Hindawi Publishing Corporation
EURASIP Journal on Bioinformatics and Systems Biology
Volume 2007, Article ID 72936, 14 pages
doi:10.1155/2007/72936

Research Article

Aligning Sequences by Minimum Description Length

John S. Conery

Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, USA

Received 26 February 2007; Revised 6 August 2007; Accepted 16 November 2007

Recommended by Peter Grünwald

This paper presents a new information theoretic framework for aligning sequences in bioinformatics. A transmitter compresses a set of sequences by constructing a regular expression that describes the regions of similarity in the sequences. To retrieve the original set of sequences, a receiver generates all strings that match the expression. An alignment algorithm uses minimum description length to encode and explore alternative expressions; the expression with the shortest encoding provides the best overall alignment. When two substrings contain letters that are similar according to a substitution matrix, a code length function based on conditional probabilities defined by the matrix will encode the substrings with fewer bits. In one experiment, alignments produced with this new method were found to be comparable to alignments from CLUSTALW. A second experiment measured the accuracy of the new method on pairwise alignments of sequences from the BAliBASE alignment benchmark.

1. INTRODUCTION

Sequence alignment is a fundamental operation in bioinformatics, used in a wide variety of applications ranging from genome assembly, which requires exact or nearly exact matches between ends of small fragments of DNA sequences [1], to homology search in sequence databases, which involves pairwise local alignment of DNA or protein sequences [2], to phylogenetic inference and studies of protein structure and function, which depend on multiple global alignments of protein sequences [3–5].

These diverse applications all use the same basic definition of alignment: a character in one sequence corresponds either to a character from the other sequence or to a "gap" character that represents a space in the middle of the other sequence. Alignment is often described informally as a process of writing a set of sequences in such a way that matching characters are displayed within the same column, and gaps are inserted in strings in order to maximize the similarity across all columns. More formally, alignments can be defined by a matrix M, where M_{ij} is 1 if character i of one sequence is aligned with character j of the other sequence, or in some cases, M_{ij} is a probability, for example, the posterior probability of aligning letters i and j [6].

This paper introduces a new framework for describing the similarities and differences in a set of sequences. The idea is to construct a special-purpose grammar for the strings that represent the sequences. If there are segments in each input sequence that are similar to corresponding segments in the other sequences, the grammar will have a single rule that directly generates the characters for these segments.

An alignment algorithm based on this new framework will consider different sets of rules to include in the grammar it produces. The focus of this paper is on the use of minimum description length (MDL) [7] as the basis of the alignment algorithm. The MDL principle argues that the best alignment will be the one described by the shortest grammar, where the length of a grammar is measured in terms of the number of bits needed to encode it.

The key idea is to use conditional probabilities to encode letters in aligned regions. If a grammar has a rule that aligns letter x in one sequence with letter y in another sequence, the encoding of the rule will be based on $p(y \mid x)$, and if the alignment is accurate, the resulting encoding is shorter than the one that encodes x and y separately in an unaligned region. But there is a tradeoff: adding a new rule to a grammar requires adding new symbols for the rule structure, and the number of bits required to encode these symbols adds to the total size of the encoded grammar. The alignment algorithm must determine the net benefit of each potential aligned region and choose the set of aligned regions that provides the overall shortest encoding.

MDL has been used to infer grammars for large collections of natural language sentences [8] and to search for recurring patterns in protein and DNA sequences [9]. These applications of MDL are examples of machine learning, where the system uses the data as a training set and the goal is to infer a general description that can be applied to other data. The goal of the sequence alignment algorithm presented here is simply to find the best description for the data at hand; there is no attempt to create a general grammar that may apply to other sequences.

Grammars have been used previously to describe the structure of biological sequences [10–12], and regular expressions are a well-known technique for describing patterns that define families of proteins [13]. But as with previous work on MDL and grammars, these other applications use grammars and regular expressions to describe general patterns that may be found in sequences beyond those used to define the pattern, whereas for alignment the goal is to find a grammar that describes only the input data.

Grammars have the potential to describe a wide variety of relationships among sequences. For example, a top level rule might specify several different ways to partition the sequences into smaller groups, and then specify separate alignments for each group. In this case, the top level rules are effectively a representation of a phylogenetic tree that shows the evolutionary history of the sequences. This paper focuses on one very restricted type of grammar that is capable of describing only the simplest correspondence between sequences. The algorithm presented here assumes that only two sequences are being aligned, and that the goal is to describe similarity over the entire length of both input sequences, that is, the algorithm is for pairwise global alignment. For this application, the simplest type of formal grammar—a right linear grammar—is sufficient to describe the alignment. Since every right linear grammar has an equivalent regular expression, and because regular expressions are simpler to explain (and are more commonly used in bioinformatics), the remainder of this paper will use regular expression syntax when discussing grammars for a pair of sequences.

Current alignment algorithms are highly sensitive to the choice of gap parameters [14–17]; for example, Reese and Pearson showed that the choice of gap penalties can influence the score for alignments made during a database search by an order of magnitude [18]. One of the advantages of the grammar-based framework is that gaps are not needed to align sequences of varying length. Instead, the parts of regular expressions that correspond to regions of unaligned positions will have a different number of characters from each input sequence.

Previous work using information theory in sequence alignment has been within the general framework of a Needleman-Wunsch global alignment or Smith-Waterman local alignment. Allison et al. [19] used minimum message length to consider the cost of different sequences of edit operations in global alignment of DNA; Schmidt [20] studied the information content of gapped and ungapped alignments, and Aynechi and Kuntz [21] used information theory to study the distribution of gap sizes. The work described here takes a different approach altogether, since gap characters are not used to make the alignments.

Regular expression alignments are similar to the alignments produced by DIALIGN [22, 23], a program that creates consistent sets of ungapped local alignments. The main differences are that fragments in DIALIGN are defined by a Smith-Waterman alignment based on finding a locally optimal score and including neighboring letters until the score drops below a threshold, and DIALIGN uses a minimum length parameter to exclude short random matches. The method presented in this paper uses the MDL criterion to find the ends of aligned regions—if adding a pair of letters is less costly than leaving the letters in a variable region, then the letters are included in the aligned region.

Other methods that consider only ungapped local alignments are also similar to regular expression alignments. Schneider [24] used information theory as the basis of a multiple alignment algorithm for small ungapped DNA sequences and successfully applied it to binding sites. More recently, Krasnogor and Pelta [25] described a method for evaluating the similarity of pairs of proteins, but their analysis describes a global similarity metric without actually aligning the substrings responsible for the similarity.

The next section of this paper provides some background information on sequence alignment and explains in more detail how a regular expression can be used to capture the essential information about the similarity in a set of sequences. The details of the MDL encoding for sequence letters and other symbols found in expressions are given in Section 3. Results of two sets of experiments designed to test the method are presented in Section 4.

The regular expression alignment method described in this paper has been implemented in a program named `realign`. The source code, which is written in C++ and has been tested on OS/X and Linux systems, is freely available under an open source license and can be downloaded from the project web site [26].

2. ALIGNMENTS AND REGULAR EXPRESSIONS

One of the main applications of sequence alignment is comparison of protein sequences. The inputs to the algorithm are sets of strings, where each letter corresponds to one of the 20 amino acids found in proteins. The goal of the alignment is to identify regions in each of the input sequences that are parts of the same structural or functional elements or are descended from a common ancestor.

Figure 1(b) shows the evolution of fragments of three hypothetical proteins starting from a 9-nucleotide DNA sequence. The labels below the leaves of the tree are the amino acids corresponding to the DNA sequences at the leaves. The only change along the left branch is a single substitution which changes the first amino acid from `P` to `T`, and an alignment algorithm should have no problem finding the correspondences between the two short sequences (Figure 1(c)).

The sequence on the right branch of the tree is the result of a mutation that inserted six nucleotides in the middle of the original sequence. In order to align the resulting sequence with one of its shorter cousins, a standard alignment algorithm inserts a gap, represented by a sequence of one or more dashes, to mark where it thinks the insertion occurred.

