Algorithms for bioinformatics

Giacomo Fantoni
telegram: @GiacomoFantoni
Ilaria Cherchi
telegram: @ilariacherchi
Elisa Pettinà
telegram: @elisapettina
Alessandro Polignano
telegram: @alepolignano

Github: https://github.com/giacThePhantom/algorithms-for-bioinformatics

June 24, 2022

Contents

Chapter 1

Needleman Wunsch

1.1 Introduction

Direct comparison of two sequences based on the presence in both of the corresponding amino acids in an identical array is insufficient to establish the full genetic relationship between two proteins. Allowance for gaps multiplies the number of comparisons that can be made but introduces unnecessary and partial comparisons.

1.2 A general method for sequence comparison

The maximum match can be defined as the largest number of amino acids of one protein that can be matched with those of another protein while allowing for all possible deletions. It can be determined by representing in a matrix all possible pair combinations that can be constructed from the amino acid sequences of the protein being compared. So A_j is the jth amino acids of protein A and B_i is the ith amino acids of protein B. A_j are the columns and B_i all the rows of the matrix MAT. Then A_{ij} represent a pair combination with amino acids A_j and B_i. Every possible comparison can be represented by pathway through the matrix. A pathway is signified by a line connecting cells of the array. Complete diagonals contain no gaps. A necessary pathway begins at a cell in the first column of row. Either i or j must increase by only one, while the other may increase by one or more, leading to the next cell in a pathway. This is repeated until i, j or both reach their limiting value. Every partial or unnecessary pathway will be contained in at least one necessary pathway. The values in the matrix are computed as:

$$MAT_{ij} = \max(MAT_{i-1,j-1} + \alpha\delta(A_j, B_i), MAT_{i-i,j} + d, MAT_{i,j-1} + d)$$

Where d is the penalty factor, a number subtracted for every gap made, may be defined as a barrier for allowing the gap. And α can be a function that can represent any theory with the significance of a pair of amino acids. No gap would be allowed in the operation unless the benefit from allowing that gap would exceed the barrier. This method can be expanded for allowing the comparison of n sequences through and n-dimensional matrix. The maximum-match pathway can be obtained by beginning at the terminals of the sequences and proceeding towards the origin, first by adding to the value of each cell possessing indices $i = y - 1$ and or $j = z - 1$. The process is iterated until all cells in the matrix have been operated upon. Each cell in the outer row or column will contain the maximum number of matches that can be obtained by originating any pathway at

that cell and the largest number in that row or column is equal to the maximum match. The cells of the array which contributed to the maximum match may be determined by recording the origin of the number that was added to each cell when the array was operated upon.

1.3 Evaluating the significance of the maximum match

To accomplish the estimate of if a result found differs significantly from a match between random sequences two sets of random sequences can be constructed, each one from the set of amino acid composition of each of the proteins. If the value found for the real proteins is significantly different the difference a function of of the sequences alone and not of the composition.

1.4 Cell values and weighting factors

Cells can be weighted in accordance with the maximum number of corresponding bases in codons of the represented amino acids, to make the comparison more accurate. Also the significance of the maximum match is enhanced by decreasing the weight of those pathways containing a large number of gaps through the penalty factor.

Chapter 2

Smith Watermann

2.1 Introduction

The Smith Watermann algorithm extends the one of Needleman and Wunsch to find a pair of segment, one from each of two long sequences, such that there is no other pair of segments with greater similarity. This similarity measure allows for deletion and insertion of arbitrary length.

2.2 Algorithm

Consider two molecular sequences $A = a_1a_2 \dots a_n$ and $B = b_1b_2 \dots b_m$. Given a similarity $s(a, b)$ of elements of the sequence and W_k the weight of deletions of length k, to find pairs of segments with high degrees of similarity, a matrix H is set up such that:

$$H_{k0} = H_{0l} = 0 \qquad \forall 0 \leq k \leq n \wedge 0 \leq l \leq m$$

H_{ij} is the maximum similarity of two segments ending in a_i and b_j. H_{ij} is computed such that:

$$H_{ij} = \max(H_{i-1,j-1} + s(a_i, b_j), \max_{k \geq 1}(H_{i-k,j} - W_k), \max_{l \geq 1}(H_{i,j-l} - W_k), 0)$$

With $1 \leq i \leq n$ and $1 \leq j \leq m$. So H_{ij} is:

- $H_{i-1,j-1} + s(a_i, b_j)$ If a_i and b_j are associated.
- $H_{i-k,j} - w_k$ if a_i is at the end of a deletion of length k.
- $H_{i-k,j} - W_l$ if b_j is at the end of a deletion of length l.
- 0 is used to prevent calculated negative similarity, indicating no similarity up to a_i and b_j.

The pair of segments with maximum similarity is found first by locating the maximum element of H. The other elements are determined sequentially with a traceback procedure ending with an element of H equal to 0. This procedure other than identifying the elements produces their alignment. The parameters where:

$$s(a_i, b_j) = \begin{cases} 1 & a_i = b_j \\ 0 & a_i \neq b_j \end{cases}$$

And

$$W_k = \frac{1}{3}k$$

This algorithm in particular allows for the alignment of sequences that contained both mismatches and internal deletions.

Chapter 3

PAM - a model of evolutionary change in proteins

3.1 Accepted point mutation

An accepted point mutation in a protein is a replacement of one amino acid by another accepted by natural selection. To be accepted the new amino acid usually must function in a similar way to the old one. The likelihood of amino acid X replacing Y is the same as Y replacing X is assumed the same because it depends on the product of the frequencies of occurrence and on their chemical and physical similarity. So evolution is a vibration around given frequencies.

3.2 Mutability of amino acids

The relative mutability is the probability that each amino acid will change in a given small evolutionary interval. To compute it the number of times that each amino acid has changed in an interval and the number of times that it has occurred in the sequences and thus has been subject to mutation must be recorded. In calculating this number in for many trees, with sequences of different lengths and evolutionary distance is combined in relative mutabilities. Each relative mutability is a ratio between the total number of changes on all branches of all protein trees considered and the total exposure of the amino acid to mutation, or the sum for all branches of its local frequency of occurrence multiplied by the total number of mutation per 100 links of that branch.

3.3 Mutation probability matrix for the evolutionary distance of one PAM

The individual kind of mutations and the relative mutability of the amino acids can be combined into a mutation probability matrix in which M_{ij} gives the probability that the amino acid in column j will be replaced by the amino acid in row i after a given evolutionary period. The non-diagonal elements are computed as:

$$M_{ij} = \frac{\lambda m_j A_{ij}}{\sum_i A_{ij}}$$

Where:

- A_{ij} is an element of the accepted point mutation matrix.
- λ is a proportionality constant.
- m_j is the mutability of the jth amino acid.

The diagonal elements are:

$$M_{jj} = 1 - \lambda m_j$$

The sum of all elements of each column or row is 1. The probability of observing a change is proportional to the mutability of the amino acid in that place. The same proportionality constant λ holds for all columns. $100 \cdot \sum f_i M_{ij}$ gives the number of amino acids that will remain unchanged when a protein 100 links long of average composition is exposed to the evolutionary change. This depends on λ. To change the evolutionary period the matrix is multiplied my itself n times, and with $n \to \infty$ each column approaches the asymptotic amino acid composition. The percentage of amino acids that will be observed to change on the average in the interval are found by:

$$100(1 - \sum_i f_i M_{ij})$$

The term of the relatedness odds matrix are:

$$R_{ij} = \frac{M_{ij}}{f_i}$$

Or the mutation probability of a change over the probability that i will occur in the second sequence by chance. Each term of this matrix gives the probability of replacement per occurrence of i per occurrence of j. Amino acids with score > 1 replace each other more often as alternative in related sequences that in random sequences.

Chapter 4

BLOSUM

4.1 Introduction

4.1.1 Abstract

The most used substitution matrix with scores for all possible exchanges of one amino acid is based on the Dayhoff model of evolutionary rates. This work proposes a different approach from blocks of aligned sequence segments, leading to improvement in alignments and in searches.

4.1.2 Introduction

Sequence alignment of proteins provide important insights into gene and protein function. There are different types of alignments:

- Global alignments of pairs related by common ancestry.
- Multiple alignments of members of protein families.
- Alignments made during data base searches to detect homology.

In each case a scoring scheme for estimating similarity is used. The mutation data matrices of Dayhoff are considered the default in alignment and searching programs. However the most common task is the detection of much more distant relationships, which are only inferred from substitution rates in the Dayhoff model.

4.2 Methods

4.2.1 Deriving a frequency table from a data base blocks

Local alignments can be represented as ungapped blocks with each row a different protein segment and each column an aligned residue position. Protomat can be used for obtaining a set of blocks given a group of related proteins. Considering a single block representing a conserved region of a protein family, for a new member a set of score for matches and mismatches that best favours a correct alignment with each of the other segments in the block. For each column of the block, the number of matches and mismatches of each type between the new sequence and every other are

counted. This is repeated for all columns of all blocks and the summed results are stored in a table. The new sequence is then added to the group. For another new sequence the procedure is repeated. Doing so the table in the end will consist of counts of all possible amino acid pairs in a column. Counts of all possible pairs in each column of each block in the data base are summed. If a block has a width w amino acids and a depth of s sequences, contributes $ws\frac{(s-1)}{2}$ amino acids pairs to the count. This results in a frequency table listing the number of times each of the different amino acid pairs occurs among the blocks. This table is used to compute the odds ratio matrix between the observed frequencies and the expected one.

4.2.2 Computing a logarithm of odds matrix

Let the total number of amino acid pairs i, j for each entry of the frequency table be f_{ij}. Then the observed probability of occurrence for each i, j pair is:

$$q_{ij} = \frac{f_{ij}}{\sum_{i=1}^{20}\sum_{j=1}^{i} f_{ij}}$$

The expected probability of occurrence for each pair is computed following the occurrence of the ith amino acid in a i, j pair:

$$p_i = q_{ii} + \sum_{j \neq i} \frac{q_{ij}}{2}$$

The expected probability of occurrence e_{ij} for each i, j pair is then $p_i p_j$ for $i = j$ and $p_i p_j + p_j p_i = 2p_i p_j$ for $i \neq j$. An odds ratio matrix is computed such that each entry is $\frac{q_{ij}}{e_{ij}}$. A lod (logarithm of odds) is then calculated in bit units as $s = \log_2\left(\frac{q_{ij}}{e_{ij}}\right)$. If the observed frequencies are as expected $s_{ij} = 0$, if less $s_{ij} < 0$, if more $s_{ij} > 0$. Lod ratios are multiplied by a 2 scaling factor and rounded to the nearest integer value to produce the block substitution matrix BLOSUM. The relative entropy or the average mutual information per amino acid pair is computed:

$$H = \sum_{i=1}^{20}\sum_{j=1}^{i} q_{ij} \times s_{ij}$$

And the expected score in bit units:

$$E = \sum_{i=1}^{20}\sum_{j=1}^{i} p_i \times p_j \times s_{ij}$$

4.2.3 Clustering segments within blocks

To reduce multiple contributions to amino acid pair frequencies from the most closely related members of a family, sequences are clustered within blocks and each cluster is weighted as a single sequence in counting pairs. A clustering percentage in which sequence segments identical for at least that value are clustered is used. The contribution of closely related segments to the frequency table is reduced. Varying the clustering percentage leads to a family of matrices.

4.2.4 Constructing blocks data bases

Protomat was used to build the block data base from 504 non redundant groups of proteins. Protomat uses an amino acid substitution matrix at two phases. The motif program uses a substitution matrix when individual sequences are aligned against sequence segments containing a candidate motif. The Motomat program uses a substitution matrix when a block is extended to either side of the motif region. A unitary substitution matrix was used, next blosum was applied to the blocks and the resulting matrix was used to construct a second database. Then blosum was applied to the second data base and the resulting matrix was used to construct version of blocks data base. The blosum program was applied to the final data base using a series of clustering percentages to obtain a family of lod substitution matrices. Similar matrices were obtained using PAM.

4.2.5 Alignments and homology searches

Global multiple alignments were done using Multalin and to provide a positive matrix each entry was increased by 8. Pearson's RDF2 program was used to evaluate local pairwise alignments. Homology searches were done using Blastp, Fasta and Ssearch. The Swiss-Prot data bank was searched. The first of the longest and most distance sequences in the group was used as a searching query, inferring distance from Protomat results. The results of each search were analysed by considering the sequences used by Protomat to construct blocks for the protein group as the true positive sequences. The number of misses is the nubmer of true positive sequences not reported for blastp. For fasta and ssearch the empirical evaluation criteria of Pearson was used: the number of misses is the number of true positive scores which ranked below the 99.5 percentile of the true negative scores.

4.3 Results

4.3.1 Comparison to Dayhoff matrices

The blosum series based on percent clustering can be compared to the Dayhoff matrices using a measure of average information per residue pair in bit units or relative entropy, which is 0 when the target distribution of pair frequencies is the same as the expected one and increases as they become more different. In the Dayhoff matrices relative entropy decreased when increasing PAM, while in blosum it increased linearly when increasing clustering percentage. Matrices with comparable relative entropy have similar expected scores.

4.3.2 Performance in multiple alignment of known structures

To test sequence alignment accuracy the results obtained to alignments seen in three dimensional structures was used. A simultaneous multiple alignment program MSA was used as a standard. Multalin, a hierarchical multiple alignment program performed worse using PAM matrices, while using blosum better. Therefore blosum matrices produced accurate global alignments of these sequences.

4.3.3 Performance in searching for homology in sequence data banks

The number of misses when searching was averaged in order to assess the overall searching performance of different matrices using blast, fasta and smith-waterman. Blosum matrices performed better than the best PAM matrix. BLOSUM improved detection of members of this family. The test was repeated with similar result of PAM for other protein families.

Chapter 5

FASTA

5.1 Introduction

5.1.1 Abstract

FASTA is a derivative of FASTP and can be used to search protein or DNA sequence data bases and compare a protein sequence to a DNA one by translating the DNA one as it is searched. FASTA includes an additional step in the calculation of the initial pairwise similarity score that allows multiple regions of similarity to be joined to increase the score of related sequences. RDF2 can be used to evaluate the significance of similarity score with a shuffling method that preserves local sequence composition. LFASTA can display all the regions of local similarity with scores greater than a threshold that can be displayed as a graphic matrix plot.

5.1.2 Introduction

There is a trade-off between sensitivity and selectivity in biological sequence comparison: methods that can detect more distantly related sequences (increased sensitivity) frequently increase the similarity scores of unrelated sequences (decreased selectivity). FASTA use an improved algorithm that increases sensitivity with a small loss of selectivity and a negligible decrease in speed. LFASTA compute local similarity analyses of DNA or amino acid sequences.

5.2 Methods

5.2.1 The search algorithm

The search algorithm has four steps in determining a score for pair-wise similarity.

5.2.1.1 First step - lookup table

FASTA achieve its speed and selectivity in this step using a lookup table to locate all identities or groups of identities between two sequences. The *ktup* parameter determines how many consecutive identities are required in a match. The 10 best diagonal regions are found using a simple formula based on the number of *ktup* matches and the distance between the matches without considering shorter runs of identities, conservative replacements, insertions or deletions.

