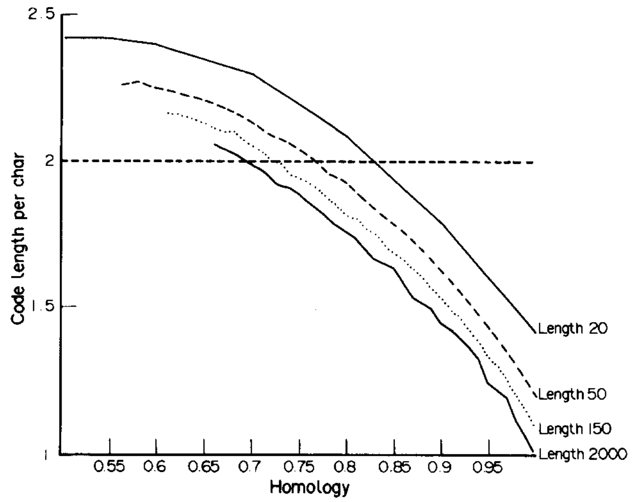

Figure 1.

It is clear that popular algorithms are doing the right sort of thing by CEM models. Given a set of alignments of two strings and provided that the ratio of indels to mismatches varies little, the alignments with a large number of matches and a small number of mutations have short encodings because they have a large bias in the frequencies of alignment-characters. This is just as well because the MML argument is confirming experience. It also explains this experience formally and can be used to interpret the plausibility of alignments and to tune parameters of alignment algorithms. In the next section the dynamic programming algorithm is modified to estimate message lengths and the problem of end segments is considered.

6. Embedded Strings. A string of interest may be embedded in a longer string and the location of the interesting part is not always known. Sellers (1980) and more recently Waterman and Eggert (1987) have addressed this problem. Here strings are considered to be made up of three segments.

$$A = A1;\ A2;\ A3$$
$$B = B1;\ B2;\ B3$$

$$e\text{-theory: } A1;\ B1;\ A2|B2;\ A3;\ B3$$
$$\text{null-theory: } A;\ B$$

We examine *e-theories* of the form that $A2$ and $B2$ are related and that the end segments, $A1$, $A3$, $B1$ and $B3$ are unrelated. An e-theory is stated by specifying the segments $A1$, $A3$, $B1$ and $B3$ and the alignment $A2|B2$. There are costs associated with encoding the lengths of the various segments, empty or not. The case when $A1$ and $A3$ are empty corresponds to searching for a short key in a long string. If $A3$ and $B1$ are empty, the strings are overlapped. If $A2|B2$ is empty, the strings are unrelated and $A3$ and $B3$ can be assumed to be empty and they need not be stated. This *null-theory* is an important special case and is not considered to be a proper e-theory.

The end segments $A1$, $B1$, $A3$ and $B3$ are unrelated and are coded as simple DNA strings at two bits per character. The alignment $A2|B2$ should be coded as described in the previous section and if two e-theories are being compared this can be done. However this does not tell us how to find a good e-theory initially. To do this, the dynamic programming algorithm is modified to find one. The weights, $d(,)$, are changed to reflect the number of bits for coding the alignment-characters. Since these costs depend on the unknown alignment, estimated values are used. These are derived from typical alignments and the calculations of the previous section.

$$\begin{array}{llll}
\text{end segments:} & d(a,-)=d(-,b)=2 & & 2.0\ bpc \\
\text{alignment:} & d(a,a) = -\log_2(0.75/4) & = 2.42, & 1.21\ bpc \\
 & d(a,b) = -\log_2(0.25/24) & = 6.58, & 3.29\ bpc \\
 & d(a,-)=d(-,b) = -\log_2(0.25/16) & = 6.00, & 3.0\ \ bpc
\end{array}$$

It is assumed that even moderately related strings have alignments with at least a fraction $p=0.75$, say, of alignment-characters being matches. Indels and mismatches are assumed to be equally probable. These costs work well in practice. If two strings are more closely related, these weights will cause an overestimate of the message length but it will still be less than for the null-theory. If the strings are less related, and this means very weakly related, the method may falsely support the null-theory.