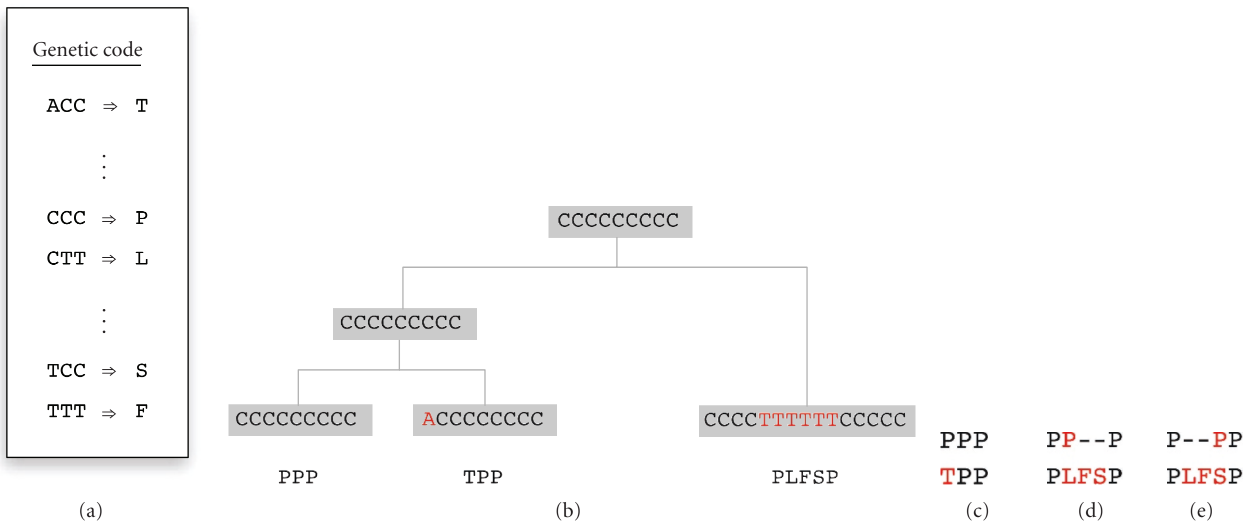


Figure 1: (a) The genetic code specifies how triplets of DNA letters (known as "codons") are translated into single amino acids when a cell manufactures a protein sequence from a gene. (b) A tree showing the evolution of a short DNA sequence. Labels below the leaves are the corresponding amino acid sequences. (c) Alignment of the two shorter sequences. (d) and (e) Two ways to align the longer sequence with one of the shorter ones.

This alignment is complicated by the fact that the insertion occurred in the middle of a codon; the single CCC that corresponded to a P in the ancestral sequence is now part of two codons, CCT and TTC. Figures 1(d) and 1(e) show two different ways of doing the alignment; the difference between the two is the placement of the gap, which can go either before or after the middle P of the short sequence.

A key parameter in the alignment of protein sequences is the choice of a substitution matrix, a 20×20 array S in which $S_{i,j}$ is a score for aligning amino acid i with amino acid j. The PAM matrices [27] were created by analyzing hand alignments of a carefully chosen set of sequences that were known to be descending from a common ancestor. PAM matrices are identified by a number that indicates the degree to which sequences have changed; a unit of "1 PAM" is roughly the amount of sequence divergence that can be expected in 10 million years [28], so the PAM20 matrix could be used to align a set of sequences where the common ancestor lived around 200 million years ago. Other common substitution matrices are the BLOSUM family [29] and the Gonnet matrix [30].

Substitution matrices give higher scores to pairs of letters that are expected to be found in alignments, and lower (negative) scores to pairings that are rare. For example, the PAM100 matrix has positive scores on the main diagonal, to use when aligning letters with themselves; the highest score is 12, for the pair W/W, since tryptophan (W) is highly conserved. Smaller positive scores are for letters that frequently substitute for one another, for example, leucine (L) and isoleucine (I) are both hydrophobic and the matrix entry for the pair I/L is 1. Histidine (H) is hydrophilic, and the matrix entry for I/H is -4. The pair P/L has a score of -4 and the pair P/S has a score of 0, so an algorithm using PAM100 would prefer the alignment shown in Figure 1(e).

Regular expressions are widely used for pattern matching, where the expression describes the general form of a string and an application can test whether a given string matches the pattern. To see how a regular expression is an alternative to a standard gap-based alignment consider the following pattern, which describes the two sequences in Figures 1(d) and 1(e):

$$\texttt{P(P | LFS)P}. \tag{1}$$

Here the vertical bar means "or" and the parentheses are used to mark the ends of the alternatives. The pattern described by this expression is the set of strings that start with a P, then have either another P or the string LFS, and end in a P. In this example, the letters enclosed in parentheses correspond to a variable region: the pattern simply says "these letters are not aligned" and no attempt is made to say why they are not aligned or what the source of the difference is. The regular expression is an abstract description, covering both the alignments of Figures 1(d) and 1(e) (and a third, biologically less plausible, alignment in which the top string would be P–P–P).

For a more realistic example, consider the two sequence fragments in Figure 2(a), which are from the beginning of two of the protein sequences used to test the alignment application. Substrings of 15 characters near the front of each sequence are similar to each other. A regular expression that describes this similarity would have three groups, showing letters before and after the region of similarity as well as the region itself (Figure 2(b)).

Any pair of sequences can be described by a regular expression of this form. The expression consists of a series of segments, written one after another, where each segment has two substrings separated by the vertical bar. But this standard

```
MNNNNYIFENENPILYNTNEGEENRSS...
MNSYKPENENPVLYNYKEDEESSHI...
```

(a)

```
(MNNNNYIF|MNSYKP)(ENENPILYNTNEGEE|ENENPVLYNYKEDEE)(NRSS...|SSHI...)
```

(b)

```
>MNNNNYIF
>MNSYKP
#ENENPILYNTNEGEE
#ENENPVLYNYKEDEE
>NRSS...
>SSHI...
```

(c)

Figure 2: (a) Strings from the start of two of the amino acid sequences used to test the alignment algorithm. The substrings in blue are similar to the corresponding substring in the other sequence. (b) A regular expression that makes explicit the boundaries of the region of similarity. (c) The canonical form representation of the regular expression. The canonical form has the same groupings of letters, but displays the letters in a different order and uses marker symbols instead of parentheses to specify group boundaries. A # means the sequence segments are blocks, where the *i*th letter from one sequence has been aligned with the *i*th letter in the other sequence. A > designates the start of a variable region of unaligned letters.

notation introduces a problem: how does one distinguish segments describing aligned characters from segments for unaligned characters? The following convention solves the problem of distinguishing between the types of segments and reduces the number of symbols to a minimum. In a *canonical form sequence expression,*

(i) each open parenthesis is replaced with a symbol that specifies the type of the segment that starts at that location. An aligned segment starts with #, an unaligned segment starts with >;

(ii) the vertical bar separating the two parts of a segment is replaced by the symbol used at the start of the segment; thus if the segment starts with #, the two parts of the segment are separated by a second #;

(iii) the closing parenthesis marking the end of a segment can just be deleted since it is redundant (every closing parenthesis is either followed by an opening parenthesis or comes at the end of the expression);

(iv) to make an expression easier to read, it is displayed by starting a new line for each # or >, with the understanding that "white space" breaking the expression into new lines is for formatting purposes only and is not part of the expression itself.

The canonical form of the expression describing the alignment of the initial parts of the two example genes is shown in Figure 2(c).

In the literature on sequence alignment, an ungapped local alignment is often referred to as a *block*. In the canonical form sequence expression, a block corresponds to a pair of lines starting with #; pairs of lines starting with > are called *variable regions*. Note that the substrings in blocks always have the same number of sequence letters, and always have at least one letter. Substrings in variable regions can have any number of sequence letters, and one of the strings can have zero letters. Since # and > define the boundaries of blocks they are referred to as *marker* symbols.

Sequence expressions can easily be extended to describe a multiple alignment of $n > 2$ sequences. Each segment in an expression would have n substrings separated by vertical bars, and the corresponding canonical form would have n lines in each block and in each variable region. The MDL code length function and the alignment algorithm in the following section assume there are only two sequences; possible extensions for multiple alignment will be discussed in the final section.

3. ALIGNMENT USING MINIMUM DESCRIPTION LENGTH

It is easy to see there is at least one canonical form sequence expression for every pair of sequences: simply create a single variable region, writing the string for each complete sequence to the right of a > symbol. This default expression is the null hypothesis that the sequences have nothing in common.

The goal of an alignment algorithm is to generate alternative hypotheses, in the form of expressions that have one or more blocks containing equal-length substrings from the input sequences. The alignment process can be viewed as a series of rewrite operations applied to variable regions. A rewrite step that creates a block splits a variable region into three parts: a variable region for characters before the block, the block itself, and a variable region for characters following the block (Figure 3). The transformation adds four marker symbols to the expression: two # symbols identify

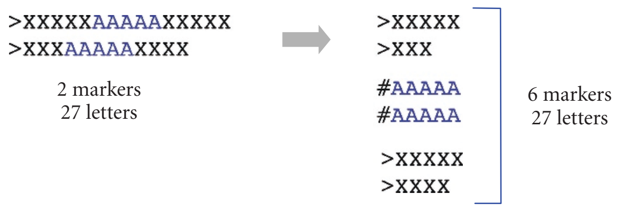


Figure 3: Schematic representation of an expression rewriting operation. A canonical form expression with a single variable region is transformed into a new expression with two variable regions surrounding a block. The number of sequence letters does not change, but four new marker symbols are added to specify the boundaries of the block.

the locations of the start of the block (one in each input sequence) and two > symbols mark the end of the block. As a special case, the block might be at the beginning or end of the expression; if so only two new # markers are added to the expression.