5.2.1.2 Second step - rescoring

In the second step the 10 regions are rescored using a scoring matrix that allows conservative replacements and runs of identities shorter than *ktup* to contribute to the similarity score. For each of these best diagonal regions, a subregion with minimal score is identified. This is the initial region.

5.2.1.3 Third step - ranking

The best scoring initial region is used to characterize pair-wise similarity and the initial scores are used to rank the library sequences. FASTA checks if several initial regions may be joined together. Given the location, their scores and a joining penalty, FASTA calculates an optimal alignment of initial regions as a combination of compatible regions with maximal score. The resulting score is used to rank the library sequences. Only for regions whose scores are above a threshold are used in this process.

5.2.1.4 Fourth step - alignment

The highest scoring library sequences are aligned by a modification of the Needleman-Wunsch and Smith-Waterman. The final comparison considers all possible alignments of the query and library that fall within a band centered around the highest scoring initial region.

5.2.2 Local similarity analyses

The program for detecting local similarities, LFASTA uses the same two steps for finding initial regions. Instead of saving 10, all diagonal regions with scores greater than a threshold are saved. Instead of focusing on a single region, LFASTA computes a local alignment for each initial region. It considers all initial regions and potential sequence alignments for some distance before and after the initial region. Starting at the end of the initial region, an optimization proceeds in the reverse direction until all possible alignment scores have gone to zero. The maximal local similarity score in the reverse direction is used to start a second optimization that proceeds in the forward direction. An optimal path starting from the forward maximum is displayed. The local homologies can be displayed as sequence alignments or on a two-dimensional graphic matrix style. The maximal local similarity score in the reverse direction is used to start a second optimization that proceeds in the forward direction. An optimal path starting from the forward maximum is displayed. The local homologies can be displayed as sequence alignments or on a two-dimensional graphic matrix style plot.

5.2.3 Statistical significance

To evaluate the statistical significance of an alignment RDF2 is used. RDF2 calculates three scores for each shuffled sequence: one from the best single initial region, a second from the joined initial regions and a third from the optimized diagonal. RDF2 can be used to evaluate amino acid or DNA sequences and allows to specify the scoring matrix. Moreover a global or local shuffle routine can be specified. Locally biased composition is the most common reason for high similarity scores of dubious biological significance. A Monte Carlo shuffle analysis that constructs random sequences by taking each residue in one sequence and placing it randomly along the length of the new sequence will break up these patches of biased composition. The scores may be lower and the sequences will appear to be related.

Chapter 6

BLAST - Basic local alignment search tool

6.1 Introduction

6.1.1 Abstract

Basic local alignment search tool is a new approach to rapid sequence comparison. It approximates alignments that optimize a measure of local similarity. The basic algorithm is simple and robust and can be implemented in a number of ways and applied in a variety of context.

6.1.2 Introduction

Classical alignment tools minimize the evolutionary distance or maximise the similarity between two compared sequences. The cost of this alignment is a measure of similarity. Dynamic programming algorithms are unfeasible due to the large searching space. Rapid heuristic algorithms can make the search faster, but the measure of similarity is not explicitly defined, like in the FASTP program. This approach however have been useful in identifying many distant but biologically significant relationships. BLAST employs a measure based on well-defined measure scores approximating the results obtained by dynamic programming, creating a process an order of magnitude faster than existing heuristic algorithms.

6.2 Methods

6.2.1 The maximal segment pair measure

Sequence similarity measures can be classified as global (the overall alignment is optimized, including large stretches of low similarity) or local (relatively conserved subsequences). Local measures are preferred for database searches, many of which begin with a matrix of similarity scores for all possible pairs of residues. Identities and conservative replacements have positive scores, while unlikely replacements have negative scores. For amino acid sequence comparisons PAM-120 is used. A sequence segment is a contiguous stretch of residues of any length and the similarity scores for two aligned segments of the same length is the sum of the similarity values for each pair of residues in them.

6.2.2 Maximal segment pair

A maximal segment pair MSP is defined as the highest scoring pair of identical length segments chosen from 2 sequences. The boundaries are chosen as to maximise its score. This BLAST attempts to heuristically calculate, providing a measure of local similarity for any pair of sequences. A segment pair si locally maximal if its score cannot be improved either by extending or shortening both segments. BLAST can seek all locally maximal segment pairs with scores above a cutoff.

6.2.2.1 Tractability to mathematical analysis

The MSP score may be computed in time proportional to the product of their lengths using a dynamic programming algorithm. The statistical significance of the MSP can be estimated under a random sequence model. Moreover for any particular scoring matrix, the frequencies of paired residues in maximal segments can be estimated.

6.2.3 Rapid approximation of MSP scores

Because only a handful in a database of thousands of sequences will be homologous to the query, only the sequence entries with MSP scores over some cutoff score S are of interest. These include those sharing highly significant similarity with the query and some borderlines. The biological significance of high scoring sequences may be inferred on the bases of the similarity score.

6.2.3.1 MSP score estimates

The highest MSP score S at which chance similarities are likely to appear can be estimated. BLAST minimizes the time spent on sequence regions whose similarity with the query has little chance of exceeding this score. Let a word pair be a segment pair of fixed length w. The main strategy is to seek only segment pairs that contain a word pair with a score of at least T. Scanning through a sequence, whether it contains a word of length w that can pair with the query with a score greater or equal to the threshold T can be quickly assessed. Any such hit is extended to determine if it is contained within a segment pair whose score is greater than or equal to S. The lower T, the greater the chance that a segment pair with a score of at least S will contain a word pair with a score of at least T. A small value for T increase the number of hits and the execution time. Random simulation permits to select a threshold T that balances execution time and precision.

6.2.4 Implementation

The algorithm consists of three steps: compiling a list of high-scoring words, scanning the database for hits and extending hits. This vary on whether the database contains proteins or DNA. For proteins the list consists of all words or $w-mers$ that score at least T when compared to some word in the query sequence: a query word may be represented by no words in the list. For useful results 50 words in the list for every residue in the query sequence.

6.2.4.1 Scanning phase

The scanning phase consists of the search of a long sequence for all occurrences of certain short sequences.

6.2.4.1.1 First approach The first approach consist of mapping each word to an integer, so a word can be used as an index into an array. The *ith* entry of such an array points the list of all occurrences in the query sequence for the *ith* word. Each database word leads immediately to the corresponding hits.

6.2.4.1.2 Second approach The second approach uses a deterministic finite automaton or finite state machine: signal acceptance is done on transitions. This saved a factor in space and time proportional to the size of the alphabet. This yielded a program that ran faster.

6.2.4.2 Hit extension

Extending a hit to find a locally maximal segment pair containing it is straightforward. The process is terminated in one direction when a segment pair whose score falls a certain distance below the best score for shorter extension. The added inaccuracy is negligible.

6.2.4.3 DNA alignment

For DNA a simpler word list is used. A query sequence of length n yields a list of $n - w + 1$ words. The database is compressed packing 4 nucleotides in a single byte using an auxiliary table to delimit the boundaries. Assuming $w \geq 11$ each hit must contain an 8-mer hit that lies on a byte boundary. This can allow to scan the database byte-wise and increase the speed 4-fold.

6.2.4.4 Dealing with locally biased composition and repeated sequence element

To deal with biased composition when the database is compressed tabulates of the frequencies of all 8-tuples are produced. Those occurring more frequently are stored and used to filter uninformative words from the query. A search of sublibrary of repetitive elements is performed and the locations in the query of significant matches are stored. Words generated by these regions are removed from the query word list. Matches to the sublibrary are reported in the final outpuit.

6.3 Results

6.3.1 Performance of BLAST with random sequences

Given a set of probabilities for the occurrence of individual residue, λ and K are used to evaluate the statistical significance of MSP scores. When two random sequences of lengths m and n are compared, the probability of finding a segment pair with a score greater or equal to S is:

$$1 - e^{-y}$$

Where $y = Kmne^{-\lambda S}$. Using this formula the approximate score an MSP must have to be distinguishable from chance similarities found in this database can be computed. The probability of finding c or more distinct segment pairs, all with a score of at least S is:

$$1 - e^{-y} \sum_{i=1}^{c-1} \frac{y^i}{i!}$$

Two sequences that share several distinct regions can be detected as significantly related.

6.3.1.1 Distribution of high-scoring segment pairs

It is of interest to know what proportion of segment pairs with a given score contain a word pair of length w with a score of at least T. For MSP arising from the comparison of random sequences a limiting distribution is provided. The probability q that a segment pair will fail to contain a word pair with score T should depend exponentially upon the score of the MSP. Because the frequencies of paired letters in MSP approaches a limiting distribution, the expected length of an MSP grows linearly with its score. The longer an MSP the more independent chances it has for containing a word with a score of at least T, implying that q should decrease exponentially with increasing MSP score S.

6.3.2 The choice of work length and threshold parameters

6.3.2.1 Time required to execute

The time required to execute BLAST is the sum of the times required to:

- Compile a list of words that can score at least T when compared with words from the query.
- Scan the database for hits.
- Extend all hits to seek segment pairs with scores exceeding the cutoff.

The time for the last is proportional to the number of hits, which depends on w and T.

6.3.2.2 Choice of w

Given a random protein model and a set of substitution scores it is simple to calculate the probability that two random words of length w will have a score of at least T, or the probability of a hit arising from an arbitrary pair of words in the query and database. For a given level of sensitivity (chance of missing a MSP), w can be chosen such that it minimizes the chance of a hit. The probability of a hit decrease for increasing w for different levels of sensitivity. Maintaining a given level of sensitivity the time spent on the third step can be decreased increasing w. But this will increase time spent on step 1 as it increase the number of words generated and the memory requirement. For protein searches the best compromise is a word size of four.

6.3.2.3 Choice of T

Reducing T improves the approximation of MSP scores, but it increases execution time: more words are generated by the query sequence and there will be more hits. The number of words generated increases exponentially with decreasing T.

6.3.2.4 Expected time complexity of BLAST

The expected time computational complexity of BLAST is:

$$aW + bN + \frac{cNW}{20^w}$$

Where W is the number of words generated, N is the number of residues in the database and a, b and c are constants. The W term accounts for compiling the word list, N covers the database scan and NW for the extension. Although W increases exponentially with T, it increases linearly with the length of the query. $T = 17$ is a good choice for the threshold.

6.3.2.5 Trade-off between accuracy and speed

Given a specific probability q of missing a MSP with score S, the T required is computed and so the execution time.

6.3.3 Performance of BLAST with homologous sequences

BLAST's great utility is for finding high-scoring MSP quickly. The overall performance depend upon the distribution of MSP scores for those sequences related on the query. The bulk of the MSP that are distinguishable from chance have a high enough score to be found readily even using relatively high values for T. In each instance the threshold S is chosen to include scores in the borderline region.

6.3.4 Comparison of two long DNA sequences

With smaller values of w more alignments are found. The use of a smaller w provides no new information.

Chapter 7

Gapped-BLAST and PSI-BLAST

7.1 Abstract

Gapped-BLAST uses a new criterion for triggering the extensions of word hits with a new heuristic for generating gapped alignments and it runs at approximately three times the speed of the original. In addition, a method is introduced for automatically combining statistically significant alignments produced by BLAST into a position-specific score matrix, and searching the database using this matrix. The resulting Position-Specific Iterated BLAST (PSI-BLAST) program runs at approximately the same speed per iteration as gapped BLAST, but in many cases is much more sensitive to weak but biologically relevant sequence similarities.

7.2 Gapped BLAST algorithm

7.2.1 Quick BLAST recap

- First, scan the database for words (three amino acids) with a score above a threshold T. An aligned word pair is called a hit.
- Then, to check if the hit lies within an alignment with a score high enough to be reported, extend the hit in both directions, until the alignment score has dropped below a threshold X.

7.2.2 Refinement of the original algorithm: two hits method

The extension step is the computationally heavy task that takes 90% of BLAST's execution time: the refinements is based on the observation that an High-scoring Segment Pair (HSP) of interest is usually longer than a single word pair and may contain multiple hits on the same diagonal within a relatively short distance (defined as the distance between their first coordinates).
So, an UNGAPPED extension is then triggered only when:two non-overlapping hits are found within distance of length A of one another on the same diagonal and any hit that overlaps the most recent one is ignored. Because we require two hits rather than one to invoke an extension, the threshold parameter T must be lowered to retain comparable sensitivity.
A GAPPED extension is triggered only when the HSP generated has a normalized score higher than Sg chosen so that no more than about one extension is invoked per 50 database sequences. The gapped alignment is reported only if it has an E-value low enough to be of interest, where

- $$E = N/2^{NormS}$$
- $$NormS = \frac{\lambda * Score - ln(K)}{ln(2)}$$
- $Score$ is the nominal HSP score
- λ and k are are two calculable parameters from the background probabilities Pi that amino acids occur randomly at all positions and the substitution matrix S_{ij}
- To distinguish below whether a given set of parameters λ and K refer to gapped or ungapped alignments, the subscripts g and u are used respectively.
- When gaps are not allowed, a further important theorem states that within HSPs the aligned pair of letters (i,j) tends to occur with the **target frequency**:

$$qij = P_i P_j e^{\lambda_u S_{ij}}$$

7.3 PSI-BLAST

7.3.1 Introduction

Database searches using position-specific score matrices, also called profiles or motifs, often are much better able to detect weak relationships than are database searches that use a simple sequence as query. To make motif searches more available, PSI-BLAST was created. It constructs a position-specific score matrix automatically from the output of a BLAST run, and modified BLAST to operate using such a matrix in the place of a simple query. It is more sensitive than the corresponding BLAST program and takes little more than the same time to run.

7.3.2 Score matrix architecture

For proteins, a query of length L and a substitution matrix of dimension 20 × 20 are replaced by a position-specific matrix of dimension L × 20. Position-specific gap costs may be defined as well. As with pairwise sequence comparison, one may choose among finding the best global alignment of the matrix and the simple sequence, finding the best alignment of the complete matrix with a segment of the sequence, and finding the best local alignment of the matrix and sequence.

7.3.3 Multiple alignment construction

- all database sequence segments that have been aligned to the query with E-value below a threshold (by default set to 0.01) are collected;
- Identical rows to the query segment are purged, and only one copy is retained of the rows that are 98% identical to one another.
- Pairwise alignment column that involve gap characters inserted into the query are simply ignored.

- The raw multiple alignment M is pruned to a simpler 'reduced' one for each column of M, so the reduced multiple alignment M_c will in general vary from one column C to the next. To construct M_c, we first specify the set R of sequences it includes to be exactly those that contribute a residue to column C. We then define the columns of M_c to be just those columns of M in which all the sequences of R are represented.

7.3.4 Multiple alignment construction

PSI-BLAST implements a modified version of the sequence weighting method of Henikoff.

- Gap characters are treated as a 21st distinct character;
- any column consisting of identical residues is ignored in calculating weights.
- Nc, which is the alignment variability in the Henikoff method, is now the mean number of different residue types, including gap characters, observed in the various columns of MC.