The dynamic programming general step is modified.

$$\begin{array}{lll}
D[i,j] = \min(& D[i-1,j-1] + d(A[i], B[j]), & \text{match or mismatch} \\
 & D[i-1,j\ \ \ \ ] + d(A[i], -), & \text{insert } A \\
 & D[i,\ \ \ \ j-1] + d(-,\ \ B[j]), & \text{insert } B \\
 & U(i) + U(j) + (i+j)*2) & \text{end segments}
\end{array}$$

If the last option of the minimum is chosen, it indicates that $A[1..i]$ and $B[1..j]$ are end segments. The algorithm is applied in both a forward direction yielding D and in a reverse direction yielding D'. Diagonally adjacent elements of D and D' are then added to form S.

```
for $i$ in $0..m$ and $j$ in $0..n$ do
    $S[i,j] := D[i,j] + D'[i+1, j+1]$
```

The smallest value in S is the MML of a good e-theory of the desired form. If the null-theory is not supported, the minimum values in S lie on the alignment $A2|B2$. The alignment can be recovered by one of the standard methods. The end segments can be detected from information stored when the last value of the minimum above is chosen. The null-theory being valid is easily detected by its message length

$$U(|A|) + U(|B|) + 2*(|A| + |B|)$$

being shorter than any competitor.

The addition of D and D' to form S is related to the way in which Hirschberg's (1975) space efficient LCS algorithm, and its related similarity and edit distance variations, cut or partition A and B recursively.

Sellers (1980) gave a method for searching for a short key in a long string. His method is to set the boundary conditions in $D[,]$ to zero for the longer string. However this cannot be extended in a simple way to embedded strings because if all end segments cost zero, only the null theory is found—apparently having zero edit cost! The method described here solves this problem by computing a good estimate of the MML for a theory of the form $A1; B1; A2|B2; A3; B3$. The global minimum in the matrix S identifies the best theory of this form. There may also be local minima which identify areas of *local similarity* and these correspond approximately to the "common patterns" described by Sellers.

Tests were made on real and artificial data. In Monte-Carlo tests, segments $A1$, $A2$, $A3$, $B1$ and $B3$ were randomly created, and $B2$ was formed by making some random changes to $A2$. String A was set to $A1; A2; A3$ and B to $B1; B2; B3$. The algorithm calculated a common section $A2'|B2'$. The goodness of this calculation was measured by

$$\sqrt{}(|A2 \cap A2'| \cdot |B2 \cap B2'|/|A2| \cdot |B2|),$$

representing the fraction of the common section correctly identified. This was plotted vs the homology of $A2|B2$ for various string lengths. As the homology approaches close to the critical value, particularly for short segments of less than 50 characters, the algorithm may not identify $A2|B2$ (Fig. 2). It seems to be unavoidable that for short strings there is a small but non-negligible chance that the null-theory is more attractive at marginal levels of homology. As the length of the common region is increased beyond 100 characters the reliability greatly increases close to the critical homology (Fig. 3). Comparing these results with Fig. 1 strongly suggests that for long strings, the algorithm will find any common segment $A2|B2$ that it is possible to be confident about.

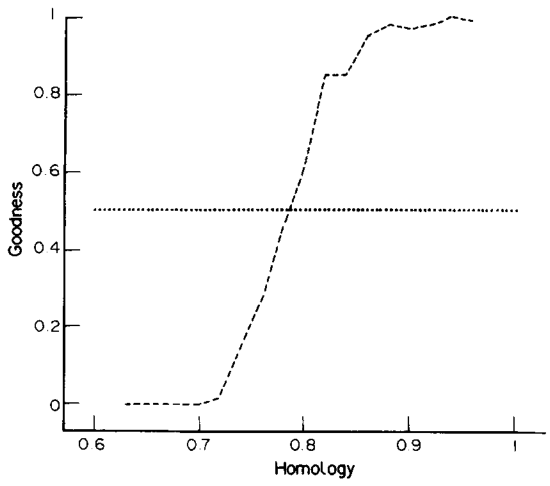

Figure 2.