Since the alignment algorithm uses the minimum description length principle to search for the simplest expression, this transformation appears to be a step in the wrong direction because the complexity of the expression, in terms of the number of symbols used, has increased. The key point is that MDL operates at the level of the *encoding* of the expression, that is, it prefers the expression that can be encoded in the fewest number of bits. As will be shown in this section, blocks of similar sequence letters have shorter encodings. If the number of bits saved by placing similar letters in a block is greater than the cost of encoding the symbols that mark the ends of the block, the transformed expression is more compact.

The code length function that assigns a number of bits to each symbol in a canonical form sequence expression has three components:

(i) a protocol that defines the general structure of an expression and the representation of alignment parameters;
(ii) a method for assigning a number of bits to each letter from the set of input sequences;
(iii) a method for determining the number of bits to use for the marker symbols that identify the boundaries between blocks and variable regions.

3.1. Communication protocol

A common exercise in information theory is to imagine that a compressed data set is going to be sent to a receiver in binary form, and the receiver needs to recover the original data. This exercise ensures that all the necessary information is present in the compressed data—if the receiver cannot reconstruct the original data, it may be because essential information was not encoded by the compression algorithm. In the case of the MDL alignment algorithm, the idea is to compress a set of sequences by creating a representation of a regular expression that describes the structure of the sequences. The receiver recovers the original sequence data by expanding the expression to generate every sequence that matches the expression.

A "communication protocol" that specifies the type of information contained in a message and the order in which the pieces of the message are transmitted is an essential part of the encoding. The representation of a sequence expression begins with a preamble that contains information about the structure of the expression and the encoding of alignment parameters.

A canonical form sequence expression is an alternating series of blocks and variable regions, where the marker symbols (# and >) inserted into the input sequences identify the boundaries between segments. The communication protocol allows the transmitter to simplify the expression as it is compressed by putting a single bit in the preamble to specify the type of the first segment. Then the only thing that is required is a single type of symbol to specify the locations of the remaining markers. For the example sequences shown in Figure 2, the expression can be transformed into the following string:

$$> \texttt{MNNNNYIF.MNSYKP.ENENPILYNTNEGEE.} \\ \texttt{ENENPVLYNYKEDEE.NRSS.SSHI} \tag{2}$$

Here the >, represented by a single bit, indicates the type of the first region. The periods identify the locations of the markers. Since the regions alternate between # and >, the receiver infers the first period that represents another >, the next two periods are #, and so on.

The key parameter in every alignment is the substitution matrix used to define joint probabilities for each letter pair and single (marginal) probabilities for each individual letter. If the transmitter and receiver agree beforehand to restrict the set of substitution matrices to a set of n commonly used matrices, each matrix can be assigned an integer ID and the preamble simply contains a single integer encoded in $\lceil \log_2 n \rceil$ bits to identify the matrix. If an arbitrary matrix is allowed, the protocol would have to include a representation for the substitution matrix.

The rest of the information contained in the preamble depends on the method used to represent the marker symbols. Three different methods are presented below in Section 3.3, and each uses a different combination of parameters; for example, the indexed representation requires the transmitter to send the length of the longest sequence, and the tagged representation requires the transmitter to send the number of bits used in the encoding of marker symbols. For numeric parameters, the transmitter can simply encode the parameter in the fewest number of bits and include the encoding as part of the preamble. A standard technique for representing a number that can be encoded in k bits is to send k 0s, a 1, and then the k bits that encode the number itself.

In general a regular expression can be expanded into more than just the original sequence strings. For example, suppose the two input strings are AB and CD, and the regular expression representing their alignment is of the form

$$(\texttt{A} \mid \texttt{C})\ (\texttt{B} \mid \texttt{D}). \tag{3}$$

A receiver can expand this expression into the two original input strings, but the expression also matches `AD` and `CB`. Thus the protocol needs a method for telling the receiver how to link together the substrings from different segments so that it will reconstruct `AB` and `CD` but not `AD` or `CB`.

One solution would be to encode sequence IDs with the substrings so the receiver correctly pieces together a sequence using a consistent set of IDs. But if a simple convention is followed, the receiver can infer the sequence IDs from the order in which the sequences are transmitted. For canonical form sequence expressions, the protocol requires that every region has exactly two strings, and that within a region, the strings need to be given in the same order each time.

3.2. *Encoding sequence letters*

The standard technique used in information theory of encoding symbols according to their probability distribution can be used to encode sequence letters. If a letter x occurs with probability $p(x)$ the encoding of x requires $-\log_2 p(x)$ bits.

The probability distribution for letters is based on the substitution matrix being used for the alignment. Scores in a substitution matrix are log odds ratios of the form

$$s(x,y) = \frac{1}{\lambda}\log\frac{p(x,y)}{p(x)p(y)} \qquad (4)$$

where $p(x,y)$ is the joint probability of observing x aligned with y, $p(x)$ and $p(y)$ are the background probabilities of x and y, and λ is a scaling factor [31]. The `realign` program uses a program named `lambda` [32] as a preprocessor that takes an arbitrary substitution matrix as input, solves for λ, and saves a table of background probabilities for each single letter and joint probabilities for each letter pair.

The number of bits used to encode a letter in a canonical sequence expression depends on whether the letter is in a block or in a variable region. For a letter x in a variable region the encoding is straightforward: simply use the background probability of x according to the transformed substitution matrix.

For a block, the encoding considers pairs of letters x and y that occur in the same relative position in the block. The number of bits to encode the letter x in one sequence is based on $p(x)$, the same as in a variable region, but for the letter y in the other sequence, the conditional probability $p(y \mid x)$ is used to reflect the fact that x and y are aligned. Since by definition $p(y \mid x) = p(x,y)/p(x)$, the substitution matrix provides the necessary information to compute the conditional probabilities.

To summarize, the cost, in bits, of encoding letters in a canonical form sequence expression is defined as follows:

(i) for a letter x in a variable region or in the first line of a block, the code length is a function of $p(x)$, the marginal probability of observing $x : c(x) = -\log_2 p(x)$;

(ii) for a letter y in the second line of a block, the code length is a function of $p(y \mid x)$, the conditional probability of seeing y in this location given character x in the same position in the first line: $c(y,x) = -\log_2 p(y \mid x)$.

Table 1: Cost (in bits) of aligning pairs of letters. $S_{x,y}$ is the score for letters x and y in the PAM100 substitution matrix. $c(x) + c(y)$ is the sum of the costs of the two letters, which is incurred when the letters are in a variable region. $c(x) + c(y \mid x)$ is the cost of the same letters when they are aligned in a block. The benefit of aligning two letters is the difference between the unaligned cost and the aligned cost: a positive benefit results from aligning similar letters, a negative benefit from aligning dissimilar letters.

x	y	$S_{x,y}$	$c(x)+c(y)$	$c(x)+c(y \mid x)$	benefit(y,x)
W	W	12	6.36 + 6.36	6.36 + 0.44	5.92
I	I	6	3.65 + 3.65	3.65 + 1.25	2.40
L	L	6	3.09 + 3.09	3.09 + 0.72	2.37
M	L	3	4.97 + 3.09	4.97 + 2.26	0.83
L	I	1	3.09 + 3.65	3.09 + 3.66	−0.01
L	Q	−2	3.09 + 5.02	3.09 + 6.09	−1.07
L	C	−6	3.09 + 5.78	3.09 + 9.38	−3.60

When x and y are the same letter, or similar according to the substitution matrix being used, the cost using the conditional probability will be lower. For any two letters x and y, the benefit of aligning y with x is the difference between the cost of placing the two letters in a variable region versus their cost in a block:

$$\begin{aligned}\text{benefit}(y,x) &= \big(c(x)+c(y)\big) - \big(c(x)+c(y \mid x)\big)\\ &= c(y) - c(y \mid x).\end{aligned} \qquad (5)$$

In general, there is a positive benefit for pairs of letters that have positive scores in a substitution matrix. On the other hand, a negative benefit is incurred when an algorithm tries to align two dissimilar letters. Table 1 shows a few examples of pairs of letters, the cost of placing them unaligned in a variable region, and the benefit gained from aligning them in a block.

3.3. *Encoding marker symbols*

Three different methods for encoding of the marker symbols that identify the boundaries between blocks and variable regions are illustrated in Figure 4. All three methods are based on the transformation in which the # and > symbols have been replaced by periods. The difference between the three methods is in the representation of each marker and the additional information included in the preamble.