7.3.5 Target frequency estimation

The method uses the prior knowledge of amino acid relationships embodied in the substitution matrix S_{ij} to generate residue pseudocount frequencies g_i, which are averaged with the observed frequencies fi to estimate the Q_i. Specifically, For a given column C, pseudocount frequencies are constructed using the formula:

$$\sum_j \frac{f_g}{P_j} q_{ij}$$

where the q_{ij} are the target frequencies implicit in the substitution matrix, and given by target frequency defined in section 7.2.2.
Q_i is then estimated by:

$$Q_i = \frac{\alpha f_i + \beta g_i}{\alpha + \beta}$$

where α and β are the relative weights given to observed and pseudocount residue frequencies.

Chapter 8

CLUSTAL-W

8.1 Introduction

CLUSTAL-W program implements a number of improvements to progressive multiple alignment methods, which are aimed at improving sensitivity without sacrificing speed and efficiency. Traditionally, one first aligns the most closely related sequences, gradually adding the more distant ones in order to obtain a faster alignment. This strategy works particularly well while working with sequences of different degrees of divergence. The main issues with the progressive approach are the local minimum problem and choice of alignment parameters.

8.1.1 Local minimum problem

The algorithm greedily adds sequences together, following the initial tree. There is no guarantee that the global optimal solution will be found and any mistakes made early in the alignment process cannot be corrected later. Even if the topology of the guide tree is correct,each alignment step in the multiple alignment process may have some percentage of the residues misaligned. The only way to correct this is to use an iterative or stochastic sampling procedure, which is not directly addressed in this paper.

8.1.2 The choice of alignment parameters

Traditionally, one chooses one weight matrix and two gap penalties (one for opening a new gap and one for extending an existing gap), which works quite well when sequences are closely related (as residue weight matrices give most weight to the identity). With very divergent sequences, however, the scores given to non-identical residues will become critically important; there will be more mismatches than identities.Different weight matrices will be optimal at different evolutionary distances or for different classes of proteins. Furthermore, the exact values of gap penalties are crucial for obtaining a correct solution.

In order to tackle the alignment parameter choice problem, the authors of the paper dynamically vary the gap penalties in a position- and residue-specific manner.

8.2 Alignment method

The basic multiple alignment algorithm consists of three main stages:

1. all pairs of sequences are aligned separately in order to calculate a distance matrix giving the divergence of each pair of sequences
2. a guide tree is calculated from the distance matrix;
3. the sequences are progressively aligned according to the branching order in the guide tree.

8.2.1 The distance matrix

In the original CLUSTAL programs,the pairwise distances were calculated using a fast approximate method. The scores are calculated as the number of k-tuple matches (runs of identical residues) in the best alignment between two sequences minus a fixed penalty for every gap. A more accurate score from full dynamic programming using two gap penalties (opening and extending gaps) and a full amino acid weight matrix is proposed. The scores are computed as the number of identities in the best alignment divided by the number of residues compared. Both of these scores are initially calculated as per cent identity scores and are converted to distances by dividing by 100 and subtracting from 1.0 to give number of differences per site.

8.2.2 The guide tree

The trees used to guide the final multiple alignment process are calculated from the distance matrix of step 1 using the Neighbour-Joining method. This produces unrooted trees with branch lengths proportional to estimated divergence along each branch. The root is placed by a "mid-point" method at a position where the means of the branch lengths on either side of the root are equal. These trees are also used to derive a weight for each sequence. The weights are dependent upon the distance from the root of the tree, but sequences which have a common branch with other sequences share the weight derived from the shared branch.

8.2.3 Progressive alignment

Use a series of pairwise alignments to align larger and large groups of sequences, following the guide tree from the tips of the rooted tree towards the root. Each step consists of aligning two existing alignments or sequences; gaps that are present in older alignments remain fixed. In order to calculate the score between a position from one sequence or alignmentan done from another,the average of all the pairwise weight matrix scores from the amino acids in the two sets of sequences is used,i.e.if you align 2 alignments with 2 and 4 sequences respectively,the score at each position is the average of $8(2 * 4)$comparisons. If either set of sequences contains one or more gaps in one of the positions being considered, each gap versus a residue is scored as zero. When sequences are weighted, each weight matrix value is multiplied by the weights from the two sequences.

8.3 Improvements to progressive alignment

All of the remaining modifications apply only to the final progressive alignment stage. Sequence weighting is relatively straightforward and is already widely used in profile searches, while the treatment of gap penalties is more complicated. Initial gap penalties are calculated depending on

the weight matrix, the similarity of the sequences and the length of the sequences. Then,an attempt is made to derive sensible local gap opening penalties at every position in each prealigned group of sequences that will vary as new sequences are added. The use of different weight matrices as the alignment progresses is novel and largely bypasses the problem of initial choice of weight matrix. The final modification allows us to delay the addition of every divergent sequences until the end of the alignment process,when all of the more closely related sequences have already been aligned.

8.3.1 Sequence weighting

Sequence weights are calculated directly from the guide tree.The weights are normalised such that the biggest one is set to 1.0 and the rest are all less than 1.0.Groups of closely related sequences receive lowered weights because they contain much duplicated information.Highly divergent sequences without any close relatives receive high weights.These weights are used as simple multiplication factors for scoring positions from different sequences or prealigned groups of sequences.

8.3.2 Gap penalties

Gap opening penalty (GOP) and gap extension penalty (GEP) can be set by the user from a menu. The software then automatically attempts to choose appropriate gap penalties depending on following factors:

- **Dependence on the weight matrix**: the average score for two mismatched residues is used as a scaling factor for the GOP.
- **Dependence on the similarity of the sequences**: the per cent identity of the two sequences to be aligned is used to increase the GOP for closely related sequences and decrease it for more divergent sequences on a linear scale.
- **Dependence on the lengths of the sequences**: the scores for both true and false sequence alignments grow with the length of the sequences. The log of the length of the shorter sequence is used to increase the GOP with sequence length.

  $GOP \rightarrow GOP + log[min(N, M)]*$(average residue mismatch score)$*$(per cent identity scaling factor)

- **Dependence on the difference in the lengths of the sequences**: the GEP is modified depending on the difference between the lengths of the two sequences to be aligned. If one sequence is much shorter than the other,the GEP is increased to inhibit too many long gaps in the shorter sequence.

  $$GEP \rightarrow GEP * [1.0 + |log(N/M)|]$$

- **Position-specific gap penalties**: before any pair of sequences or prealigned groups of sequences are aligned, a table of gap opening penalties for every position in the two(sets of)sequences is generated. Initial gap opening penalty is manipulated in a position-specific manner, while the local gap penalty rules are applied in a hierarchical manner.
- **Lowered gap penalties at existing gaps**: if there are already gaps at a position,then the GOP is reduced in proportion to the number of sequences with a gap at this position and the GEP is lowered.

  $$GOP \rightarrow GOP * 0.3 * \text{(no. of sequences without gap/no. of sequences)}$$

- **Increased gap penalties near existing gaps**: if a position does not have any gaps but is within 8 residues of an existiing gap, the GOP is increased by:

$$GOP \rightarrow GOP * 2 + [(8 - \text{distance from ga}) * 2]/8$$

- **Reduces gap penalties in hydrophilic stretches**: any run of 5 hydrophilic residues is considered to be a hydrophilic stretch. The residues that are to be considered hydrophilic may be set by the user but are conservatively set to D, E, G, K, N, Q, P, R or S by default. If, at any position, there are no gaps and any of the sequences has such a stretch,the GOP is reduced by one third.

- **Residue-specific penalties**: if there is no hydrophilic stretch and the position does not contain any gaps, then the GOP is multiplied by one of the 20 numbers according to a specific table.

8.3.3 Weight matrices

The weight matrices offered to the user are the Dayhoff PAM series and the BLOSUM series. In each case, there is a choice of matrix ranging from strict ones (useful for closely related sequences) to very "soft" ones. The distances are measured directly from the guide tree.

8.3.4 Divergent sequences

The most divergent sequences are usually the most difficult to align correctly. It is sometimes better to delay the incorporation of these sequences until all of the more easily aligned sequences are merged first, improving gap placements and matching weakly conserved positions against the rest of sequences. A choice is offered to set a cut-off, the default is 40% identity or less.

8.4 Discussion

As more divergent sequences are included,it becomes increasingly difficult to find good alignments and to evaluate them.What we find with CLUSTAL-W is that the basic block-like structure of the alignment(corresponding to the major secondary structure elements)is usually recovered,with some of the most divergent sequences misaligned in small regions. This is a very useful starting point for manual refinement, as it helps define the major blocks of similarity.

There are several areas where further improvements in sensitivity and accuracy can be made.

1. the residue weight matrices and gap settings can be made more accurate as more data accumulate and matrices for specific sequences types can be derived.

2. stochastic or iterative optimisation methods can be used to refine initial alignments

3. the average number of examples of each protein domain or family is growing steadily. New information should be used to make alignments more accurate

Chapter 9

T-COFFEE - ELISA

T-COFFEE method is broadly based on the popular progressive approach to multiple alignment but avoids the most serious pitfalls caused by the greedy nature of this algorithm. By pre-processing a data set of all pair-wise alignments between the sequences, a library is built to guide the progressive alignment. Intermediate alignments are then based not only on the sequences to be aligned next, but also on how all of the sequences align with each other. This alignment information can be derived from heterogeneous sources such as a mixture of alignment programs and/or structure superposition.

9.1 Introduction

The most commonly used heuristic methods are based on the progressive-alignment strategy, with ClustalW being the most widely used implementation. Although successful in a wide variety of cases, this method suffers from its greediness, as errors made in the first alignments cannot be rectified later as the rest of the sequences are added in. T-Coffee is an attempt to minimize that effect, and although being still a greedy progressive method, it allows for much better use of information in the early stages.

The main alternative to progressive alignment is the simultaneous alignment of all the sequences. Two such packages exist (MSA and DCA), based on the Carrilo and Lipman algorithm, but they remain an extremely CPU and memory-intensive approach. Iterative strategies do not provide any guarantees about finding optimal solutions but are reasonably robust and much less sensitive to the number of sequences than their deterministic counterparts. Alternatively, one might wish to consider local similarity, as occurs when two proteins share only a domain or motif. Lalign, a variant of SW algorithm from the FASTA package, produces sets of non-overlapping local alignments from the comparison of two sequences. For multiple sequences, the Gibbs sampler and Dialign2 are the main automatic methods, which perform well when there is a clear block of ungapped alignment shared by all of the sequences. They perform poorly, however, on general sets of test cases when compared with global methods. T-Coffee is aimed at providing a simple, flexible and, most importantly, accurate solution to the problem of how to combine global and local multiple alignments.

9.2 T-Coffee Algorithm

T-Coffee (Tree-based Consistency Objective Function for alignment Evaluation) has two main features. First, it provides a simple and flexible means of generating multiple alignments, using heterogeneous data sources. The data from these sources are provided to T-Coffee via a library of pair-wise alignments. The second main feature is the optimization method, which is used to find the multiple alignment that best fits the pair-wise alignments in the input library. We use a so-called progressive strategy, which is similar to that used in ClustalW. T- Coffee is a progressive alignment with an ability to consider information from all of the sequences during each alignment step, not just those being aligned at that stage. The progressive alignment is speed and simple, and a lower tendency to make errors is observed. An overall scheme of T-Coffee algorithm is reported in figure 9.1.

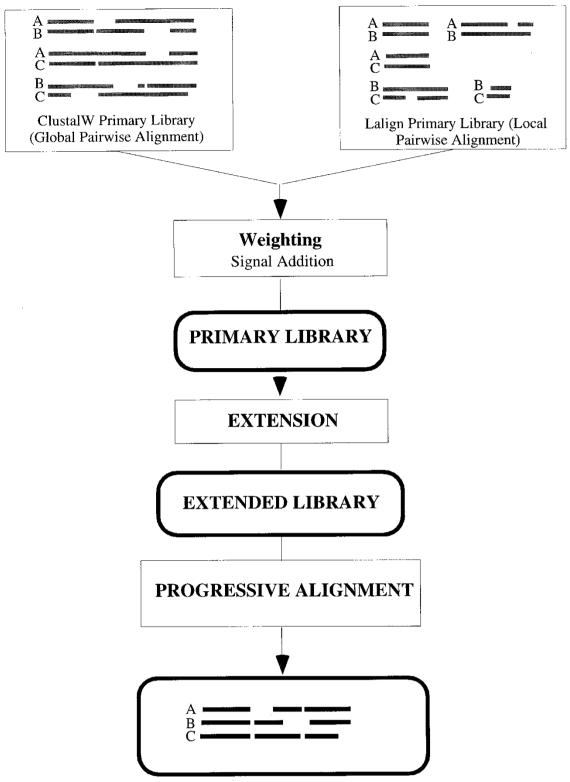


Figure 9.1

9.2.1 Generating a primary library of alignments

The primary library contains a set of pair-wise alignments between all of the sequences to be aligned. In the library, we include information on each of the $N(N-1)/2$ sequence pairs, where N is the number of sequences.

- global alignments are computed with ClustalW

- local alignments are the ten top- scoring non-intersecting local alignments, between each pair of sequences, gathered using the Lalign

In the library, each alignment is represented as a list of pair-wise residue matches (e.g. residue x of sequence A is aligned with residue y of sequence B). In effect, each of these pairs is a constraint. All of these constraints are not equally important, some may come from parts of alignments that are more likely to be correct. We take this into account when computing the multiple alignment and give priority to the most reliable residue pairs by using a weighting scheme.

9.2.2 Derivation of the primary library weights

T-Coffee assigns a weight to each pair of aligned residues in the library. An ideal primary weight will reflect the correctness of a constraint. Sequence identity is used, as it is known to be a reasonable indicator of accuracy when aligning sequences with more than 30% identity. This weighting scheme proved to be highly effective for a previous consistency-based objective function and has the advantage of great simplicity. Each constraint receives a weight equal to percent identity within the pair-wise alignment it comes from.

9.2.3 Combination of the libraries

ClustalW and Laling primary libraries are pooled in a simple process of addition. If any pair is duplicated between the two libraries, it is merged into a single entry that has a weight equal to the sum of the two weights. Otherwise, a new entry is created for the pair being considered. Pairs of residues that did not occur are not represented (by default they will be considered to have a weight of zero). This primary library can be used directly to compute a multiple sequence alignment. For each pair of aligned residues in the library, we can assign a weight that reflects the degree to which those residues align consistently with residues from all the other sequences.

9.2.4 Extending the library

Fitting a set of weighted constraints into a multiple alignment is a well-known problem, formulated as an instance of the "maximum weight trace", an NP-complete problem. The genetic algorithm is robust but may require prohibitive computation time, while the graph-theory-based algorithm has a complexity only partially characterized and may fail in some cases for reasons that are difficult to predict. The overall idea of the heuristic library extension algorithm is to combine information in such a manner that the final weight, for any pair of residues, reflects some of the information contained in the whole library. To do so, a triplet approach is used: the weight associated with a pair of residues will be the sum of all the weights gathered through the examination of all the triplets involving that pair. The more intermediate sequences supporting the alignment of that pair, the higher its weight. Extension will be carried out on each pair of residues of A and B. Once the operation is complete, sequence pair A and B will have gathered information from all the other sequences in the set. The complete set of pairs constitutes the extended library. The worst-case complexity of this computation is $O(N^3L^2)$ with L being the average sequence length. However, this will only occur when all the included pair-wise alignments are totally inconsistent. In practice the complexity is close to $(O)N^3L$.