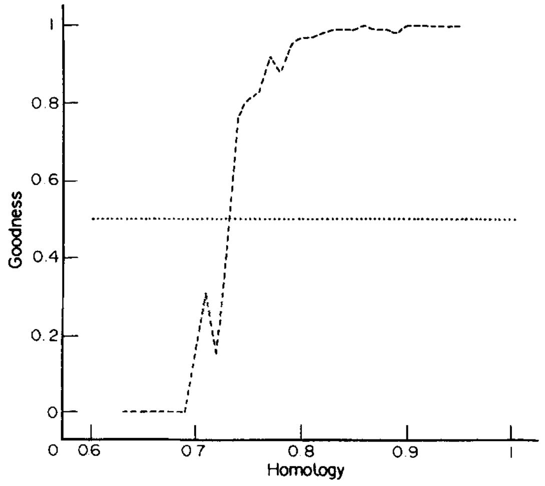

Figure 3.

The embedded-string algorithm uses an alignment algorithm as a subroutine. The model used to generate the data and the alignment model were the same CEM in these tests, however we believe that the results will be equally good on other kinds of data provided only that a good alignment model is used.

Note that the embedded-string algorithm does not suggest false *e*-theories when two unrelated strings are given. Stating the lengths of $A3$ and $B3$ is part of the cost of an *e*-theory, which the null-theory does not share, therefore the null-theory is favoured unless there is a significant alignment $A2|B2$ on which to base an *e*-theory.

When tested on some of the Blur strings, the algorithm correctly identifies that Blur 1 is essentially the front half of the typical full Blur string, and that Blur 10 is the second half. When Blur 11 and Blur 13 are compared, the algorithm detects the prefix on Blur 13 and the suffix on Blur 11. The difference in message lengths is approximately 100 bits in favour of this *e*-theory rather than the null-theory. When an alignment that forces the end points to be coincident is used, its message length is only slightly less than implied by the null-theory. That Blur 11 and 13 are related by the *e*-theory is overwhelmingly the most plausible of the three theories, by a factor of something like 2^{100}.

7. Block Editing Models (BEM). It is common for alignment programs to incorporate a penalty for introducing a *gap* or indel in a string. This favours fewer larger gaps over more smaller gaps, even of the same total length, and is more realistic biologically. Gotoh (1982) gave a fast algorithm for linear cost functions $w(k)=a+b\times k$ and for piecewise linear functions. Waterman (1984b) argued for concave cost functions and conjectured a fast algorithm for this case. Miller and Myers (1988) demonstrated such an algorithm.

An end segment, as in the previous section, is a special case where one end of the gap is known to be at the end of the string which allows its effect to be calculated easily and efficiently. We argue that the correct coding for an arbitrary gap in string B, say, is to state the substring that was notionally inserted in string A to cause the gap. Further, a block mismatch is naturally treated by stating two substrings—one from string A and one B. In this way a block indel becomes a special case of a block mismatch with one substring empty. An editing model which places more emphasis on blocks than on individual characters is called a *block editing model* (BEM).

A cost function implies a prior probability distribution on the lengths of blocks. A CEM cost function $w(k)=b\times k$, with cost proportional to length, corresponds to a geometric distribution; it amounts to coding lengths in unary notation by concatenating code words of b bits. Using such cost functions, the BEM algorithm behaves like a CEM algorithm. A linear cost function $w(k)=a+b\times k$ also implies a geometric distribution but with a higher mean, see the section on coding lengths. In the absence of more information on the actual distribution, we code $k\geq 1$ in $U(k)$ bits as argued previously.

A BEM alignment of two strings consists of alternating block matches and block mismatches. A header gives half the number of blocks and the type of the first block. A match states its length (≥ 1) and then gives its particular characters. A mismatch states two lengths and then gives the characters involved. The characters can be coded at 2 bits each, except that the first characters (and the last) of a block mismatch must differ and they can be coded in $\log_2(12)$ bits for two characters. Note also that it is possible for a mismatch to contain matching characters, for example:

```
CCTTACCT
    |
GAGAAGG
```

One cannot confidently assert that the marked A's are related in this context.