3.3.1. *Indexed representation*

The indexed representation for marker symbols is based on the observation that it is not necessary to include the marker symbols themselves, but only their locations in each string. If an expression has m segments, the transmitter can construct a table of $(m-1)$ entries for each string. The number of bits for each table entry depends on n, the length of the corresponding input sequence. Using this technique, the preamble of a message is constructed as follows:

(i) order the input sequences so the longest sequence is the first one in the message;

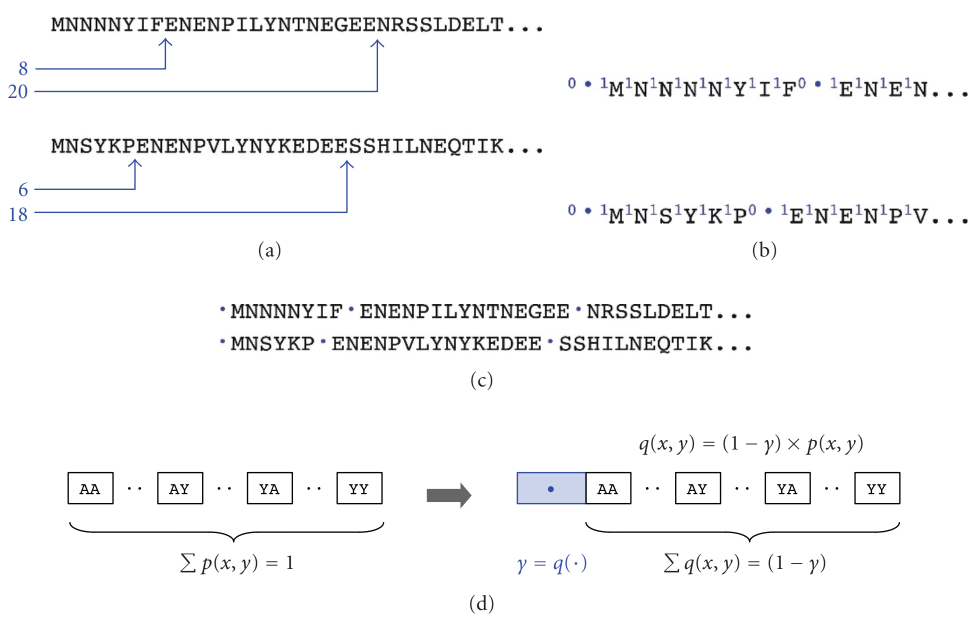

MNNNNYIFENENPILYNTNEGEENRSSLDELT...
8
20
MNSYKPENENPVLYNYKEDEESSHILNEQTIK...
6
18
(a)
0 • 1M1N1N1N1N1Y1I1F0 • 1E1N1E1N...
0 • 1M1N1S1Y1K1P0 • 1E1N1E1N1P1V...
(b)
•MNNNNYIF•ENENPILYNTNEGEE•NRSSLDELT...
•MNSYKP•ENENPVLYNYKEDEE•SSHILNEQTIK...
(c)
AA AY YA YY
Σ p(x, y) = 1
q(x, y) = (1 − γ) × p(x, y)
γ = q(·)
Σ q(x, y) = (1 − γ)
(d)

FIGURE 4: The items in blue correspond to information added to a string to specify the locations of marker symbols. (a) Indexed representation. The preamble contains two tables of $m - 1$ numbers to specify the locations of the m marker symbols (the first marker is always at the front of the string) in each sequence. Each table entry has $k = \lceil \log_2 n \rceil$ bits to specify a location in a string of length n. (b) Tagged representation. A one-bit tag added to each symbol identifies the symbol class (letter or marker), and is followed by the bits that represent the symbol itself. (c) Scaled representation. The number of bits for each symbol x is simply $-\log_2 q(x)$ where $q(x)$ is the probability of the symbol based on a distribution that includes the probability of a marker. (d) Given a probability γ for marker symbols, the joint probabilities for the letter pairs are scaled by $1.0 - \gamma$ so the sum of probabilities over all symbols is 1.0.

(ii) use one bit to specify the type of the first segment (which will be the same for both sequences);

(iii) use $\lceil \log_2 s \rceil$ bits to specify which one of the s substitution matrices was used to encode letters and letter pairs;

(iv) use $2\log_2 n + 1$ bits to specify n, the length of the first input sequence. This number also allows the receiver to determine $k = \log_2 n$, the number of bits required to represent a single marker table entry;

(v) the next $2\log_2 m + 1$ bits specify m, the number of marker symbols in each sequence;

(vi) create a table of size mk bits for the locations of the m markers in the first sequence, followed by another table of the same size for the markers of the second sequence.

Following the preamble, the body of the message simply consists of the encoding of the letters defined in the previous section. Since the receiver knows the length of the first sequence, there is no need to include an end-of-string marker after the first sequence. This location becomes a de facto marker for the start of the second sequence.

Figure 4(a) shows how the start of the two example sequences would be encoded with the indexed representation. The numbers in blue are indices between 0 and the length of the longer of the two sequences.

The advantage of this representation is that no additional parameters are required to align a pair of sequences: the only alignment parameter is the substitution matrix, which determines the individual probability for each letter and the joint probability for each letter pair.

3.3.2. *Tagged representation*

There are two drawbacks to the indexed representation. The first is that the number of bits used to represent a marker grows (albeit very slowly) with the length of the input sequences. That means one might get a different alignment for the same two substrings of sequence letters in different contexts; if the substrings are embedded in longer sequences, the number of bits per marker will increase, and the alignment algorithm might decide on a different placement for the markers in the middle of the substrings.

The second disadvantage is that in many cases marker symbols identify the locations of insertions and deletions, which are evolutionary events. The number of bits used to represent a marker should correspond to the likelihood of an insertion or deletion, but not the length of the sequence. If anything, longer sequences are more likely to have had insertions or deletions, so the number of bits representing those events should be lower, not higher.

The tagged representation addresses these problems by defining a prefix code for markers and embedding the marker codes in the appropriate locations within each sequence string. This method requires the user to specify a value for a new parameter, named α, the number of bits required to represent a marker. Each symbol in the expression is preceded by a one-bit tag that identifies the type of symbol, for example, a zero for a marker and a one for a sequence letter. Following the tag is the representation of the symbol itself: α bits for markers, and $c(x)$ bits for a letter x using the cost function defined in the previous section.

The preamble of a message based on the tagged representation is much simpler: it only contains the single bit designating whether the first segment is a block or a variable region, the substitution matrix ID, and the value of α. The tagged representation of the alignment of the example sequences is shown in Figure 4(b).

3.3.3. *Scaled representation*

The additional bits attached to each symbol in the tagged representation result in a rather awkward code from an information theoretic point of view, where the number of bits used to represent a symbol should depend on the probability of observing that symbol.

In order to define the number of bits for each symbol s as $-\log_2 q(s)$, where s is either a sequence letter or a marker symbol, one can scale each element in the joint probability matrix by a constant factor $1-\gamma$ (where $0<\gamma<1$) and then define the number of bits in the representation of a marker as $\alpha=-\log_2(\gamma)$ (Figure 4(d)). Now the body of the message is simply the representation of each symbol, encoded according to the modified probability matrix (see also Figure 4(c)):

$$\begin{aligned} c(x) &= -\log_2 q(x), \\ c(y \mid x) &= -\log_2 q(y \mid x), \\ c(\cdot) &= -\log_2(\gamma). \end{aligned} \tag{6}$$

The preamble of a message encoded with the scaled representation is the same as the preamble for a tag-based message, except that the additional parameter is γ instead of α.

Since the probability of each single letter is the marginal probability summed over a row of the joint probability matrix, and each matrix entry was multiplied by a constant scale factor, the single-letter probabilities are also scaled by this same amount:

$$\begin{aligned} q(x) &= \sum_y (1-\gamma)p(x,y) \\ &= (1-\gamma)\sum_y p(x,y) = (1-\gamma)p(x). \end{aligned} \tag{7}$$

But note that conditional probabilities are not affected by the scaling since the scale factors cancel out:

$$\begin{aligned} q(y \mid x) &= \frac{q(x,y)}{q(x)} = \frac{(1-\gamma)p(x,y)}{(1-\gamma)p(x)} \\ &= \frac{p(x,y)}{p(x)} = p(y \mid x). \end{aligned} \tag{8}$$

Recall from Section 3.2 that a pair of letters will be included in a block if there is a positive benefit from aligning them, that is, if $c(y)-c(y \mid x)>0$. In the scaled representation, this calculation compares a cost based on a scaled probability with a cost defined by an unscaled probability. Since the scaled probabilities are lower than the original probabilities, the scaled costs of single letters are higher, and some letter pairs that had a negative benefit according to the original probabilities will now have a positive benefit. For example, in the PAM matrices, letter pairs with scores of 0 or higher have a positive benefit using unscaled probabilities, but when scaled with $1-\gamma=0.75$ pairs of slightly dissimilar amino acids with scores of -1 have a positive benefit.