9.2.5 Progressive alignment strategy

This alignment uses the weights in the extended library above to align the residues in the two sequences. This pair of sequences is then fixed and any gaps that have been introduced cannot be shifted later. Then the next closest two sequences are aligned or a sequence is added to the existing alignment of the first two sequences, depending which is suggested by the guide tree. The next two closest sequences or pre-aligned group of sequences are always joined.,until all the sequences have been aligned. To align two groups of pre-aligned sequences the scores from the extended library are used, as before, but the average library scores in each column of existing alignment are taken. The procedure does not require any additional parameters such as gap penalties, as the substitution values (the library weights) were computed on alignments where such penalties had already been applied. Furthermore, high scoring segments that show consistency within the data set see their score enhanced by the extension to such a point that they become insensitive to gap penalties. In practice, this means that during the progressive phase, we use a dynamic-programming algorithm with gap-opening penalties and gap- extension penalties set to zero for aligning two sequences or two groups of pre-aligned sequences.

9.3 Discussion

T-Coffee is a new progressive method for sequence alignment. It can combine signals from heterogeneous sources (e.g. sequence-alignment programs, structure alignments, threading, manual alignment, motifs and specific constraints) into a unique consensus multiple sequence alignment. The combination of local and global alignments leads to a significant increase in alignment accuracy. The main difference from traditional progressive alignment methods is that, instead of using a substitution matrix for aligning the sequences, a position-specific scoring scheme is used (the extended library). Errors are less likely to occur during early stages of the progressive alignment. As a consequence, even though the paradigm "once a gap always a gap" remains true, misplacing gaps becomes much less likely. The second important feature of T-Coffee is the combination of local and global information. The end-user benefits from the simplicity of the method and does not need to provide any extra parameter values.

A key ingredient of the method is the primary weighting scheme. The main reason why T-Coffee can tolerate small segments with high similarity is more likely to occur by chance is because short high-scoring segments are rarely consistent enough to have a strong effect on the position-specific scoring scheme after extension. Moreover, final alignments are processed using dynamic programming,making it less likely for misplaced high-scoring segments to affect the alignment. Although the protocol proposed here employs a minimal combination of local and global information, there is no theoretical limit to the number of methods that can be used. For instance, alignments from structural comparisons could be combined with sequence alignments. It is also possible to incorporate, in the library, information extracted from multiple alignments.

Chapter 10

T-COFFEE - ILARIA

T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment We describe a new method (T-Coffee) for multiple sequence alignment that provides a dramatic improvement in accuracy with a modest sacrifice in speed as compared to the most commonly used alternatives.

10.1 Introduction

The simultaneous alignment of three or more nucleotide or amino acid sequences is one of the commonest tasks in bioinformatics. Multiple alignments are an essential prerequisite to many analyses of protein families such as homology modeling or phylogenetic reconstruction, or to illustrate conserved and variable sites within a family.

10.1.1 Reasons for the development

10.1.1.1 Once a gap, always a gap

The most commonly used heuristic methods are based on the **progressive alignment strategy**, with **ClustalW** being the most widely used implementation. The idea is to take an initial, approximate, phylogenetic tree between the sequences and to gradually build up the alignment, following the order in the tree. Although successful, this method suffers from its greediness. Errors made in the first alignments cannot be rectified later as the rest of the sequences are added in. Even though the paradigm "once a gap always a gap" remains true, misplacing gaps becomes much less likely.

10.1.1.2 Global and local alignment

Some methods attempt to carry out global alignments, where one tries to align the full lengths of the sequences with each other. Alternatively, one might wish to consider local similarity, as occurs when two proteins share only a domain or motif (Smith-Waterman, Lalign). In principle, a method able to combine the best properties of global and local multiple alignments might be very powerful. This is the second motivation for T-Coffee: the design of a method that provides a simple, flexible and, most importantly, accurate solution to the problem of how to combine information of this sort.

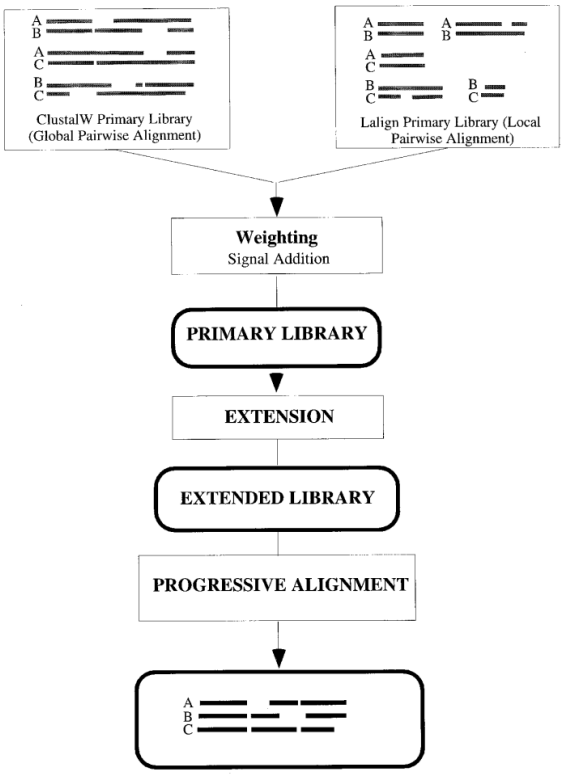


Figure 10.1: Layout of the T-Coffee strategy; the main steps required to compute a multiple sequence alignment using the T-Coffee method. Square blocks designate procedures while rounded blocks indicate data structures.

10.2 T-Coffee Algorithm

T-Coffee (Tree-based Consistency Objective Function for alignment Evaluation) has two main features.
First, it provides a simple and flexible means of generating multiple alignments, using heterogeneous data sources. The data from these sources are provided to T-Coffee via a **library** of pair-wise alignments.
The main feature of T-Coffee is the **optimization** method, which is used to find the multiple alignment that best fits the pair-wise alignments in the input library. A so-called **progressive strategy** is implemented, which is similar to that used in ClustalW. The difference from CustalW is that T-Coffe makes use of the information in the library to carry out progressive alignment in a manner that allows us to consider the alignments between all the pairs while we carry out each step of the progressive multiple alignment. This gives us progressive alignment, with all its advantages of speed and simplicity, but with a far lesser tendency to make errors like the one shown in Figure 10.2(a), i.e. misalignment of the word CAT. T-Coffee is a progressive alignment with an ability to consider information from all of the sequences during each alignment step, not just those being aligned at that stage.

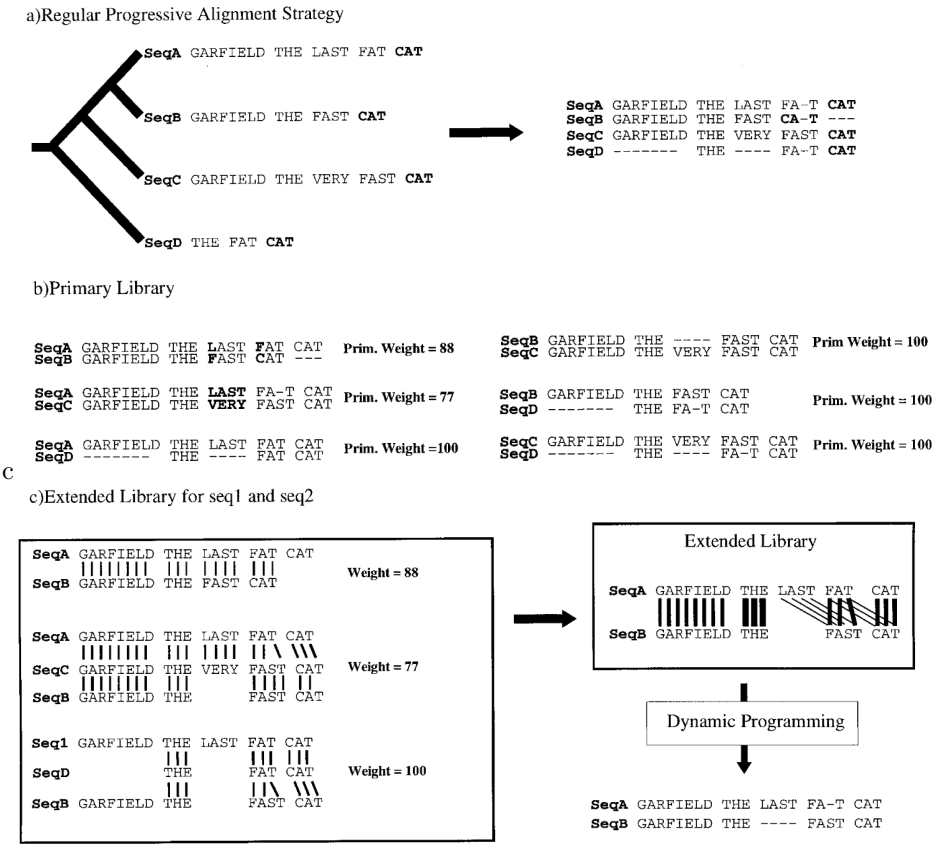


Figure 10.2: The library extension. **(a) Progressive alignment**. Four sequences have been designed. The tree indicates the order in which the sequences are aligned when using a progressive method such as ClustalW. The resulting alignment is shown, with the word CAT misaligned. **(b) Primary library**. Each pair of sequences is aligned using ClustalW. In these alignments, each pair of aligned residues is associated with a weight equal to the average identity among matched residues within the complete alignment (mismatches are indicated in bold type). **(c) Library extension for a pair of sequences**. The three possible alignments of sequence A and B are shown (A and B, A and B through C, A and B through D). These alignments are combined, as explained in the text, to produce the position-specific library. This library is resolved by dynamic programming to give the correct alignment. The thickness of the lines indicates the strength of the weight.

10.2.1 Generating a primary library of alignments

The primary library contains a set of pair-wise alignments between all of the sequences to be aligned. In the library, we include information on each of the $N(N-1)/2$ sequence pairs, where N is the number of sequences. Here, we use two alignment sources for each pair of sequences, one **local** and one **global**. The global alignments (Figures 10.1 and 10.2(b)) are constructed using ClustalW on the sequences, two at a time. The local alignments (Figure 10.1) are the ten top scoring non-intersecting local alignments, between each pair of sequences, gathered using the Lalign program.
In the library, each alignment is represented as a list of pair-wise residue matches (e.g. residue x of sequence A is aligned with residue y of sequence B). In effect, each of these pairs is a constraint. All

of these constraints are not equally important. Some may come from parts of alignments that are more likely to be correct. This is taken into account when computing the multiple alignment and give priority to the most reliable residue pairs. This is achieved by using a weighting scheme, which is described in subsection 10.2.2.

10.2.2 Derivation of the primary library weights

T-Coffee assigns a weight to each pair of aligned residues in the library (Figure 2(b)). The **sequence identity** weighting scheme is used, which has been prove to be effective and of great simplicity. Libraries are lists of weighted pair-wise constraints. Each constraint receives a weight equal to percent identity within the pair-wise alignment it comes from (Figure 10.2(b)). For each set of sequences, two primary libraries are computed along with their weights, one using ClustalW (global alignments; Figure 10.2(b)) and the second using Lalign (local).

10.2.3 Combination of the libraries

The aim is the efficient combination of local and global alignment information. This is achieved by pooling the global and local primary libraries in a simple process of **addition**. If any pair is duplicated between the two libraries, it is merged into a single entry that has a weight equal to the sum of the two weights. Otherwise, a new entry is created for the pair being considered (process called unofficially **"stacking" of the signal**). Pairs of residues that did not occur are not represented (weight of zero).
This primary library can be used directly to compute a multiple sequence alignment. However, we enormously increase the value of the information in the library by examining the consistency of each pair of residues with residue pairs from all of the other alignments. For each pair of aligned residues in the library, we can assign a weight that reflects the degree to which those residues align consistently with residues from all the other sequences. This process is called library extension (subsection 10.2.4).

10.2.4 Extending the library

Fitting a set of weighted constraints into a multiple alignment is a well known NP-complete problem. We circumvent the problem by using a heuristic algorithm that we call **library extension** (Figure 10.2(c)). The idea is to combine information in such a manner that the final weight, for any pair of residues, reflects some of the information contained in the whole library. To do so, a **triplet approach** is used, as summarized in Figure 10.2(c).
It is based on taking each aligned residue pair from the library and checking the alignment of the two residues with residues from the remaining sequences.

10.2.4.1 A quick example

For instance, let us consider the four sequences A, B, C and D of Figure 2. Let us call A(G) the G of GARFIELD in sequence A, B(G) the equivalent G in sequence B and W(A(G), B(G)) the weight associated with this pair of symbols in the primary library. In the direct alignment of A and B, A(G) and B(G) are matched (Figure 10.2(b) and (c)). Therefore, the initial weight for that pair of residues can be set to 88 (primary weight of the alignment of sequence A and B, which is the percent of identity of this pair). If we now look at the alignment of sequence A and sequence B through sequence C (Figure 10.2(c)), we can see that the A(G) and C(G) are aligned, as well as C(G) and A(G). We conclude that there is an alignment of A(G) with B(G) through sequence

C. We associate that alignment with a weight equal to the minimum of $W_1 = W(A(G), C(G))$ and $W_2 = W(C(G), B(G))$. Since $W_1 = 77$ and $W_2 = 100$, the resulting weight is set to 77. In the extended library, this new value is added to the previous one to give a total weight of 165 for the pair A(G), B(G). The complete extension will require an examination of all the remaining triplets. Not all of them bring information. For instance, the alignment of A and B through sequence D does not contain any information relative to A(G) or B(G), and, therefore, it has no influence on the weight associated with A(G) and B(G). In summary, the weight associated with a pair of residues will be the sum of all the weights gathered through the examination of all the triplets involving that pair.

10.2.4.2 Alignment

Weights will be zero for any residue pairs that never occur (this will be true of the majority of residue pairs). Otherwise, the weight will reflect a combination of the similarity of the pair of sequences or sequence segments that the residue pair comes from and the consistency of that residue pair with all other residue pairs in the primary library. These scores can then be used to align any two sequences from our data set using conventional dynamic programming. When one normally aligns a pair of sequences, one uses a set of scores derived from some general table of amino acid weights such as a Blosum matrix. In our case, we can replace that matrix with a set of scores that are specific to every possible pair of residues in our two sequences. This will allow an alignment to be carried out that will take account of the particular residues in the two sequences but will also be guided towards consistency with all of the other sequences in the data set.