The cost of stating a length is concave downwards and non-linear in the length. Provided that certain simplifying approximations are made, a fast $O(n^2)$ or $O(n^2 \log n)$ algorithm for finding an MML alignment is possible. Miller and Myers (1988) gave an algorithm for concave gap weights (only) which takes $O(n^2)$ or $O(n^2 \log n)$ time depending on properties of the cost function. Our algorithm generalizes Miller and Myers' *candidate-list* algorithm to allow matching costs also to be concave and non-zero; candidate lists are maintained for the horizontal, vertical and also diagonal directions. Hirschberg's (1975) linear-space technique can also be applied. Note that concavity is not necessary to the use of a candidate-list. A sufficient condition is that $w(k)$ and $a+w(b+k)$ have at most one crossing point. Thus, the sum of a context-free, concave function of length and of a length-insensitive function of characters is sufficient. It is therefore possible to use a non-uniform coding for the characters if desired and still to have a fast algorithm.

Two simplifying approximations are necessary to give a fast algorithm. Firstly it is assumed that the two lengths stated in a block mismatch are independent.

probabilities: $P(\langle m, n\rangle) = P(m) \cdot P(n) \quad m, n \geq 0$
coding: $-\log_2(P(\langle m, n\rangle)) = -\log_2(P(m)) - \log_2(P(n))$

In fact the two lengths are not independent because at most one can be zero. The algorithm however assumes independence and the probability is renormalized by "redistributing" $P(<0, 0>)$ which is small. Given this assumption, it is ony necessary to maintain candidate-lists for insertions, deletions and matches; a mismatch is equivalent to an insertion plus a deletion. At each position in the dynamic-programming matrix, a list can grow by at most one element. Thus the *average* list management cost is $O(1)$. Without independence, candidate lists could grow by $O(n)$ at each step and would have to be merged and this would given an algorithm with $O(n^4)$ or possibly $O(n^3)$ complexity. Any error introduced by the independence assumption is probably too small to measure in terms of wrong alignments.

The first and last characters of a block mismatch can be coded in $\log_2(12) < 4$ bits for two characters. The saving for the first characters can be incorporated when a deletion follows an insert, or *vice versa*, to make a mismatch. The saving for the last characters is made at the switch back to a match or at the end of the alignment.

The second simplifying assumption, made when finding an alignment, is to ignore the overhead of stating (half) the number of blocks. This cost is concave and the average cost per block decreases with increasing numbers of blocks. For 30 blocks the total cost is approximately 11 bits. It is assumed that two rival, good alignments have a similar number of blocks and therefore a similar overhead. If this is valid, it is reasonable to ignore the overhead when finding an alignment. However once an alignment is found, it is then possible to calculate its true message length exactly, including all overheads. It is important to do this to be able to compare it to other hypotheses such as the null-theory. Note that without this assumption it would be necessary to use an expensive backtracking procedure.

The speed of the BEM algorithm depends on the difficulty of solving equations incorporating the cost functions. If such equations can be solved in $O(1)$ time, the algorithm is $O(n^2)$. Failing this, the equations are solved by a binary-search method. The alignment algorithm is then $O(n^2 \log(n))$.

8. Generality. It cannot be claimed that a BEM is the best editing model for DNA, indeed a point of this paper is that such a claim is impossible. However, if the priors for block lengths are taken as parameters of the model, the BEM can be seen to include *every* model that corresponds to a monotonic walk though $D[,]$ from $D[0, 0]$ to $D[|A|, |B|]$. Improvements will therefore either be at the tactical level of including contextual information and refining the priors or at the strategic level of incorporating major events such as reversals, duplicates and complements. The latter especially may be difficult unless an increase in computation time is accepted.

In the interests of simplicity, it has been assumed that letters occur with equal probabilities, that all substitutions are equally probable and we have used a particular prior for integers but these choices are not central to the MML principle.