3.4. *Example*

Two different alignments of the sequences of Figure 2 are shown in Figure 5. The alignments were made using the scaled representation with the PAM20 substitution matrix and $\gamma=0.02$. The code length for the null hypothesis—a single variable region containing all letters from the two productions—is 240.279 bits. The code length of the expression with two variable regions and one block is 224.728 bits. The cost of the expression with the block is less because the net benefit from using conditional probabilities to compute the costs of the aligned letters $(129.508-91.381=38.127$ bits) outweighs the cost of introducing four marker symbols $(4\times 5.644=22.576$ bits) for the boundaries of the block.

4. EXPERIMENTAL RESULTS

To evaluate the feasibility of aligning pairs of sequences by finding the minimum cost sequence expression, a simple graph search algorithm was developed and implemented in a program named `realign`. The algorithm creates a directed acyclic graph where nodes represent candidate blocks defined by equal-length substrings from each input sequence. Weights assigned to nodes represent the cost in bits of the corresponding block, and weights on edges connecting two nodes are defined by the cost of a variable region for the characters between the two blocks. The minimum cost path through the graph corresponds to the optimal alignment.

In one set of experiments, alignments produced by `realign` were compared to pairwise alignments generated by `CLUSTALW` [33], one of the most widely used alignment programs. In a second experiment, `realign` was used to align pairs of sequences from the BaliBase benchmark suite [34].

4.1. *Plasmodium orthologs*

An important concept in evolutionary biology is *homology*, defined to be similarity that derives from common ancestry. In molecular genetics, two genes in different organisms are said to be *orthologs* if they are both derived from a single gene in the most recent common ancestor.

In genome-scale computational experiments, a simple strategy known as "reciprocal best hit" is often used to identify pairs of orthologous genes. For each gene a from organism A, do a BLAST search [2] to find the gene b from organism B that is most similar to a. If a search in the other direction, using BLAST to find the gene most similar to b in

```
>MNNNNYIFENENPILYNTNEGEENRSS
>MNSYKPENENPVLYNYKEDEESSHI
```

Cost of null hypothesis:
$228.99 + 2\alpha = 240.279$ bits

(a)

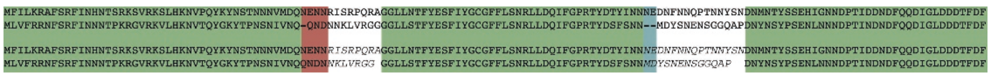
```
>MNNNNYIF
>MNSYKP
#ENENPILYNTNEGEE
#ENENPVLYNYKEDEE
>NRSS
>SSHI
```

$\sum c(x) + c(y)$ for letters in the block: 129.508 bits
$\sum c(x) + c(y|x)$ for the block: 91.381 bits

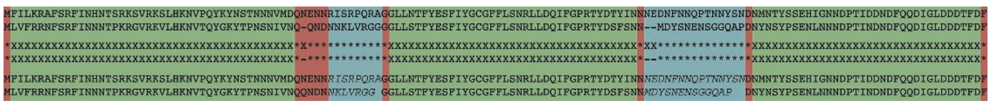
Cost of the expression with one block:
$64.272 + 91.381 + 35.211 + 6\alpha = 224.728$ bits

(b)

Figure 5: Cost of alternative expressions for the example sequences using the PAM20 substitution matrix and $\gamma = 0.02$. The cost for each marker symbol is $\alpha = -\log_2 \gamma = 5.644$ bits. (a) The cost for the null hypothesis is the sum of all the individual letter costs plus the cost of the two marker symbols. (b) When the letters in blue are aligned with one another, the costs of the letters in the second sequence are computed with conditional probabilities. This reduces the cost of the letters in the block by $129.508 - 91.381 = 38.127$ bits. The transformed grammar has four additional markers, but the reduction in cost afforded by using the block outweighs the cost of the new markers ($4 \times 5.644 = 22.576$ bits) so the expression with one block has a lower overall cost.



(a)



(b)

		Untrim	Trim
Aligned by both	🟩	0.473	0.469
Aligned by neither	🟦	0.147	0.258
CLUSTALW only	⬜	0.38	0.267
Realign only	🟥	<0.001	0.006

(c)

Figure 6: Alignment of sequences *MAL7P1.11* and *Pv087705* from ApiDB [35]. (a) Comparison of CLUSTALW alignment (top two lines of text) and the regular expression alignment (bottom two lines). Background colors indicate whether the two algorithms agree. Green: columns aligned by both algorithms; blue: letters not aligned by both algorithms; white: letters aligned by CLUSTALW but appearing in variable regions in the regular expression; red: letters aligned in the regular expression but not by CLUSTALW. (b) Same as (a), but comparing the trimmed CLUSTALW alignment with regular expression alignment. The middle row of two lines shows the result of the alignment trimming algorithm; an asterisk identifies a column from the CLUSTALW alignment that was removed by "gap expansion." (c) Proportion of each type of column averaged over all 3909 alignments.

organism A, reveals that a is most similar to b, then a and b are most likely orthologs.

Once pairs of genes are identified as reciprocal best hits, a more detailed comparison is done using a global alignment algorithm such as CLUSTALW [33]. To see how well the regular expression-based alignment algorithm performs on real sequences, a series of alignments of orthologous genes made with realign were compared to the CLUSTALW alignments of the same genes. The complete set of genes from *Plasmodium falciparum*, the parasite that causes malaria, and a close relative known as *Plasmodium vivax* were downloaded from ApiDB, the model organism database for this family of organisms [35]. A set of 3909 orthologs were identified by using BLAST to search for reciprocal best hits. Since *P. falciparum* diverged from *P. vivax* approximately 200 MYA [36], all the alignments used the PAM20 substitution matrix. The realign alignments were made using the scaled representation for marker symbols with $\gamma = 0.02$ since insertion and deletion events are relatively rare at this short evolutionary time scale.

Figure 6 shows a detailed comparison of the alignments for one pair of genes (*MAL7P1.11* and *Pv087705*). The top two lines in Figure 6(a) are the alignment produced by CLUSTALW, and the bottom two are the regular expression alignment. To make it easier to compare the alignments, the marker symbols have been deleted, and the letters in variable regions printed in italics to distinguish them from letters in blocks. The four background colors indicate the level of agreement between the two alignments: a pair can be aligned by both programs, aligned by neither, or aligned by one but not the other.

Researchers often apply an "alignment trimming" algorithm to the output of an alignment algorithm to identify suspect columns in an alignment [37]. An example of a suspect column is the one shown in Figure 1 where an insertion occurred in the middle of a codon. Figure 6(b) shows the alignment of the *Plasmodium* genes after an alignment trimming operation [38] was applied to the CLUSTALW alignments. The middle two lines in this figure show the results of the trimming application: an X indicates a letter that was left in the alignment, and a $\star$ indicates a position that was originally aligned but has now been converted to a gap. In this example, the alignment trimming algorithm agreed with the regular expression alignment: columns that were previously shown as aligned (white background color) are now unaligned (blue).

Over all the 3909 pairs of sequences, the two alignment methods agreed on 62% of the letters (top two rows of Figure 6(c)). The disagreement was almost entirely due to the fact that in 38% of the columns, the regular expression alignment was more conservative and placed characters in an unaligned region when CLUSTALW aligned those same letters. There are very few instances where `realign` put letters in an aligned block and CLUSTALW did not. Applying the alignment trimming algorithm increases the level of agreement: approximately one fourth of the columns originally considered aligned by CLUSTALW were reclassified as unaligned, in agreement with `realign`. The number of columns aligned only by `realign` also increased, but that is simply due to the fact that the alignment trimming algorithm used here [38] is very conservative and also trims away the last character in an aligned region (as shown by the red columns at the ends of blocks in Figure 6(b)).

These results show that for sequences with a high degree of similarity (separated by only 200MY of evolution), the MDL method implemented in `realign` does a credible job of global alignment. A more detailed analysis of genes with known alignments, preferably including structural and functional alignment, would be required to determine whether the 25% of the letter pairs aligned by CLUSTALW should in fact be aligned, or whether `realign` was correct in leaving them in variable regions.