10.2.5 Progressive alignment strategy

The *normal* progressive alignment strategy consists in creating a guide tree (a phylogenetic tree) using the neighbor-joining method. The closest two sequences on the tree are aligned first using normal dynamic programming. This pair of sequences is then fixed and any gaps that have been introduced cannot be shifted later. Then the next closest two sequences are aligned or a sequence is added to the existing alignment of the first two sequences, depending which is suggested by the guide tree.

As used here, the procedure does not require any additional parameters such as gap penalties. This stems, in part, from the fact that the substitution values (the library weights) were computed on alignments where such penalties had already been applied. Furthermore, high scoring segments that show consistency within the data set see their score enhanced by the extension to such a point that they become insensitive to gap penalties.

10.3 Biological validation

I chose not to report the comparisons with other tools and the complexity of the algorithm. If needed, exhaustive tabled can be found in the paper.

10.3.1 Application to serine/threonine kinases

A major application of any alignment algorithm will be the delineation of motifs or domains. 19 sequences from a sub-family of the serine/threonine kinases were provided. Each sequence in the alignment contains a nucleotide-binding site (NBS). In all these sequences, the NBS is followed by a second conserved motif toward the C terminus. T-Coffee was able to accurately align 18 of the 19

NBSs, ClustalW was only able to correctly align 16 of these NBSs. The second motif is more difficult because of the long indel in st11 yeast. Here as well, T-Coffee can properly align 18 of the motifs, while ClustalW get 15 correct. As a result of combining local and global alignment information, T-Coffee managed to align almost all of the motifs as in the BaliBase reference alignment. Moreover, T-Coffee was the only program that correctly aligned the second motif of kp68 human, which is an interferon-induced kinase.

Chapter 11

Protein profiles with HMMs

11.1 Introduction

An alignment of two or more sequences can be used to infer the evolutionary relationships between sets of protein (or nucleic acid) sequences and eventually predict their 3D structure. In order to obtain meaningful results, the accuracy of the alignment must be as high as possible. Consider a family of protein sequences that all have a common three-dimensional structure, for example, the family of globins. A **profile** of globins can be thought of as a statistical model for the family of globins, in that for any sequence of amino acids, it defines a probability for that sequence, such that globin sequences tend to have much higher probabilities than non-globin sequences. The aim of this paper is to build a class of structures related to profiles and propose them as statistical models of protein families, such that we are able to "learn" or "estimate" the stochastic model directly from raw unaligned sequences. Once the model is obtained, it is possible to use it to discriminate sequences in the family from sequences not in the family, or to obtain a multiple alignment of all the sequences in the family to the model.

The proposed method of multiple alignment is quite different from conventional methods, which are usually based on pairwise alignments using the standard dynamic programming scheme with gap penalties. Such alignments often depend strongly on the particular values of the parameters required by the method, in particular the gap penalties. Furthermore, the same penalties are used for both fairly conserved regions and highly variable regions; substitutions, insertions, or deletions in a region of high conservation should ideally be penalized more than in a variable region. The statistical model here proposed corresponds to multiple alignment with variable penalties, depending on the position and learned from the data itself.

The type of statistical model is a hidden Markov model (HMM, or simply “model” for short). The HMM we build identifies a set of positions that describe the (more-or-less) conserved first-order structure in the sequences. In biological terms, this corresponds to identifying the core elements of homologous molecules. Each of these positions may be viewed as corresponding to a column in a multiple alignment of the sequences, or to a position in space, Often not all positions are modeled, but only the ones with the most significantly nonrandom statistics. As in a profile, for each position in the model, the relative probabilities of the different amino acids in that position are given, as is the probability that the amino acid at that position is deleted. For any amino acid x,the negative logarithm of the probability of x at a given position in the model can be interpreted as a penalty for

an alignment that puts z in this position, and the negative logarithm of the probability of deletion at that position can be similarly interpreted as a penalty for an alignment that puts a " - " (gap character) in that position. Furthermore, for each pair of adjacent positions, the model includes the probability for initiating an insertion of one or more amino acids between these positions, and the probability for extending it. Other things are also included, such as the probability that an amino acid at a particular position is deleted, given that the amino acid at the previous position was deleted. This helps model the higher probability of consecutive deletions/insertions in certain regions. By taking negative logarithms as above, these methods provide a very flexible, position-dependent form of initiation and extension gap penalties.

11.2 The protein HMM

A hidden Markov Model describes a series of observations by a "hidden" stochastic process - a Markov process. In this paper we are proposing an HMM similar to the ones used in speech to model protein families - or families of other molecular sequences like DNA and RNA. When modeling proteins, the observations are the amino acids in the primary sequence of a protein. A model for a set of proteins is one that assigns high probability to the sequences in that particular set.

The HMM contains a sequence of M states that we call match states. These correspond to positions in a protein or columns in a multiple alignment. Each of these states can generate a letter x from the 20- letter amino acid alphabet according to a distribution $P(x|m_k), k = 1...M$. For each match state m_k there is a delete state d_k that does not produce any amino acid, it is a "dummy" state, shown as circles. Finally, there are insertion states between all the match states, $M + 1$ in total, shown as diamonds. They generate amino acids in exactly the same way as the match states, but they use probability distributions $P(x|i_k)$. From each state, there are three possible transitions to other states: transitions into match or delete states always move forward in the model, whereas transitions into insertion states do not. Multiple insertions between match states can occur because of the self loop on the insert state, meaning that a transition from an insert state to itself is possible. The transition probability from state s_1 to s_2 is called $\mathcal{T}(s_2|s_1)$. The overall model is reported in figure 11.1.

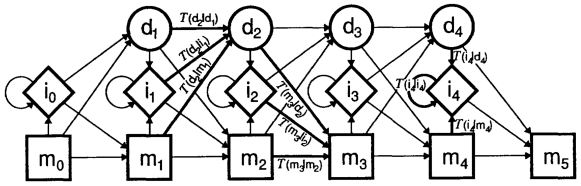


Figure 11.1

A sequence can be generated by a "random walk" through the model as follows: beginning at state m_0 (BEGIN) choose a transition randomly according to the probabilities $\mathcal{T}(m_1|m_0), \mathcal{T}(d_1|m_0), \mathcal{T}(i_0|m_0)$. If m_1 is chosen, generate the first amino acid x_1 from the probability distribution $P(xlm_1)$, and choose a transition to the next state according to probabilities $\mathcal{T}(\cdot|m_1)$where " $\cdot$ " indicates any possible next state. Continue in this manner all the way to the END state, generating a sequence of amino acids $x_1, \ldots, x_L$by following a path of states $y_0 \ldots y_{N+1}$ through the model, where $y_0 = m_0$ (the BEGIN state) and $y_N + 1 = m_{M+1}$ (the END state). Because the delete states do not produce any amino acid, N is larger than or equal to L.

If y_i is a match or insert state, we define $l(i)$ to be the index in the sequence $x_1, \ldots, x_L$ of the amino acid produced in state y_i. The probability of the event that the path $y_0 \ldots y_{N+1}$ is taken and the sequence

$$\text{Prob}(x_1, \ldots, x_L, y_0 \ldots y_{N+1}) = \mathcal{T}(m_{N+1} \mid y_N) \prod_{i=1}^{N} \mathcal{T}(y_i \mid y_{i-1}) \mathcal{P}(x_{l(i)}|y_i) \tag{11.1}$$

where we set $\mathcal{P}(x_{l(i)}|y_i) = 1$ if y_i is a delete state. The probability of any sequence $x_1, \ldots, x_L$ of amino acids is a sum over all possible paths that could produce that sequence which we write as follows:

$$\text{Prob}(x_1, \ldots, x_L) = \sum_{\text{paths}} \text{Prob}(x_1, \ldots, x_L, y_0 \ldots y_{N+1}) \tag{11.2}$$

In this way a probability distribution on the space of sequences is defined. The goal is to find a model that accurately describes a family of proteins by assigning large probabilities to sequences in that family.

This structure for the HMM captures the structural intuition of a protein as a sequence of positions, each with its own distribution over the amino acids, along with the possibility for either skipping a position or inserting extra amino acids between consecutive positions, and allowing for the possibility that once a position is skipped, it may be more likely that positions following that one are also skipped. This choice appears to have worked well for modeling the globins, but one can choose any structure for the states and transitions that is appropriate for the problem at hand.

11.3 Estimating the HMM

All the parameters in the model (i.e., probabilities) could in principle be chosen by hand from an existing alignment of protein sequences, or from information about the three-dimensional structure of proteins. The novel approach we take is to "learn" the parameters entirely automatically from a set of unaligned primary sequences, using an Expectation Maximization algorithm. The particular method we use to learn the model can be viewed as a computationally efficient approximation to Maximum A Posteriori (MAP) estimation of the parameters of the model.

In MAP estimation, the posterior probability is defined using Bayes rule,

$$\text{Prob}(\text{model}|\text{sequences}) = \frac{\text{Prob}(\text{sequences}|\text{model})\,\text{Prob}(\text{model})}{\text{Prob}(\text{sequences})} \tag{11.3}$$

The prior on the models contains prior beliefs on what a model should be like, and can be used to "penalize" models that are known to be bad or uninteresting. To find the best model one can optimize the posterior probability 11.3 with respect to the parameters of the model.

11.3.1 The distance from sequence to model

First, we need a way to calculate the probability Prob(sequences|model). Call the sequences $s^1 \ldots s^n$. Then, assuming independence

$$\text{Prob}(s^1 \ldots s^n \mid \text{ model }) = \prod_{\mu=1}^{n} \text{Prob}(s^\mu \mid \text{ model }). \tag{11.4}$$

The expression for Prob(s^μ|model) is given in can be calculated using the "forward algorithm". The previous equation can be approximated by the following for simplicity:

$$\text{Prob}(s \mid \text{ model }) \simeq \max_{\text{paths}} \text{Prob}(s, \text{path } \mid \text{ model}) \tag{11.5}$$

Instead of maximizing the probability, it is most convenient to minimize the negative logarithm of the probability, which is defined as the distance from the sequence to the model.

$$\text{dist }(s,\ \text{model }) = -\min_{\text{paths}} \log \text{Prob}(s,\ \text{path } \mid \text{ model }) = -\min_{\text{paths}} \sum_{i=1}^{N+1} \left[\log \mathcal{T}\left(y_i \mid y_{i-1}\right) + \log \mathcal{P}\left(x_{l(i)} \mid y_i\right)\right) \tag{11.6}$$

Where:

- path: $y_0 \ldots y_{N+1}$
- sequence s: $x_1 \ldots x_L$
- penalty: $\mathcal{T}\left(y_i \mid y_{i-1}\right)$

The penalties depends on the position in the model and the distance is very similar to the standard "edit distance" from one sequence to another. The most probable path can be found using the usual backtracking technique, as explained in the following section.

11.3.2 Estimation algorithm

The Baum-Welch or forward-backward algorithm is a version of the general EM (Expectation-Maximization) method often used in statistics, which can be applied faster but using the Viterbi approximation. The Viterbi-EM adaptation of the model to the sequences is an iterative process. It proceeds as follows:

1. choose the initial model
2. find the most probable path through the model for each of the sequences. The number of these paths that pass through state y is denoted $n(y)$, the number of times the transition from y to y' occurs $n(y'|y)$, and the number of times an amino acid x was generated in state y is denoted $m(x|y)$. Obviously $n(m_k) = n(m_{k+1}|m_k) + n(d_{k+1}|m_k) + n(i_k|m_k)$ and similarly for delete and insert states
3. Reestimate the transition probabilities and amino acid probabilities in the match and insertion states from

$$\mathcal{T}\left(y' \mid y\right) = \frac{n\left(y' \mid y\right) + R\left(y' \mid y\right)}{n(y) + \sum_{y''} R\left(y'' \mid y\right)} \tag{11.7}$$

$$\mathcal{P}(x \mid y) = \frac{m(x \mid y) + R(x \mid y)}{n(y) + \sum_{i=1}^{20} R\left(a_i \mid y\right)} \tag{11.8}$$

Here $R(y'|y)$ and $R(x|y)$ represent a regularizer, which comes from the model prior 11.3 We assume that $R(y"|y) = 0$ if there is no transition from state y to state $y"$. Often R is just set equal to one in order to avoid zero probabilities, but it can be used to bias the model in a certain direction.

4. repeat step 2 and 3 until the parameters change only insignificantly

The main difference between this method, which uses the Viterbi approximation, and the standard Baum-Welch algorithm is that in the latter all paths (not just the most probable) are considered, but weighted according to their probability.

11.3.3 Choosing the length of the model

A simple heuristic selects a good model length, and even helps in the problem of local minima. After learning, if more than a fraction γ_{del} of the optimal paths of the sequences choose d_k, the delete state at position k,that position is removed from the model. Similarly, if more than a fraction γ_{ins} msake insertions at position k (in state i_k), a number of new positions are inserted into the model after position k. This number is equal to the average number of insertions the sequences make at that position. After these changes in the model, it is retrained, and this cycle is repeated until no more changes are needed. We call this "model surgery". Currently we choose γ_{del} and γ_{ins} to be 1/2.

11.3.4 Over-fitting and regularization

A model with too many free parameters cannot be estimated well from a relatively small data set of training sequences. If we try to estimate such a model, we run into the problem of overfitting. One way to deal with this problem is to control the effective number of free parameters in the model by regularization, which can be viewed as a way of implementing MAP estimation. In our application of regularization to HMMs, the regularizer is closely related to the model prior, and it can be used to bias the model in some specific direction. The regularizer $(R)$is added to the frequency counts in 11.7 and 11.8 as if we actually saw more than $n(y'|y)$ or $n(x|y)$ events. For instance, in our models we believe a prior i that sequences should spend most of their time in the match states on the main line of the HMM, so we incorporate this belief into the model prior by setting $R(m_i|y)$to a large value compared to the regularizer $R(y|m_i)$,which is added to transitions diverging from the main line. In so doing, these parameters become less adjustable, and as a result we are less prone to overfit the data. We also tend to get better models because we are encouraging the learning algorithm to first explore the more a prior i likely part of model space when trying to fit the data. In the extreme case, the regularizers $R(\cdot|\cdot)$can be such large numbers that the parameters simply become "hardwired". We have found that the method works best when very large regularizers $R(x|y)$, proportional to the relative overall frequencies of amino acids in globin sequences, are used when y is an insert state. This has the effect of fixing the distributions in the insert states, rather than trying to estimate them from the training sequences.

11.4 Conclusions

The method utilizes the tremendous amount of information contained in many sequences from the same family. For the simple case of globins, with this method it is possible to obtain multiple alignments that mirror structural alignments using only the unaligned primary sequences as input. The method requires that many sequences be available from the family one wants to model; since the number of sequences in the protein databases is growing rapidly, this may be less of a problem in the future. There are two possible answers to the problem of small sets of training sequences.

The first is to add more prior knowledge into the training process by starting with a better initial

model, and by using more involved kinds of regularization. The "soft tying" regularization method combines the idea of tying states in which the number of free parameters is reduced by having groups of states all share the same distribution on the output alphabet (the 20 amino acids in this case), and the idea of soft weight, in which the regularizer (in this case for the distribution of amino acids) is also adaptively modified during learning. Even more complex, position-specific kinds of regularization are possible. Switching from the alphabet of the primary sequences to a different representation based more on the structural or chemical properties of the amino acids in the sequence may also help.