9. Edit Distance. An MML alignment does not immediately assign an edit distance to a pair of strings $\langle A, B\rangle$ but there are a number of ways to do so. One way is to apply any distance function to the MML alignment. A better way is to use the difference in message lengths.

$$\mathrm{MML}(A, B) - ((\mathrm{MML}(A, A) + \mathrm{MML}(B, B))/2$$

If A and B are the same, this equals zero. It seems to be a metric under "reasonable" conditions but it is not clear what are necessary and sufficient conditions.

10. Conclusions. We have argued that the "minimum" in minimum message length encoding is the true meaning of the terms accuracy, parsimony and simplicity when comparing theories over strings. MML encoding explains the good performance of simple character editing models for closely related DNA strings and their poor performance in marginal cases. It gives an explanation of the improved performance when gap weights and non-linear gap costs are used. Given enough data, the odds ratio of two models of mutation can be calculated from message lengths.

For a given model, the posterior odds ratio of two alignments can be calculated from message lengths. As there is a natural null-theory for comparison, we can judge the plausibility of alignments. Improved alignment algorithms can be based on MML encoding. We gave an algorithm for the character editing model with an objective treatment of end segments. Finally, we argued that matching costs as well as indel costs should be non-linear and non-zero and we described a fast and general MML alignment algorithm.

The work makes it clear that the sole difference amongst a large class of editing models is the choice of prior probability distributions for block lengths. We suggest that any systematic search for the "true" distributions should be done at the point of capture of strings into genetic data-bases.

MML encoding allows comparison of alignments of a pair of strings within a model of mutation and, at a higher level, comparison of theories of string mutation. Current work is aimed at applications to the alignment of many strings and at the inference of phylogenetic trees for DNA and for proteins.

This work would not have been possible without the advice of Chris Wallace.

APPENDIX: CODING INTEGERS

Rissanen argues that a colourless or universal prior for the integers should be based on a prefix code for them. An integer $k \geq 1$ can be coded in $\lceil \log_2(k) \rceil$ bits except that the length of the code word must first be transmitted. This length is also an integer so its length must be transmitted and so on; fortunately this sequence very rapidly tends to zero, which need not be transmitted. The leading bit of k is necessarily "1" so there is no need to transmit it, except that it can be used as a flag to determine whether the current value is a length or the final integer k proper; lengths are thus given a leading "0". Such a prefix code can be used to code integers of arbitrary size. Unfortunately the length of a code word as a function of k is neither concave nor smooth although it is monotonic increasing.

Rissanen gives $r(k)$ as a continuous approximation to code word lengths and argues that a universal prior should be $2^{-r(k)}$.

$$r(k) = \log_2{}^*(k) + \log_2(2.865)$$
$$P(k) = 2^{-r(k)}$$

The constant term in $r(k)$ is to normalize the probability distribution.

Unfortunately $\log_2{}^*$ and $r(k)$ are not concave functions being convex at $k=2, 4, 16, 2^{16}$ and so on. This makes $r(k)$ unsuitable for use in the dynamic programming algorithm. Concavity can be restored by interpolation for $k=2$, 4 and so on.

A suitable concave function is $\log Y_2$

$$\log Y_2(k) = \log_2(k) + \log_2^2(k+2-1) + \log_2^3(k+2^2-1) + \log_2^4(k+2^{2^2}-1) + \cdots$$

but it increases rather quickly for small k. Note, only a few terms are needed to approximate $\log Y_2$.