4.2. BAliBASE reference alignments

The main parameter of the regular expression alignment method is the substitution matrix, which defines the probabilities for amino acid letters. A second parameter, the number of bits to use for a marker symbol or the probability associated with a marker symbol, is required if expressions are encoded with the tagged or scaled representations, respectively. To illustrate the effects of these parameters, an experiment evaluated the accuracy of `realign` alignments compared to known reference alignments from the BAliBASE [34] benchmark suite.

Sequences in BAliBASE are organized in a collection of different test sets. The sets were designed to provide different challenges to multiple alignment programs, for example, all sequences in a test are equally distant, or sequences are in two distinct subgroups. Sequences in each set have known 3D structures, and each test set was manually curated to identify conserved core blocks within each multiple alignment. The accuracy of an alignment algorithm can be assessed by comparing how it aligns amino acids in the core blocks. The comparisons reported here were made by aligning all pairs of sequences in each test set.

Figure 7 illustrates how the choice of a substitution matrix affects the accuracy of an alignment. The blocks in Figure 7(b) are from an alignment based on PAM20, and the blocks in Figure 7(c) are from the same pair of sequences aligned with PAM250. Letters shown in blue are accurate pairings of letters in core blocks in the reference alignment, and letters in red are misaligned—either they are placed in variable regions, or if they are in blocks, they are aligned with the wrong letter from the other sequence (e.g., the letters in the block marked with (2)). The overall accuracy is higher for the PAM250 alignment, which is not surprising since these two sequences are only about 40% identical, and sequences with this low level of similarity have probably diverged for much more than 200MY.

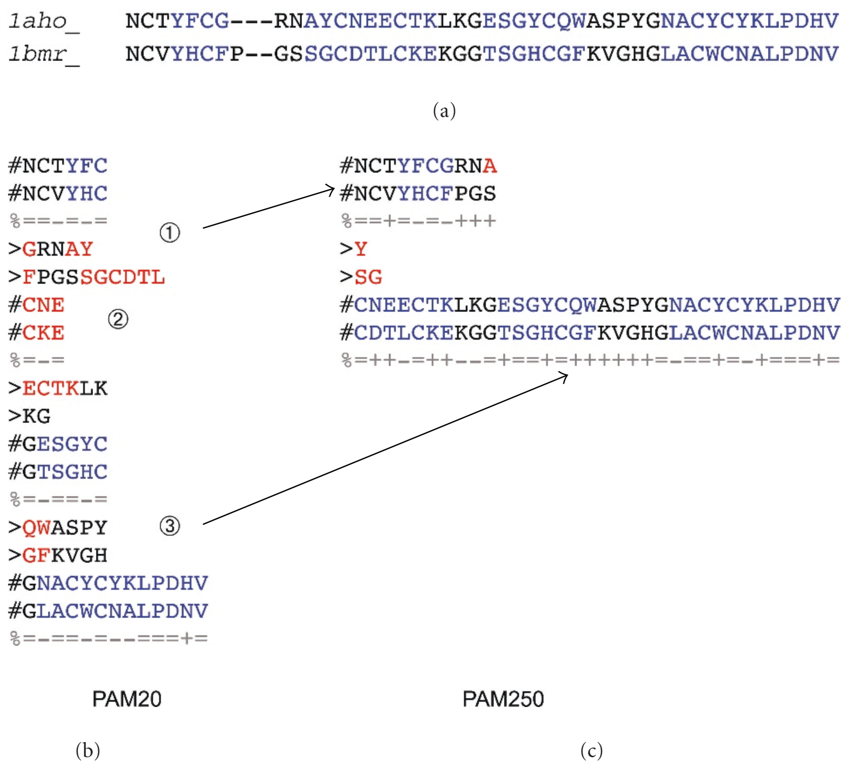
The block marked with a (3) in Figure 7 is an example of how a less strict substitution matrix leads to longer blocks. The letter pair Q and G are dissimilar in PAM20, and the block ends at this letter pair. But with PAM250, there is a slight benefit to aligning Q with G ($c(\mathtt{G}|\mathtt{Q}) < c(\mathtt{G})$) so these two letters are aligned.

Note that in the region indicated by (1) in Figure 7, the letters G and F still have a negative benefit with PAM250. But they are included in a longer block in the PAM250 alignment because they are surrounded on both sides by runs of similar letters, and it was less expensive for the algorithm to keep them in this block than to break them out into a short variable region.

Varying the alignment parameter that determines the number of bits used to represent a marker symbol also has an effect on accuracy. The longer sequences evolve, the more likely it is that an insertion or deletion mutation occurs in one or both sequences, and the regular expression alignment algorithm will need the flexibility to insert more marker symbols. When aligning pairs of sequences from BAliBASE, small values of α, either specified directly when the tagged representation is used or computed as $-\log_2\gamma$ for the scaled representation, yields the most accurate alignments.

Since the goal of the alignment algorithm is to find the sequence expression that can be represented in the fewest number of bits, a natural question is whether the algorithm should try to search for the value of γ that leads to the overall lowest cost expression. A related question, for sequences which have a known reference alignment, is whether the expression with the shortest encoding also corresponds to the most accurate alignment.

Unfortunately, the answers to these questions are not straightforward. The plots in Figure 8 show the results of a set of experiments that measure the effect of γ on the number



```
1aho_    NCTYFCG---RNAYCNEECTKLKGESGYCQWASPYGNACYCYKLPDHV
1bmr_    NCVYHCFP--GSSGCDTLCKEKGGTSGHCGFKVGHGLACWCNALPDNV
```

(a)

```
#NCTYFC                    #NCTYFCGRNA
#NCVYHC                    #NCVYHCFPGS
%==-=-=                    %==+=-=-+++
>GRNAY                     >Y
>FPGSSGCDTL                >SG
#CNE                       #CNEECTKLKGESGYCQWASPYGNACYCYKLPDHV
#CKE                       #CDTLCKEKGGTSGHCGFKVGHGLACWCNALPDNV
%=-=                       %=++-=++--=+==+=+++++++=-==+=-+===+=
>ECTKLK
>KG
#GESGYC
#GTSGHC
%=-===-=
>QWASPY
>GFKVGH
#GNACYCYKLPDHV
#GLACWCNALPDNV
%=-==-=--===+=
```

FIGURE 7: Portions of alignments of sequences *1aho* and *1bmr* from the BAliBASE alignment benchmark (Release 3) [34]. (a) The reference alignment from BAliBASE. Letters in core blocks are highlighted in blue. (b) Alignment from `realign`, using PAM20 and $\gamma = 0.2$. (c) Same as (b) but using PAM250. In (b) and (c) lines starting with % are comments that show the degree of similarity of corresponding letters in the preceding block: identical (=), similar (+), or dissimilar (−). Sequence letters in blue are correctly aligned core blocks. Red letters are core block column that should have been aligned but were left in variable regions. The circled numbers highlight changes in the alignment (see text).

of bits needed to encode a set of sequence expressions and the accuracy of the alignments. To make sure the alignment algorithm had enough data to work with, the alignments were done on the longest set of sequences in BAliBASE. There are eight sequences in this test set (BB12007), ranging in length from 994 to 1084 letters, with a mean length of 1020 letters. 28 pairwise alignments were created, using all possible pairs of sequences from the set.

Figure 8(a) shows that the number of bits required to represent an alignment increases as γ increases. There is a very slight decrease in cost near $\gamma = 0.02$. At smaller values of γ the cost of representing a marker symbol ($-\log_2 \gamma$) is too high for the algorithm to include any blocks. Near $\gamma = 0.02$, a few blocks are found and the overall cost is lowered. But as γ increases, the cost of the sequence letters increases, since they are scaled by a factor of $1 - \gamma$. There are typically far more letter symbols than marker symbols in a sequence expression, and the increase in the size of each letter outweighs any gain from a shorter representation for marker symbols.

One could argue that for a given value of γ, it is not the total size of a sequence expression that is important, but rather the amount of compression that results from that value of γ, where compression is the difference in the number of bits required to encode the null hypothesis (that the sequences have nothing in common) and the number of bits to encode the shortest sequence expression. Figure 8(b) shows a plot of the change in compression as a function of γ, where there is a peak in the range $0.07 \leq \gamma \leq 1.0$. Superimposed on this graph is a plot of the accuracy of the best alignment, also as a function of γ. The peak in this plot is an accuracy of 69%, at $\gamma = 0.05$.

The most accurate alignments, with a mean accuracy of 80%, were created using the tagged representation and very small values of α between 1.25 and 1.75 bits (including the tag bit). To obtain a comparable ratio between the cost of a marker symbol and sequence letter in the scaled representation γ would have to be around 0.25. But because the scaled representation requires the algorithm to compare letter probabilities scaled by $1 - \gamma$ with unscaled conditional probabilities, the accuracy deteriorates with higher values of γ. This distortion might be the reason the peak in the accuracy curve does not correspond more closely to the peak in the compression curve in Figure 8(b).