The second answer is to give up trying to model the whole protein in cases where only few sequences are known, and instead try to model only pieces of the protein (motifs or domains) that have the same or similar structure in other proteins. In this case we can find more training sequences by adding in the corresponding pieces of these other proteins. In fact, the EM method can be used to locate the appropriate pieces by itself, given the entire sequences. Several models can also be linked together with free insertions in between, to model more than one subsequence in the molecules.

Finally, when a relatively large number of sequences are available, it is sometimes possible to get better results by dividing these sequences into clusters of similar sequences, and training a different model for each cluster. The partition into clusters can also be done automatically by the EM algorithm during training, in a procedure known as mixture modeling. To implement this, we simply start with more than one model and force the models to “compete” for the sequences, and adapt to fit only the ones they “won”.

Chapter 12

Construction of phylogenetic trees

*A method based on **mutation distances** as estimated from cytochrome c sequences is of general applicability (1967)*

12.1 What are phylogenetic trees

They are graphs made up of nodes, which represent the taxonomic units and from branches that join the nodes, representing the distances between the two.
Topology is defined as the general structure of a tree. If to the branches evolutionary distance is not given, we have a cladogram (I), otherwise we have a filogram (II) and with time a dendogram (III). Trees without root do not foresee evolutionary meanings in time terms and describe simply the relationships between sequences. Trees with root they accept the hypothesis as true of the molecular clock and the nodes they are in a specific temporal order.

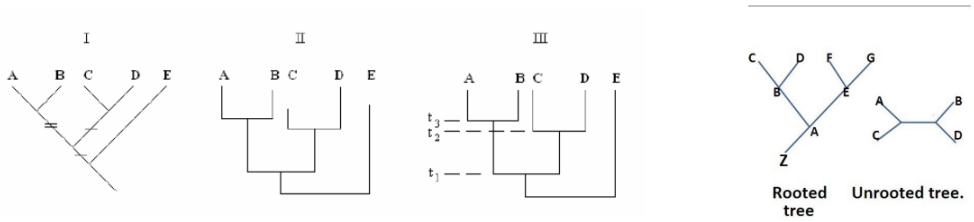


Figure 12.1: Topologies of phylogenetic trees

12.2 Introduction

Some methods to constructs phylogenetic trees were developed before the one that will be discussed here, like studies of the degree of interspecific hybridization of DNA, number of amino acid replacements between homologous proteins, etc. These methods are not satisfactory, because:

- i) the portion of the genome examined was often very resticted
- ii) Low accuracy in the mutation distance between the genes examined

- iii) no adequate mathematical treatment of large datasets

To tackle especially points ii) and iii), a new method is hereby proposed and the validity of the results is carried out with the construction of a tree considering only cytochrome *c*, for much much information is available. Even just considering one gene, the resulting tree is much similar to the phylogenetic tree constructed directly from biological data.

12.2.1 Ancenstral genes, Orthologs and Paralogs

Among homologous genes, one has to distinguish **orthologous** from **paralogous** sequences. Orthologs are **homologous** genes in different species that diverged from a single **ancestral gene** after a speciation event and paralogs are homologous genes that originate from the intragenomic duplication of an ancestral gene.

12.2.2 Proteins or nucleic acids?

For molecular phylogeny it is preferable to use nucleotide sequences. In phylogeny both are used, but it is necessary to fix some key aspects that define the different action ranges of the two:

- Protein sequences
  - need 20x20 replacement matrices, very complex to deal with
  - are the expression of coding regions only
  - identical amino acids can be the expression of multiple codons
- Nucleotide sequences
  - they can be described with 4x4 matrices.
  - can be extracted from non-coding genomic sequences, therefore with higher variation
  - they have no degeneration or redundancy.

12.3 Determining the mutation distance

The mutation distance between two cytochromes is defined here as the minimal number of nucleotides that would need to be altered in order for one cytochrome to code for the other. This distance is determined by making pair wise comparison of homologous amino acids. For each pair a mutation value is taken from a pre-defined table, which gives the minimum number of nucleotide changes required to convert the coding from one amino acid to the other.
For each possible pairing of cytochromes, the 110 mutation values found are summed to obtain the minimal mutation distance. The basic approach tot he construction of the tree is illustrated in figure 12.2, which shows three hypothetical proteins, A, B and C, and their mutation distances. There are two fundamental problems:

- Which pair to join first? As a first approximation, one solves it by simply choosing the pair with the smallest mutation distance, which in this case is A and B, with a distance of 24. A and B are shown connected at the lower apex.
- What are the lengths of legs *a, b* and *c*? One notes that the distance from A to C, 28, is 4 less than the distance form B to C. Meaning, there must have bee at least 4 countable mutations in the descent of B from the lower apex than in the descent of A.

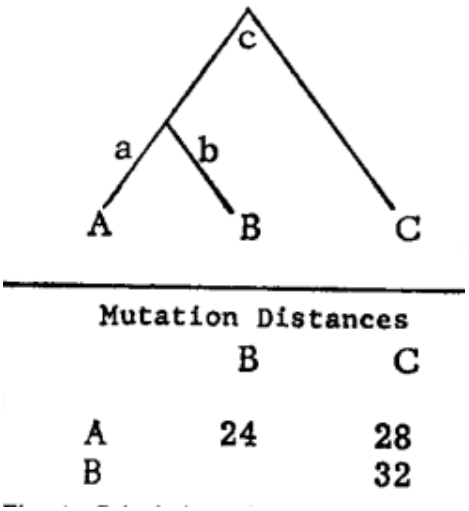


Figure 12.2: Calculation of observed mutation distances. The upper apex represents a hypothetical ancestral organism that divided into two descending lines, one of which subsequently also divided. Thus we have three present-day species, A, B, C. The number of observable mutations that have occurred since the A and B lines of descent diverged are represented respectively by a end b. The number of mutations that separate the lower apex and C is represented by c. The sums of $a+b, a+c, b+c$ then are the mutation distances of the three species as currently observed.

When information from more than three protein is utilized, the basic procedure is the same, except that initially each protein is assigned to its own subset. One then simply joins two subsets to create a single, more comprehensive subset, and the process is repeated until all proteins are members of a single subset. A phylogenetic tree is but a grphical representaion of the order in which the subsets were joined.
In the case of this study, they started with 20 subsets, each subset consisting of a single cytochrome c amino acid sequence. One arbitrarily accepts, from among all the possible pairings examined, that assigned of protein subsets to sets A, B and C which provides the lowest average mutation distance from A to B. The leg lengths are then recorded. Henceforth the proteins of A and B so joined are treated as a single subset, and the entire procedure described in the preceding paragraph is repeated. Thus the number of proteins(N) is reduced by 1 at each cycle. The tree will be produced after $N-1$ joinings.

12.3.1 Genetic distances

From slides, lec 13
For the phylogenetic distinction of two sequences, it is necessary to know how much they diverge. We therefore need an objective parameter and a calculable, defined genetic distance. For conserved nucleic acids, the number of percentage substitutions observed after multiple alignments

$$d = \frac{n.replacements}{length} \tag{12.1}$$

For less conserved nucleic acids the d is corrected with the Jukes-Cantor formula:

$$d' = -\frac{3}{4} log(1 - \frac{4}{3}d) \tag{12.2}$$

For proteins, aligned with a replacement matrix, it is often used the approximated formula:

$$d = \frac{(Sobs - Srand)}{(Smax - Srand} \tag{12.3}$$

With Sobs the score of the alignment, Smax average of the scores of the alignments of all proteins with themselves, Srand score expected for sequences of the same length and composition.

12.4 Testing alternative trees

Because of the arbitrary nature of the rule by which proteins were assigned to sets A and B, the initial tree will not necessarily represent the best use of the information. The solution provided by the authors is simply to construct another tree by assigning an alternative pair of protein subsets to sets A and B whenever the mutation distance between the two subsets is not greater by some arbitrary amount than that between the members of the initial pair used in constructing the initial phylogenetic tree.

12.4.1 Standard deviation

If the absolute difference between two such mutation distances $|(i,j) - (j,i)|$ is multiplied by 100 and divided by (i,j), the result is the of change from the input data. If such values are squared and the squares are summed over all values of $i < j$, the resultant sum ($\mathcal{E}$) may be used to obtain the percent "standard deviation" of the reconstructed values from the input mutation distances. The number of mutation distances summed is $N(N-1)/2$. The number is reduced by 1, divided into the sum $\mathcal{E}$ and the square root taken, the result is the percent "standard deviation".

12.5 The statistically optimal tree

In testing phylogenetic alternatives, one is seeking to minimize the percent standard deviation.
In addition to using a single gene product to discover evolutionary relationships among several species, one can similarly delineate evolutionary relationships among different genes (int he paper the gene phylogeny, from the amino acid sequence was reconstructed for human alpha, beta, gamma and delta hemoglobinn chains and whale myoglobin).
A cautionary note may be derived from this. A wildly incorrect result could e easily be obtained if the presence of multiple, homologous genes were not recognized and a phylogeny were reconstructed from sequences which were coded for, say, half by genes for alpha hemoglobin chains and half by genes for beta hemoglobin chains. This results from the speciation having occurred more recently than the gene duplication which permitted the separate evolution of the alpha and beta genes.

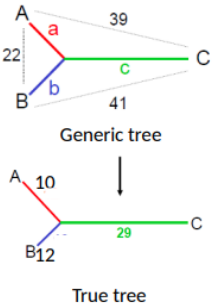
12.6 Phylogenetic relationships in trees

From slides, lec 13
The systems for building trees aims to discover how the sequences are in relation to each other: the data (the sequences themselves or distances) are used to create a system of equations that

models "when" the sequences diverge (the nodes) and how extensive it is the separation separation (branches).

It starts from a theoretical tree in which the various OTUs are inserted, without giving importance to the various branches. The goal is to understand how, how much and where they fork using various criteria, not necessarily molecular.

12.6.1 Fitch-Margoliash algorithm (FM)



- The Fitch-Margoliash algorithm is used to calculate the length of the branch given a tree with a known topology or a star topology when it is unknown.
- It works well when the distances are relatively comparable and assumes that the length of the branches is additive.
- For 3 sequences proceed as follows:
  (1) a + b = 22
  (2) a + c = 39
  (3) b + c = 41
- Example:
- (4) = (2) - (3) = a - b = 39 - 41 = - 2
- (4) + (1) = a - b+ (a+b) = 2a = 20 => a = 10
- from (1): to + 10 = 22 => b = 12
- from (2): 10 + c = 39 => c = 29
- System: 3 linear equations 3 unknowns => solvable

Figure 12.3

Chapter 13

Toward defining the course of evolution: minimum change for a specific tree topology

"It has been a goal of those attempting to deduce phylogenetic relationshps on biological characteristics to find the ancestral relationship(s) that would permit one to account for the descent of those characteristics in a manner requiring a minimum number of evolutionary steps or changes. The result could be called the most parsimonious evolutionary tree and might be expected to have a high degree of correspondence to the true phylogeny (Camin and Sokal, 1965)"

13.1 Maximum parsimony principle

Two definition of the **maximum parsimony principle** are:

- "find the ancestral relationship(s) that would permit one to account for the descent of those characteristics in a manner requiring a minimum number of evolutionary steps or changes" (Fitch)
- "We believe that selection was more likely to discover the utility of a mutation in two closely related, and possibly contemporaneous, lines of descent than that a mutation once found beneficial should then become deleterious relative to its previous ancestral form"

They arise from Occam's razor and minimum description length principle. However, we might find mutations that are neither beneficial nor detrimental.
from entry "Simplicity" on Stanford's Philosophy Enciclopedia: "Most philosophers believe that, other things being equal, simpler theories are better. But what exactly does theoretical simplicity amount to? Syntactic simplicity, or elegance, measures the number and conciseness of the theory's basic principles. Ontological simplicity, or parsimony, measures the number of kinds of entities postulated by the theory."

13.1.1 Maximum parsimony method

The most parsimonious (phylogenetic) tree is the one with lowest score, where the score is the number of mutations. An example is shown in figure 13.1.

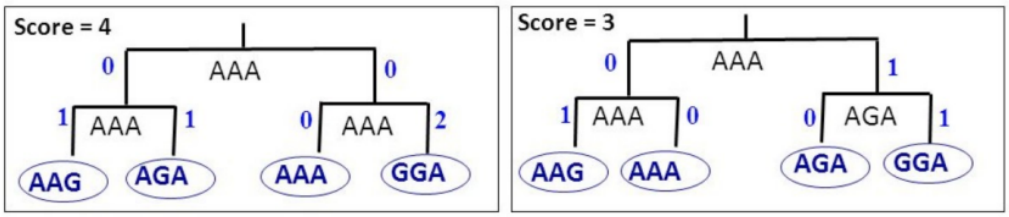


Figure 13.1: Most parsimonious tree is the one on the right, because it has a lower score.

13.2 Introduction

The justification of finding the a most parsimonious tree lies in the most efficient use of the information available and does not presuppose that evolution follows a most parsimonious course. The numerical taxonomic procedure seen in the chapter before (Fitch and Margoliash, 1967) provide dendograms that could be among the more parsimonious solutions, but there's no way to be sure.
A sub-problem to finding the most parsimonious tree(s) is finding all possible, most parsimonious assignments of the information to any given particular tree. The solution provides the most reasonable hypotheses on the ancestral states and therefore is the best estimate of the course of evolution.
The biological characteristics to be used are the nucleotides of ortologous genes (but the method is applicable to any data for which the underlying assumptions are acceptable).
Given a set of descendent nucleotide sequences and a topology presumed to describe their ancestral relationship one cans et forth:

- The minimum number of mutations to account for the descent of those sequences from a common ancestor;
- all possible ancestral sequences at each node (branching point);
- combinations of nodal ancestral sequences;
- a way of assigning relative weights among the various alternative nucleotide replacements.

13.3 Method

Task: given a tree whose topology has been already determined find for each taxonomic unit its ancestral value at each node, following the maximum parsimony principles.

13.3.1 Assumptions

The assumptions are:

- A set of ortologous descendant sequences is available;
- the ancestral relationships among them are known;
- any nucleotide may replace any other other nucleotide at any position. The word *may* indicates that all logical options are to be considered, not denying however the possibility that selective forces may have an impact on some options.

Each position can have as a value the nucleotides A, C, G, or U or some combination of the,. Ambiguity can arise because of uncertainty of the codon (redundancy of the genetic code), or uncertainty about ancestral character states. An unknown position is represented by a set of letters. The algorithm must be repeated for every nucleotide position in the gene. The reconstruction of the ancestral gene the follows two phases, the preliminary phase, in which nucleotides are placed on all ancestral nodes, and a final phase in which corrections to the preliminary assignments are made. Finally, since not every possible link should be treated as equally representative of those events that did occur in the descent, "probabilities" are assigned to each link.