The solution adopted in this paper to the coding of integers is to use $\log_2^*$, with interpolation, for $k \leq 8$ and $\log Y$ for larger k, see Fig. 4. This composite function is called "U" and implies a prior $P(k)=2^{-U(k)}$.

```
function U(k:integer):real; {k> =1}
  const delta=2.279
  var ans:real;
begin if k< =8 then
      begin
         if k in [2,4] then
              ans: =0.45*logstar(k−1)+0.55*logstar(k+1) {interpolate}
         else ans: =logstar(k);
            ans: =ans+logY(9)−logstar(9) {smooth join}
      end
      else  {k>8}
         ans: =logY(k);
      U: =ans+log2(delta) {normalize}
end;
```

k	code (bits)	probability
1	$U(1)=1.586$	$P(1)=0.3332$
2	$U(2)=2.823$	$P(2)=0.1413$

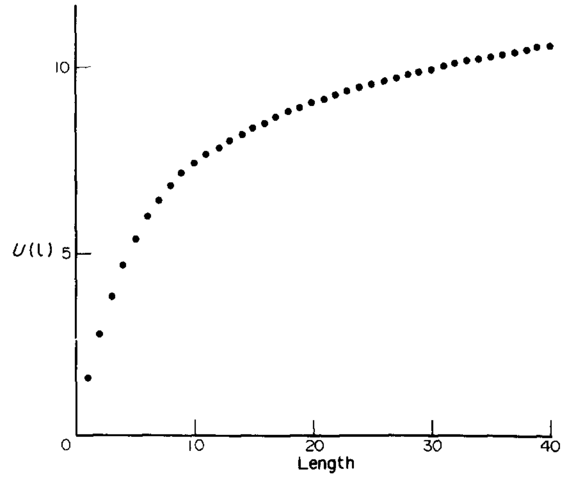

Figure 4.

3	$U(\ 3)=3.835$	$P(\ 3)=0.0701$
4	$U(\ 4)=4.698$	$P(\ 4)=0.0385$
5	$U(\ 5)=5.404$	$P(\ 5)=0.0236$
6	$U(\ 6)=5.995$	$P(\ 6)=0.0157$
7	$U(\ 7)=6.457$	$P(\ 7)=0.0114$
8	$U(\ 8)=6.835$	$P(\ 8)=0.0088$
9	$U(\ 9)=7.155$	$P(\ 9)=0.0070$
10	$U(10)=7.416$	$P(10)=0.0059$

Rather than call U repeatedly, values are calculated once and stored.

An integer variable, $k \geq 1$ is coded in $U(k)$ bits. A variable $m \geq 0$ is coded in $U(m+1)$ bits. Recall that arithmetic coding is capable of transmitting fractional numbers of bits. Note that we do *not* argue that $2^{-U(k)}$ is the correct prior for integers in all cases, just that it is a good universal prior to use if no more information is available. If something is known of an actual distribution then it most certainly should be used!

LITERATURE

Allison, L. and T. I. Dix. 1986. A bit-string longest common subsequence algorithm. *Inf. Processing Lett.* **23**, 305–310.

Bains, W. 1986. The multiple origins of the human Alu sequences. *J. molec. Evol.* **23**, 189–199.

Boulton, D. M. and C. S. Wallace. 1969. The information content of a multistate distribution. *J. theor. Biol.* **23**, 269–278.

Chaitin, G. J. 1966. On the length of programs for computing finite binary sequences. *J. ACM* **13**, 547–569.

Cohen, D. N., T. A. Reichert and A. K. C. Wong. 1975. Matching code sequences utilizing context free quality measures. *Math. Biosci.* **24**, 25–30.

Dayhoff, M. O. 1978. *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 3. Washington, DC: National Biomedical Research Foundation.

Deken, J. 1983. Probabilistic behaviour of longest common subsequence lengths. In *Time Warps, String Edits and Macro-Molecules*, D. Sankoff and J. B. Kruskall (eds). Reading, MA: Addison Wesley.

Gatlin, L. L. 1974. Conservation of Shannon's redundancy of proteins. *J. mol. Evol.* **3**, 189–208.

Georgeff, M. P. and C. S. Wallace. 1984. A general selection criterion for inductive inference. Proceedings of the European Conference on Artificial Intelligence, pp. 473–482.

Gotoh, O. 1982. An improved algorithm for matching biological sequences. *J. molec. Biol.* **162**, 705–708.

Hamming, R. W. 1980. *Coding and Information Theory*. Englewood Cliffs, NJ: Prentice Hall.