5. SUMMARY AND FUTURE WORK

This paper has shown that regular expressions provide useful descriptions of alignments of pairs of sequences. The expressions are simple concatenations of alternating blocks and variable regions, where blocks are equal-length substrings

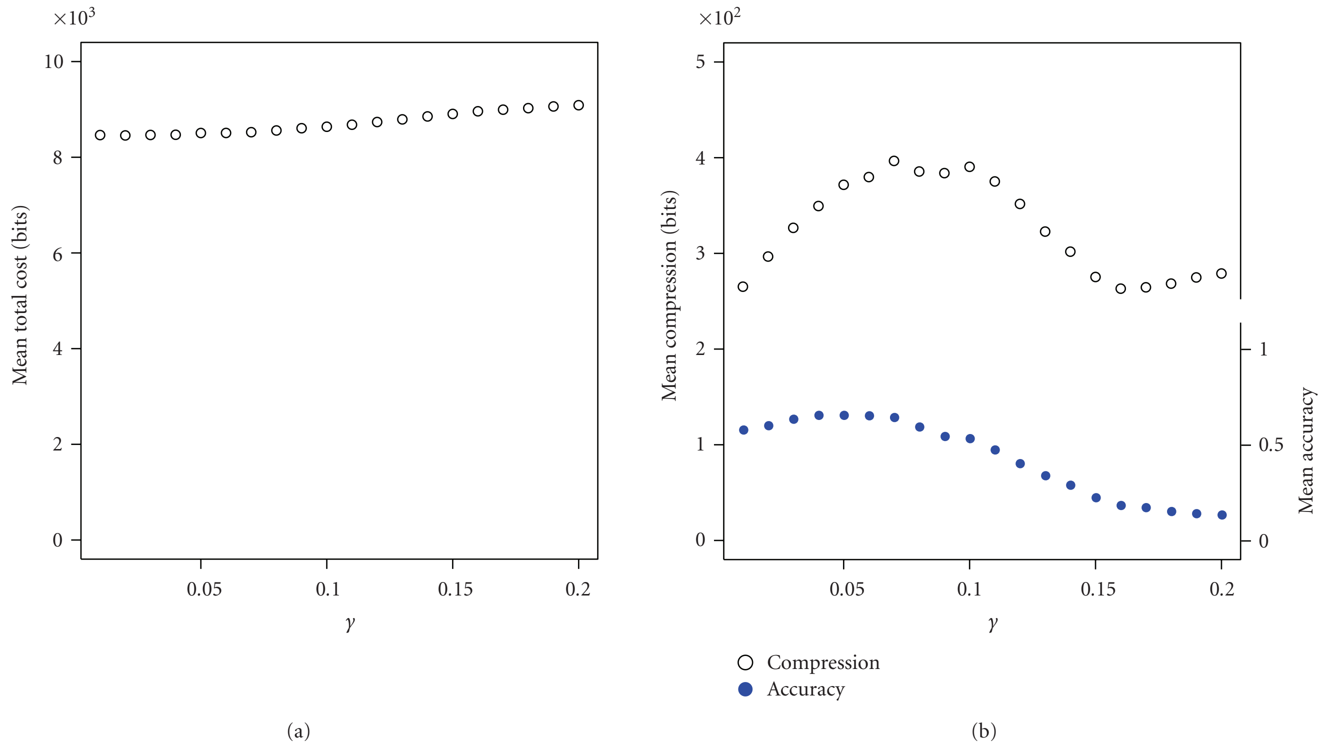


Figure 8: The effect of the scaling parameter γ on alignments of pairs of sequences from BAliBASE [34] test set BB12007. There are eight sequences in the set; the data points are based on averages over all $(8 \times 7)/2 = 28$ pairs of sequences. (a) Mean cost (in bits) of alignments as a function of γ. (b) Mean compression (the difference between the cost of the null hypothesis and the lowest cost alignment for each pair of sequences) is indicated by open circles. The mean accuracy of the alignments (proportion of core blocks correctly aligned) is indicated by closed circles (scale shown on the right axis).

from each input sequence and variable regions are strings of unaligned characters.

Alignment via regular expressions is an application of information theory: a hypothetical sender constructs a regular expression that describes the sequences, compresses the expression by encoding blocks with conditional probabilities, and transmits the encoded expression to a receiver, who can recover the original sequences by generating every string that matches the expression. The only parameter that is required is a substitution matrix, which sets the background probabilities for unaligned letters and the conditional probabilities for pairs of aligned letters. For greater flexibility, an optional second parameter specifies the number of bits to use for the marker symbols that denote block boundaries. This information theoretic framework does not use gaps to align variable-length sequences—instead a global alignment of sequences of different length will have at least one variable region with a different number of letters from the input sequences—and thus finesses issues associated with gap penalties.

Accurate alignment of biological sequences needs to take into account the amount of time the sequences have been changing since they diverged from their most recent common ancestor. The two parameters that affect the encoding of regular expressions—the choice of substitution matrix and the number of bits to use for marker symbols—are related to the two main types of mutations that can occur since the input sequences diverged. The substitution matrix is the basis for computing the probability of aligning pairs of letters, and generally reflects the probability that one of the letters changed via point mutation into the other letter. Marker symbols typically denote block boundaries that are the result of insertion or deletion mutations, and for very diverse sequences a smaller number of bits per marker reflect a higher probability of an insertion or deletion.

An alignment algorithm based on this approach can be seen as a process that begins with a default null hypothesis that the sequences are unrelated, represented by an expression that has all characters in a single unaligned region. The algorithm searches for candidate blocks, consisting of equal-length substrings from each input sequence, and checks to see if the encoding of an expression that includes a block is shorter than the encoding without the block. The tradeoff that must be taken into account is that blocks of similar letters will have denser encodings due to the use of conditional probabilities, but adding a block means increasing the number of marker symbols that denote the edges of blocks.

A comparison of this new method with CLUSTALW, a widely used standard for sequence alignment, shows that the regular expression alignments generally agree with CLUSTALW on regions included in blocks in the regular expression. Approximately, three quarters of the characters left unaligned in a regular expression are aligned by CLUSTALW, but that number drops to one half if the CLUSTALW alignments are treated with an "alignment trimming" algorithm to remove ambiguous regions. A more detailed case-by-case analysis would be required to determine if the remaining unaligned characters should remain unaligned (i.e., alignment trimming should be more ambitious) or if they need to be aligned (i.e., the regular expression approach is not aligning some characters that should be aligned).

A second set of experiments compared the output of the regular expression method with known reference alignments from the BAliBASE alignment benchmark. Since the benchmark is designed to test multiple alignment algorithms, and it is generally accepted that multiple alignment is more accurate than simple pairwise alignment [28], it is not possible to say whether the regular expression approach is as accurate as recent multiple alignment methods, but the overall accuracy of over 80% for sequences with 20% to 40% identity is encouraging.

One direction for future research is to try to automatically determine, for each substitution matrix, the best value for α or γ, the parameters that determine the number of bits per marker symbol. Based on extensive investigation (e.g., [39]) of different combinations of substitution matrix and other parameters BLAST, CLUSTALW, and other applications set default values for gap penalties based on the choice of substitution matrix. A similar analysis, perhaps based on insertion and deletion mutation rates, might be used to match a substitution matrix with a setting of α or γ for regular expression alignments.

A second direction for future research is to expand the method to perform multiple alignment of more than two sequences. One approach would be to use pairwise local alignments produced by `realign` as "anchors" for DIALIGN [22, 23], a progressive multiple alignment program that joins consistent sets of ungapped local alignments into a complete multiple alignment. A different approach would align all the sequences at the same time, using sum-of-pairs or some other method to average conditional costs based on each of the $n \times (n-1)/2$ pairs of sequences.

A third direction for future research is to extend the canonical sequence expressions or the equivalent grammar to include other forms of descriptions of regions of similarity. One idea is to use PROSITE blocks [40] as "subroutines" that can be embedded in blocks. For example, PROSITE block PS00007 is [RK]-x(2,3)-[DE]-x(2,3)-Y, using a notation similar to a regular expression where a string in brackets means "any one of these letters" and x(2,3) means "any sequence between 2 and 3 letters long." A string that matches this pattern, RDIKDPEY, occurs in one of the *Plasmodium* sequences discussed in Section 4.1. A block for the region containing this pattern might include a reference to the PROSITE block, for example, instead of

$$\texttt{\#DLLRDIKDPEYSYT} \tag{9}$$

the block would be something like

$$\texttt{\#DLL}\,\texttt{ps00007}\,(\texttt{R},\texttt{DIK},\texttt{D},\texttt{PE})\,\texttt{SYT}, \tag{10}$$

where the arguments to the procedure call are pieces of the sequence to plug in to the pattern. A benefit from using PROSITE or other predefined collections of patterns is that blocks can be encoded in fewer bits. Where the pattern specifies one of a small set of k letters, only $\log_2 k$ bits are required to encode one of these letters, assuming they are equally probable in this context. In particular, constants in the pattern require zero bits, since the receiver knows these letters as soon as the pattern is specified. A second benefit is that PROSITE blocks allow the expression to describe small amounts of variability in the length of a region without introducing a new variable region. Of course these benefits are offset by the additional complexity of an encoding that allows for rule names and parameter delimiters.