13.3.2 Reconstruction of possible ancestral nucleotides-preliminary phase

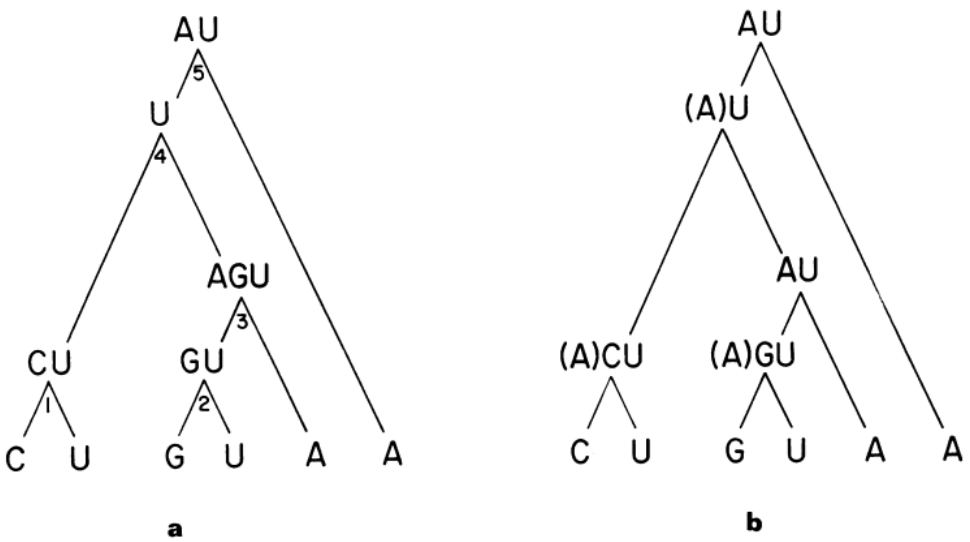


Figure 13.2: Reconstruction of ancestral nucleotide forms

The preliminary phase proceeds from the descendent character sets by formulating an ancestral character set for an immediate ancestor and working backward, one successive node at a time, until the nodal character set for the most distant ancestor has been formed. The formulation of each nodal set follows the simple rule: The preliminary nodal set shall be comprised of all characters (nucleotides) common to both immediately descendent sets; if none are common to both, then the preliminary nodal set will be comprised of all characters found in either. In mathematical terms, the nodal set is the intersection of its immediately descendant sets if the intersection exists (i.e., is not empty) otherwise it is the union of those sets. An example is shown in figure 13.2a (on the left). Following the example in figure 13.2a: because there are so many different nucleotides in only 6 taxonomic units, many of the descendants have no nucleotides in common, with the result that the first (lowest) three preliminary nodal sets are formed by unions. At the penultimate node [4], the intersection $CU \cap AGU = U$ and so U is the preliminary nodal set. The ultimate node is once again a union. Note that with real amino acid sequences such as the cytochromes *c*, the vast majority of the nodal sets are simpler in that they are formed by intersections of identical elements.
It should be noted that for every occasion that a union is required to form the preliminary nodal

set, a mutation (nucleotide replacement) must have been fixed in this nucleotide position at some point during the evolution of this position.

13.3.3 Reconstruction of possible ancestral nucleotides-final phase

The necessity of a second phase arises from the deficiencies of the preliminary phase.

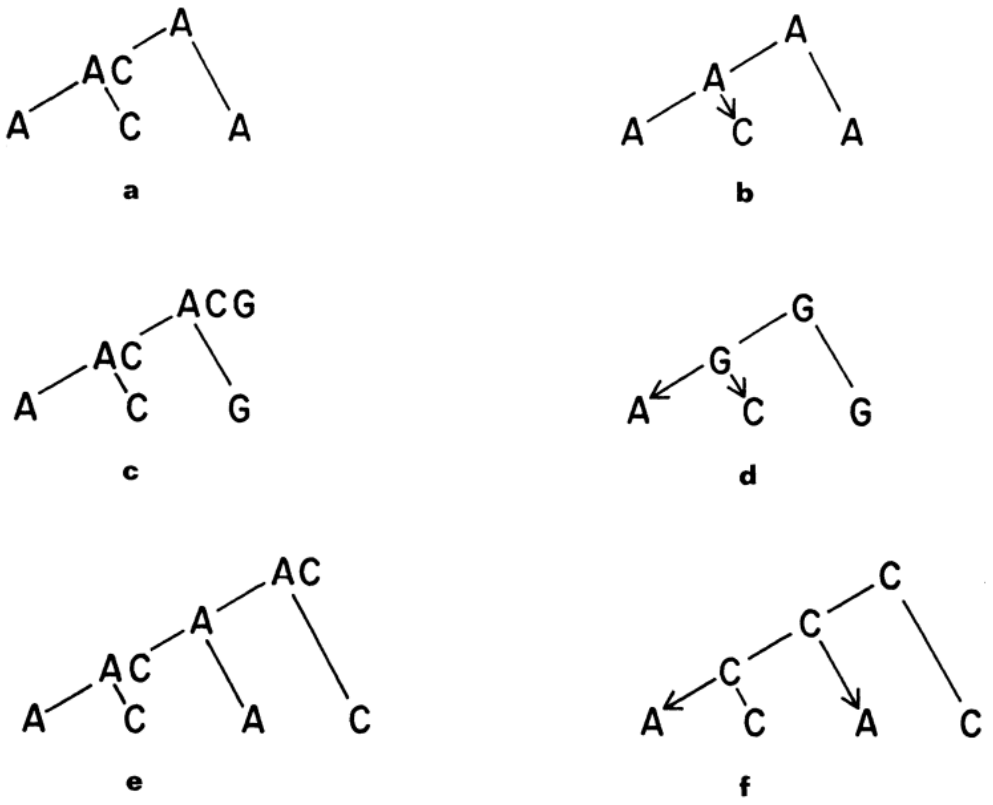


Figure 13.3: Deficiencies of preliminary phase reconstruction.

IN figure 13.3a the ambiguous AC represents an initial inability to decide whether the ancestral form was an A replaced by a C or vice versa. The only certainty is that one replacement is required. The only formulation that will permit the descendent positions to be accounted for in a single replacement requires that replacement from A to C. The elimination of C from the first node is determined by the *rule of diminished ambiguity*. The rule is described in steps I and II of the algorithm.
Figure 13.3d shows an equally adequate solution to that of 13.3c. The fact that G is an valid alternative for the first node in encompassed by the *rule of expanded ambiguity*, which is described in steps III and IV.
In Figure 2e (lower left) is shown a third preliminary phase reconstruction that accounts for four descendants using two replacements. Figure 13.3f is a valid alternative. Indeed, the C at the lowest node is a valid alternative to A if and only if a C is allowed at the penultimate node above. Two

nodes contain a nucleotide not present in the intervening node because of the intersection process. Hence this is called the *rule of encompassing ambiguity*, described in step V.

13.3.3.1 The algorithm

- **I**: If the preliminary nodal set contains all of the nucleotides present in the final nodal set of its immediate ancestor, go to II, otherwise go to III.
- **II**: Eliminate all nucleotides from the preliminary nodal set that are not present in the final nodal set of its immediate ancestor and go to VI.
- **III**: If the preliminary nodal sey was formed by a union of its descendant sets go to IV, otherwise V.
- **IV**: Add to the preliminary nodal set any nucleotides in the final set of its immediate ancestor that are not present in the preliminary nodal set and go to solution which is not encompassed by the VI.
- **V**: Add to the preliminary nodal set any nucleotides not already present provided that they are present in both the final set of the immediate ancestor and in at least one of the two immediately descendent preliminary sets and go to VI.
- **VI**: The preliminary nodal set being examined is now final. Descend one node as long as any preliminary nodal sets remain and return to I above.

Figure 13.1 shows the operation of the algorithm.

13.3.4 Permitted links between nucleotides in successive nodal sets

Figure 13.4 is a redrawing of the ancestral nodes of figure 13.1b with links connecting those nucleotides that comprise valid lines of descent in a most parsimonious tree. The arrowheads are on the internodal links that involve a change of nucleotide.

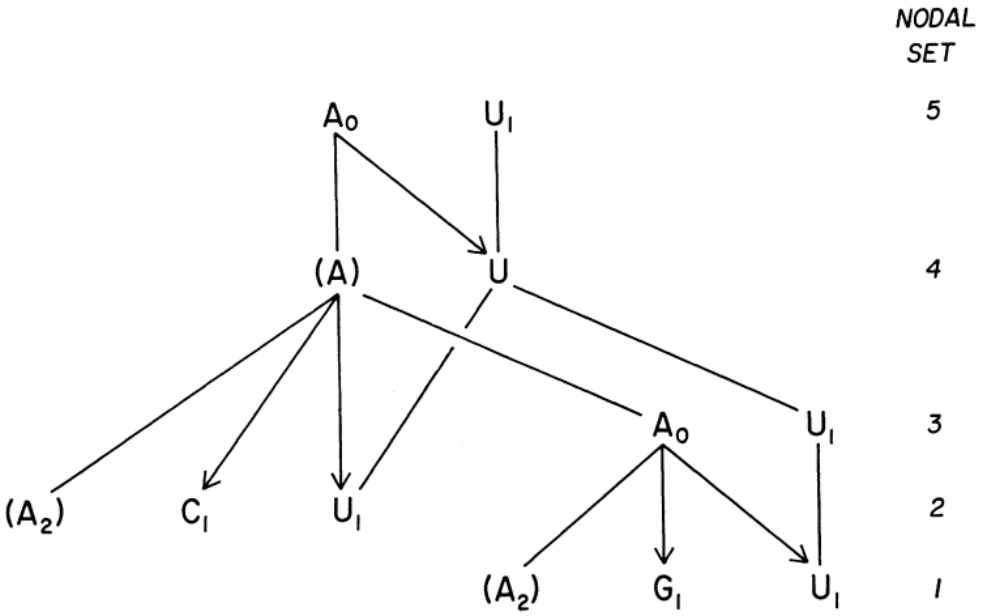


Figure 13.4: Possible temporal sequence of ancestral nucleotides.

The subscripts give the number of nucleotide replacements required in those cases where one or both immediate descendants are not other ancestral nodes. Since node 4 has only other ancestral nodes for its immediate descendants, its nucleotides have no subscripts.
In figure 13.4 a complete phylogeny will require a network composed of one nucleotide from every node using only the links shown. Such a complete linkage will involve nucleotides with subscripts and possibly links bearing arrowheads. The sum of the arrowheads plus subscripts on any complete valid linkage is 4, the minimum number of replacements required to account for the data.
How does one discover all the possible linkages like in figure 13.4? First, the nucleotides without parentheses were placed in the nodal set during the preliminary phase, while those with parenthesis during the final phase. Such nucleotides will be represented as N_i and (N_i), where the subscript denotes one of the four nucleotides. An arrow is used to denote descent from an ancestor tot he next node. Thus $(N_i) \rightarrow N_j$ means that an ancestral nucleotide that was added to a nodal set during the final phase of reconstruction is replaced (since $i \neq j$) by a nucleotide that was originally assigned to the descendant nodal set in the preliminary phase. An example is $(A) \rightarrow (C)$ in figure 13.4.
The rules are then as follows: given that a particular ancestral nucleotide is i,

- **I**: $N_i \rightarrow N_i$ or $(N_i) \rightarrow N_i$ is obligatory i f the descendent N_i exists, if it does not, then:
- **II**: all possible linkages are permitted except $N_i \rightarrow (N_j)$ and $(N_i) \rightarrow (N_j)$

13.3.5 "Probabilities associated with specific links

The number of fixations (back mutation) ascribed to a single nucleotide position is the same for all of the most parsimonious tree, it should be clear from figure 13.4 that which nucleotide replaces which is a function of the particular set of valid linkages that is examined. What then is the best estimate of the weight that should be assigned to any given link in the most parsimonious tree for

the time period represent?
A caveat is necessary. In real world descendent sequences, there will be events for which there's no evidence (i.e. we compute the event $A \rightarrow G$, but it really went like $A \rightarrow C \rightarrow G$). The weighting procedure only accounts for evident events and the "probabilities" obtained sum up to 1. However, since the real world is less parsimonious than the more restricted computational world, every such probability necessarily overestimates the likelihood that that event actually occurred in the evolutionary history of that nucleotide position, hence the quotation marks. This computation is pursued anyway, since for most purposes we do not need to know the likelihood of events for which there is no evidence.

13.3.5.1 The procedure for calculating "probabilities"

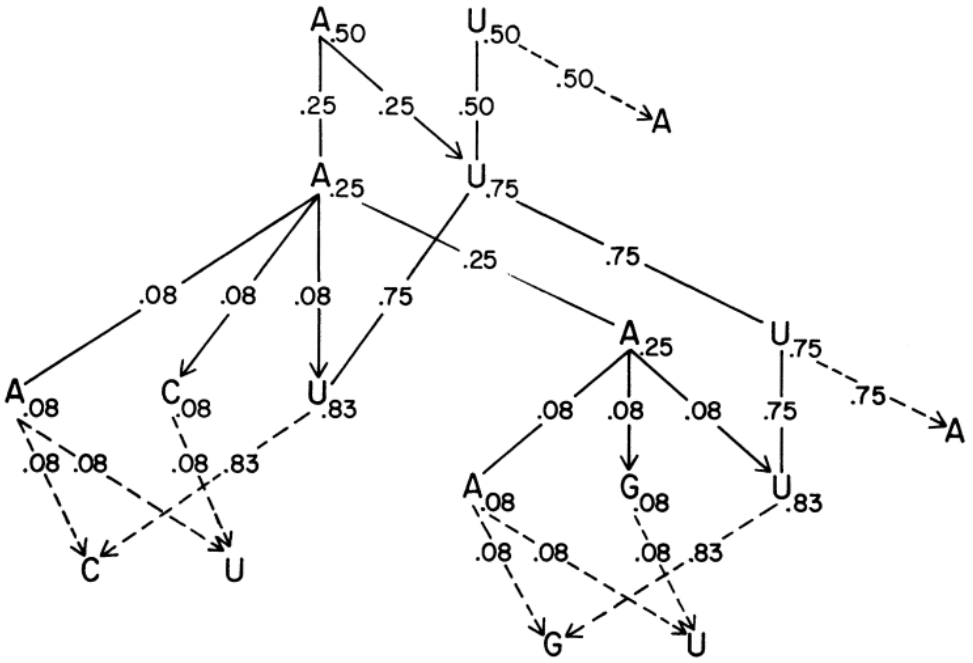


Figure 13.5: "Probability" that segments are part of the temporal sequence of descent.

Assumption: all permitted nucleotides for the ultimate ancestor are equiprobable (i.e., in figure 13.5 $P(^5A) = P(^5U) = 0.5$). This is to tackle the bias in favor of the predominating nucleotide in the line descending from the ultimate ancestor that has the fewest bifurcations. This bias comes from the assumption that all possible valid sets of complete linkages are equiprobable.
Our assumption also assumes that every valid link from a given nodal nucleotide, $L(^kN)$, is equiprobable. Thus, the probability that a link is correct is $P[L(^kN)] = P(^kN)/n$, where n is the number of links descending from ^kN.
In figure 13.5 the dotted lines are the probabilities associated with every link and every nodal nucleotide shown in figure 13.4.