Hasegawa, M. and Taka-Aki Yani. 1975. The genetic code and the entropy of protein. *Math. Biosci.* **24**, 169–182.

Hirschberg, D. S. 1975. A linear space algorithm for computing maximal common subsequences. *Commun. ACM* **18**, 341–343.

Jimenez-Montano, M. A. 1984. On the syntactic structure of protein sequences and the concept of grammar complexity. *Bull. math. Biol.* **46**, 641–659.

Kolmogorov, A. N. 1965. Three approaches to the quantitative definition of information. *Prob. Inf. Transmission* **1**, 1–7.

Langdon, G. G. 1984. An introduction to arithmetic coding. *IBM J. Res. and Dev.* **28**, 135–149.

Miller, W. and E. W. Myers. 1988. Sequence comparison with concave weighting functions. *Bull. math. Biol.* **50**, 97–120.

Ming Li and P. M. B. Vitanyi. 1988. Two decades of applied Kolmogorov Complexity. Proceedings of the Third Annual Conference on Structure in Complexity Theory. IEEE 80–101.

Reichert, T. A., D. N. Cohen and K. C. Wong. 1973. An application of information theory to genetic mutations and the matching of polypeptide sequences. *J. theor. Biol.* **42**, 245–261.

Rissanen, J. 1983. A universal prior for integers and estimation by minimum description length. *Ann. Stat.* **11**, 416–431.

Sankoff, D. and J. B. Kruskall (eds). 1983. *Time Warps, String Edits and Macro-Molecules.* Reading, MA: Addison Wesley.

Sellers, P. H. 1974. On the theory and computation of evolutionary distances. *SIAM J. appl. Math.* **26**, 787–793.

Sellers, P. H. 1980. The theory and computation of evolutionary distances: pattern recognition. *J. Algorithms* **1**, 359–373.

Shepherd, J. C. W. 1981. Method to determine the reading frame of a protein from the purine/pyrimidine genome sequence and its possible evolutionary justification. *Proc. natl. Acad. Sci.* **78**, 1596–1600.

Smith, T. F. 1969. The genetic code, information density and evolution. *Math. Biosci.* **4**, 179–187.

Smith, T. F. and M. S. Waterman. 1980. Protein constraints induced by multiframe encoding. *Math. Biosci.* **49**, 17–26.

Smith, T. F., M. S. Waterman and W. M. Fitch. 1981. Comparative biosequence metrics. *J. molec. Evol.* 18, 38–46.

Solomonoff, R. 1964. A formal theory of inductive inference, I and II. *Inf. Control* **7**, 1–22, 224–254.

Staden, R. and A. D. McLachlan. 1982. Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucleic Acids Res.* **10**, 141–156.

Turing, A. M. 1936. On computable numbers, with an application to the entscheidungsproblem. *Proc. Lon. math. Soc.* **2**, 230–265, 544–546.

Wallace, C. S. and D. M. Boulton. 1968. An information measure for classification. *Comput. J.* **11**, 185–194.

Wallace, C. S. and P. R. Freeman. 1987. Estimation and inference by compact coding. *J. R. Stat. Soc.* **B49**, 240–265.

Wallace, C. S. 1989. Personal communication.

Waterman, M. S. 1984. General methods of sequence comparison. *Bull. math. Biol.* **46**, 473–500.

Waterman, M. S. 1984b. Efficient sequence alignment algorithms. *J. theor. Biol.* **108**, 333–337.

Waterman, M. S. and M. Eggert. 1987. A new algorithm for best subsequence alignments and application to tRNA–rRNA comparison. *J. molec. Biol.* **197**, 723–728.

Witten, I. H., R. M. Neal and J. G. Cleary. 1987. Arithmetic coding for data compression. *Commun. ACM* **30**, 520–540.

Wong, A. K. C., T. A. Reichert, D. N. Cohen and B. O. Aygun. 1974. A generalized method for matching informational macromolecular code sequences. *Comput. Biol. Med.* **4**, 43–57.

Received 24 May 1989
Revised 20 September 1989