As the last example shows, regular expressions and grammars are very flexible, with many different rule structures able to describe the same set of sequences. The different rule structures convey different information about the strings generated by the grammars, and the goal will be to see if minimum description length encoding of these alternative structures and selection of the shortest encoding accurately provides the best description of the relationships between the sequences.

ACKNOWLEDGMENTS

The anonymous reviewers made several valuable comments. The indexed representation for marker symbols was suggested by one of the reviewers, and the scaled representation is due to Peter Grünwald. The author gratefully acknowledges support by grants from the National Science Foundation (MCB-0342431), the National Institutes of Health (5R01RR020833-02), and E.T.S. Walton Visitors Award from Science Foundation Ireland.

REFERENCES

[1] E. W. Myers, "The fragment assembly string graph," *Bioinformatics*, vol. 21, suppl. 2, pp. ii79–ii85, 2005.

[2] S. F. Altschul, T. L. Madden, A. A. Schaffer, et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[3] A. J. Phillips, "Homology assessment and molecular sequence alignment," *Journal of Biomedical Informatics*, vol. 39, no. 1, pp. 18–33, 2006.

[4] J. O. Wrabl and N. V. Grishin, "Gaps in structurally similar proteins: towards improvement of multiple sequence alignment," *Proteins*, vol. 54, no. 1, pp. 71–87, 2004.

[5] K. Sjölander, "Phylogenomic inference of protein molecular function: advances and challenges," *Bioinformatics*, vol. 20, no. 2, pp. 170–179, 2004.

[6] B.-J. M. Webb, J. S. Liu, and C. E. Lawrence, "BALSA: Bayesian algorithm for local sequence alignment," *Nucleic Acids Research*, vol. 30, no. 5, pp. 1268–1277, 2002.

[7] J. Rissanen, "Modelling by the shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[8] P. Grünwald, "A minimum description length approach to grammar inference," in *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, vol. 1040 of *Lecture Notes in Computer Science*, pp. 203–216, Springer, Berlin, Germany, 1996.

[9] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen, "Pattern discovery in biosequences," in *International Conference on Grammar Inference (ICGI '98)*, V. Honavar and G. Slutski, Eds., vol. 1433 of *Lecture Notes in Artificial Intelligence*, pp. 257–270, Springer, Ames, Iowa, USA, 1998.

[10] L. Cai, R. L. Malmberg, and Y. Wu, "Stochastic modeling of RNA pseudoknotted structures: a grammatical approach," *Bioinformatics*, vol. 19, suppl. 1, pp. i66–i73, 2003.

[11] D. B. Searls, "The computational linguistics of biological sequences," in *Artificial Intelligence and Molecular Biology*, pp. 47–120, American Association for Artificial Intelligence, Menlo Park, Calif, USA, 1993.

[12] D. Bsearls, "Linguistic approaches to biological sequences," *Computer Applications in the Biosciences*, vol. 13, no. 4, pp. 333–344, 1997.

[13] A. Bairoch, "PROSITE: a dictionary of sites and patterns in proteins," *Nucleic Acids Research*, vol. 20, pp. 2013–2018, 1992.

[14] M. Vingron and M. S. Waterman, "Sequence alignment and penalty choice. Review of concepts, case studies and implications," *Journal of Molecular Biology*, vol. 235, no. 1, pp. 1–12, 1994.

[15] S. Henikoff, "Scores for sequence searches and alignments," *Current Opinion in Structural Biology*, vol. 6, no. 3, pp. 353–360, 1996.

[16] G. Giribet and W. C. Wheeler, "On gaps," *Molecular Phylogenetics and Evolution*, vol. 13, no. 1, pp. 132–143, 1999.

[17] Y. Nozaki and M. Bellgard, "Statistical evaluation and comparison of a pairwise alignment algorithm that a priori assigns the number of gaps rather than employing gap penalties," *Bioinformatics*, vol. 21, no. 8, pp. 1421–1428, 2005.

[18] J. T. Reese and W. R. Pearson, "Empirical determination of effective gap penalties for sequence comparison," *Bioinformatics*, vol. 18, no. 11, pp. 1500–1507, 2002.

[19] L. Allison, C. S. Wallace, and C. N. Yee, "Finite-state models in the alignment of macromolecules," *Journal of Molecular Evolution*, vol. 35, no. 1, pp. 77–89, 1992.

[20] J. P. Schmidt, "An information theoretic view of gapped and other alignments," in *Proceedings of the 3rd Pacific Symposium on Biocomputing (PSB '98)*, pp. 561–572, Maui, Hawaii, USA, January 1998.

[21] T. Aynechi and I. D. Kuntz, "An information theoretic approach to macromolecular modeling: I. Sequence alignments," *Biophysical Journal*, vol. 89, no. 5, pp. 2998–3007, 2005.

[22] B. Morgenstern, "DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment," *Bioinformatics*, vol. 15, no. 3, pp. 211–218, 1999.

[23] M. Brudno, M. Chapman, B. Göttgens, S. Batzoglou, and B. Morgenstern, "Fast and sensitive multiple alignment of large genomic sequences," *BMC Bioinformatics*, vol. 4, p. 66, 2003.

[24] T. D. Schneider, "Information content of individual genetic sequences," *Journal of Theoretical Biology*, vol. 189, no. 4, pp. 427–441, 1997.

[25] N. Krasnogor and D. A. Pelta, "Measuring the similarity of protein structures by means of the universal similarity metric," *Bioinformatics*, vol. 20, no. 7, pp. 1015–1021, 2004.

[26] J. S. Conery, "Realign: grammar-based sequence alignment," University of Oregon, http://teleost.cs.uoregon.edu/realign.

[27] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "A model of evolutionary change in proteins," in *Atlas of Protein Sequence and Structure*, vol. 5, suppl. 3, pp. 345–352, Washington, DC, USA, 1978.

[28] D. W. Mount, *Bioinformatics: Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, New York, NY, USA, 2nd edition, 2004.

[29] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 89, no. 22, pp. 10915–10919, 1992.

[30] G. H. Gonnet, M. A. Cohen, and S. A. Benner, "Exhaustive matching of the entire protein sequence database," *Science*, vol. 256, no. 5062, pp. 1443–1445, 1992.

[31] S. Karlin and S. F. Altschul, "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 87, no. 6, pp. 2264–2268, 1990.

[32] S. R. Eddy, "Where did the BLOSUM62 alignment score matrix come from?" *Nature Biotechnology*, vol. 22, no. 8, pp. 1035–1036, 2004.

[33] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Research*, vol. 22, no. 22, pp. 4673–4680, 1994.

[34] J. D. Thompson, F. Plewniak, and O. Poch, "A comprehensive comparison of multiple sequence alignment programs," *Nucleic Acids Research*, vol. 27, no. 13, pp. 2682–2690, 1999.

[35] C. Aurrecoechea, M. Heiges, H. Wang, et al., "ApiDB: integrated resources for the apicomplexan bioinformatics resource center," *Nucleic Acids Research*, vol. 35, pp. D427–D430, 2007.

[36] R. Carter, "Speculations on the origins of Plasmodium vivax malaria," *Trends in Parasitology*, vol. 19, no. 5, pp. 214–219, 2003.

[37] M. Cline, R. Hughey, and K. Karplus, "Predicting reliable regions in protein sequence alignments," *Bioinformatics*, vol. 18, no. 2, pp. 306–314, 2002.

[38] J. S. Conery and M. Lynch, "Nucleotide substitutions and the evolution of duplicate genes," in *Proceedings of the 6th Pacific Symposium on Biocomputing (PSB '01)*, pp. 167–178, Big Island of Hawaii, Hawaii, USA, January 2001.

[39] W. R. Pearson, "Comparison of methods for searching protein sequence databases," *Protein Science*, vol. 4, no. 6, pp. 1145–1160, 1995.

[40] N. Hulo, A. Bairoch, V. Bulliard, et al., "The PROSITE database," *Nucleic Acids Research*, vol. 34, pp. D227–D230, 2006.