13.3.5.1.1 Notes If you want to have a look at the algorithm step by step, take a look at the slides of lecture 14.

Chapter 14

Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach

14.1 Maximum likelihood estimation

In statistics, maximum likelihood estimation (MLE) is a method of estimating the parameters of an assumed probability distribution, given some observed data. This is achieved by maximizing a likelihood function so that, under the assumed statistical model, the observed data is most probable. The point in the parameter space that maximizes the likelihood function is called the maximum likelihood estimate.

14.1.1 Principles

We model a set of observations as a random sample from an unknown joint probability distribution which is expressed in terms of a set of parameters. The goal of maximum likelihood estimation is to determine the parameters for which the observed data have the highest joint probability. We write the parameters governing the joint distribution as a vector ector $\theta = [\theta_1, \theta_2, \ldots, \theta_k]^\mathsf{T}$ so that this distribution falls within a parametric family $\{f(\cdot\,;\theta) \mid \theta \in \Theta\}$ where Θ is called the parameter space, a finite-dimensional subset of Euclidean space. Evaluating the joint density at the observed data sample $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ gives a real-valued function,

$$\mathcal{L}_n(\theta) = \mathcal{L}_n(\theta; \mathbf{y}) = f_n(\mathbf{y}; \theta) \tag{14.1}$$

which is called the likelihood function. For independent and identically distributed random variables, $f_n(\mathbf{y}; \theta$ will be the product of univariate density functions:

$$f_n(\mathbf{y}; \theta) = \prod_{k=1}^{n} f_k^{\mathsf{univar}}(y_k; \theta) \ . \tag{14.2}$$

The goal of maximum likelihood estimation is to find the values of the model parameters that maximize the likelihood function over the parameter space,[6] that is

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}_n(\theta\,;\mathbf{y}) \ . \tag{14.3}$$

Intuitively, this selects the parameter values that make the observed data most probable. The specific value $\hat{\theta} = \hat{\theta}_n(\mathbf{y}) \in \Theta$ that maximizes the likelihood function $\mathcal{L}_n$ is called the maximum likelihood estimate. Further, if the function $\hat{\theta}_n : \mathbb{R}^n \to \Theta$ so defined is measurable, then it is called the maximum likelihood estimator. It is generally a function defined over the sample space, i.e. taking a given sample as its argument. A sufficient but not necessary condition for its existence is for the likelihood function to be continuous over a parameter space Θ that is compact.[7] For an open Θ the likelihood function may increase without ever reaching a supremum value. In practice, it is often convenient to work with the natural logarithm of the likelihood function, called the log-likelihood:

$$\ell(\theta\,;\mathbf{y}) = \ln \mathcal{L}_n(\theta\,;\mathbf{y}) \ . \tag{14.4}$$

14.1.2 Phylosophy

The maximum likelihood method finds that estimate of a parameter which maximizes the probability of observing the data given a specific model for the data.
Likelihood principle: "In statistics, the likelihood principle is the proposition that, given a statistical model, all the evidence in a sample relevant to model parameters is contained in the likelihood function."

14.1.3 ML for phylogenetic trees

Given a multiple sequence alignment and probabilistic model of for substitutions (like PAM does, but different) find the tree which has the highest probability of generating the data.
It was first proposed by Felsestein in 1981, assuming some simplifying assumptions:

- Positions evolved independently
- After species diverged they evolve independently

Formally, the task of the algorithm is to find the tree T such that assuming evolution model M $Pr[Data|T, M]$ is maximized.

14.2 Introduction

Problem with parsimony methods is that they implicitly assume that change is improbable a priori. If the amount of change is small over the evolutionary times being considered, parsimony methods will be well-justified statistical methods. Most data however involve moderate to large amounts of change, and it is in such cases that parsimony methods can fail.
Distance approaches estimate the tree from information on the pairwise similarity of the sequences (Fitch and Margoliash 1967), without attempting to make full use of the information available in the original sequences.

14.3 Computing likelihood of a tree

Maximum likelihood estimation is the method of statistical inference most readily applicable to data of this sort. It involves finding that evolutionary tree which yields the highest probability of evolving the observed data. Note that although the likelihood of a tree is the probability of the data given the hypothesis, it is taken as a function of the hypothesis (the tree) rather than a function of the data. This means that the likelihoods for different trees do not sum to unity. Note also that the likelihood of a tree is not the probability that the tree is the correct one.
The problem reduces to computing the probability of a particular set of sequences on a given tree and maximizing this probability over all evolutionary trees. The assumption of independent events, even if not biologically correct, makes the computation feasible. Since we are willing to assume independence of evolution at different sites, it turns out that the probability of a given set of data arising on a given tree can be computed site by site, and the product of the probabilities taken across sites at the end of the computation.
So we concentrate on computing this probability for a single site. For this we make use of the probabilities $Pij(t)$, where i and j take values 1,2, 3, and 4 corresponding to the four bases A, C, G, and T. $Pij(t)$ is the probability that a lineage which is initially in state i will be in state j after t units of time have elapsed. We compute the $Pij(t)$ later, for the moment we concentrate on obtaining an expression for the likelihood of the tree.
We assume that after speciation two lineages evolve independently, and that the same stochastic process of base substitution applies in all lineages. Instead of writing general expression for the likelihood of a tree, we will refer to a specific case, shown in figure 14.1.

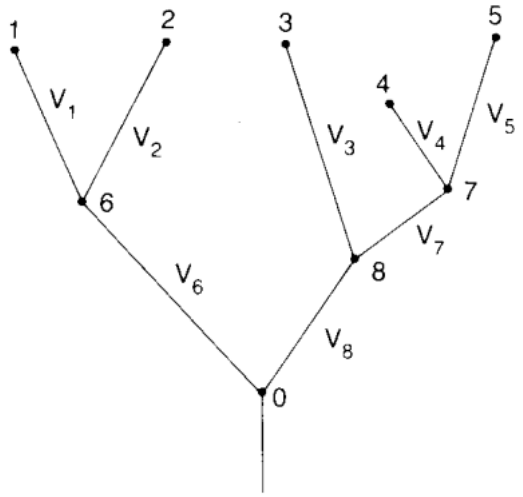


Figure 14.1: The tree used in the discussion of computing the likelihood. The v's are the lengths of the segments

The lengths of the segments of the tree are given by v_i. If we knew the states (bases) at a particular site at points 0, 6, 7 and 8 on this tree, and these were s_0, s_6, s_7, and s_8, the likelihood of the tree would be the product of the probabilities of change in each tree segment, times the prior probability π_{s_0} of state s_0. In practice we do not know s_0, s_6, s_7, and s_8, so the likelihood will be the sum over all possible assignments of bases to those forks on the tree:

$$L = \sum_{s_0} \sum_{s_6} \sum_{s_7} \sum_{s_8} \pi_{s_0} P_{s_0 s_6}(v_6) P_{s_6 s_1}(v_1) P_{s_6 s_2}(v_2) P_{s_0 s_8}(v_8) P_{s_8 s_3}(v_3) P_{s_8 s_7}(v_7) P_{s_7 s_4}(v_4) P_{s_7 s_5}(v_5) \tag{14.5}$$

This expression for n species will have 2^{2n-2} terms (in this case, 256). I will deliberately skip all the simplifications of the expression, what's important to know is that we can define $L_s^{(k)}$ as the likelihood based on the data at or above point k on the tree, given that point k is known to have state s for the site under consideration. If point k is a tip, then $L_s^{(k)}$ will be zero for all s except that actually observed, for which $L_s^{(k)}$. We work our way down the tree from the tips (we perform a postorder tree traversal). For point k, whose immediate descendants are i and j, we can compute for all four values of s_k

$$L_s^{(k)} = \big(\sum_{s_i} P_{s_k s_i}(v_i) L_{s_i}^{(i)} \big) \big(\sum_{s_j} P_{s_k s_j}(v_j) L_{s_j}^{(j)} \big) \tag{14.6}$$

For the bottom fork, point 0 in our example, we will then have computed the four conditional likelihoods $L_{s_0}^{(0)}$ given the possible states of the site at point 0.

14.4 The base substitution probabilities

We have not yet specified how the quantities $Pij(t)$ are to be computed. These are the probabilities of transition from one base to another over a segment of length t. We assume that these probabilities reflect a Markov process, a process in which the probability of a base changing may depend on its current identity, but not on its past history.
We assume that in a small interval of time of length dt, there is a probability udt that the current base at a site is replaced. The quantity u is the rate of base substitution per unit time. If a base is replaced, its replacement is A, C, G, or T with probabilities π_1 π_2, π_3, π_4. Note that this means that a base could be replaced by the same base, so that not all substitutions are observable even in principle. Note also that this model makes no distinction between transitions and transversions. If we let $delta_{ij}$ be 0 if $i \neq j$ and 1 if $i = j$ (the Kronecker delta function), then we are in effect assuming that for infinitesimal dt

$$P_{ij}(dt) = (1 - udt)\delta_{ij} + udt\pi_j \tag{14.7}$$

from this, we can derive

$$P_{ij}(dt) = e^{-ut}\delta_{ij} + (1 - e^{-ut})\pi_j \tag{14.8}$$

This follows almost immediately once one observes that e^{-ut} is the probability that the site does not change at all over a length of time t, and that if it does change the probability that it ends up in state j is π_j.

14.5 The pulley principle

The reversibility of our Markov process and the absence of constraints on segment lengths can be used to establish an interesting and useful property of the estimation of evolutionary trees under this model. Reversibility requires that for all i,j, and t:

$$\pi_i P_{ij}(t) = P_{ji}(t)\pi_j \tag{14.9}$$

Consider the last two steps of our algorithm for calculating likelihoods. They involved (in our example) forks 0, 6, and 8 in the expression for the likelihood of the tree at one site. Since the likelihood L is our only basis for comparing evolutionary trees, this means that the tree whose likelihood is being calculated could have its root anywhere between points 6 and 8, as shown in figure 14.2a. The root of the tree is a sort of pulley, so that if all parts of the tree to one side of the root are moved down, and all parts to the other side moved up by the same amount, the likelihood remains unaltered.

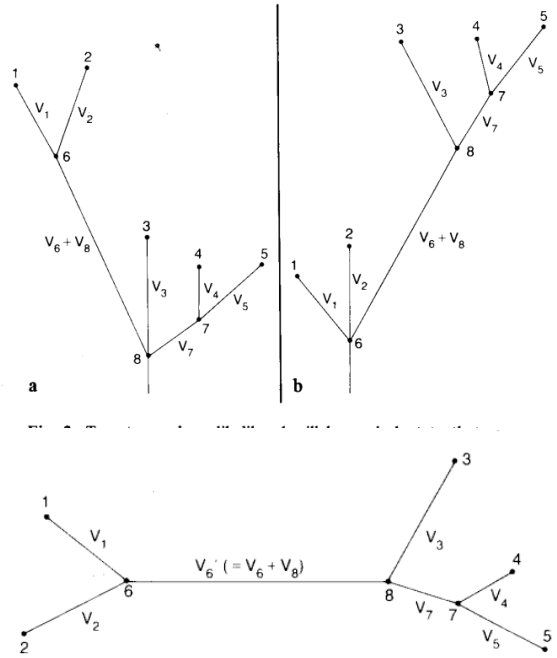


Figure 14.2: Visual representation of the pulley principle. (a) Two trees whose likelihood will be equivalent to that in Fig 14.1 under the assumption of this paper (b) The unrooted tree whose likelihood is equivalent to those in Fig 14.1 an 14.2a

Figure 14.2b shows the unrooted tree, which is what we are in effect estimating. The root of the tree can be placed anywhere on that tree without affecting the likelihood.

14.6 Finding the Maximum Likelihood tree

Our interest in the Pulley Principle is that it allows us to regard any given segment of the unrooted tree as contraining the root. This in turn allows us to alter the length of that segment in an optimal fashion, which will be useful. We have a computationally feasible method for evaluating the likelihood of a given tree, but this leaves us with the task of finding the maximum likelihood tree. Consider the problem of finding values of the v_i which maximize the likelihood of the tree given a particular topology. The pulley principle allows us to construct an algorithm which alters one of the

v_i at a time, each one being altered to that value which results in the highest likelihood. This process continues until none of the v_i can be altered in a way which substantially improves the likelihood. At each stage one v is changed to the value which gives the greatest possible likelihood, given that only that v can be varied. Thus at each step the likelihood of the tree increases.

14.7 Searching among tree topologies

There still remains the problem of examining many different tree topologies. The strategy proposed is to build the tree up by successively adding species to it, starting with a two-species tree. When the k-th species is being added to the tree, there will be 2k-5 segments from which it could arise. Each of these is tried and the maximum likelihood within the resulting topology evaluated, by the iteration technique presented below. The placement yielding the highest likelihood is accepted. If the tree now has more than four species, before the next species is added local rearrangements are carried out in the tree to see if any of these improves the likelihood of the tree. If any does, it is accepted and the local rearrangement process continues until a tree is found which no local rearrangement can improve. This strategy of searching among possible topologies is not guaranteed to find the best topology.

14.8 Finding optimum segment lengths

Within each topology we examine, we must adjust the v_i to their maximum likelihood values. This is done by adjusting each of the v_i in turn according to a method which guarantees that each of the v i changes to that value which maximizes the likelihood of the tree, given the current values of the other v's. Consider segment 7 of the tree in figure 14.2b, the segment connecting nodes 7 and 8. We can consider the root to be located in this segment, and use the pruning algorithm given above to compute the sets of conditional likelihoods at points 7 and 8. Suppose that we take the root to be immediately to the right of point 8. The likelihood of the tree for one site is then given by

$$L = \sum_{s_0}\sum_{s_6}\sum_{s_7}\sum_{s_8} \pi_{s_0}\left(P_{s_0 s_8} L_{s_8}^{(8)}\right)\left(P_{s_0 s_7}(v_7) L_{s_7}^{(7)}\right) \tag{14.10}$$

$$= \sum_{s_0} \pi_{s_0} L_{s_0}^{(8)} \left(\sum_{s_7} P_{s_0 s_7}(v_7) L_{s_7}^{(7)}\right. \tag{14.11}$$

If we substituite into (14.10) the expression (14.8) with $u = 1$, we will obtain an expression that is the factor of the full likelihood which corresponds to one site[1]. There is one factor like this for each site in the DNA, so that the full likelihood is of the form

$$L = \prod_i (A_i q + B_i p) \tag{14.12}$$

where $q = e^{v_7}$, $p = 1 - q$ and A_i and B_i are the terms

$$A_i = \sum_s \pi_s L_s^{(8)} L_s^{(7)} \tag{14.13}$$

[1] not reported here, it's expression 11 in the original paper

and

$$B_i = (\sum_{s_8} \pi_{s_8} L_{s_8}^{(8)})(\sum_{s_7} \pi_{s_7} L_{s_7}^{(7)}) \tag{14.14}$$

for the i-th DNA site. We want to find the value of v_7 which maximizes the likelihood. We can the proceed by taking the logarithm of equation 14.12 and equating the derivative of this expression to zero. All the steps of this very last part are reported in the paper.