Multiple sequence alignment

In biology we are frequently faced with the problem of aligning multiple sequences together, e.g. we have managed to extract the sequence of a gene of interest from multiple organisms, and want to find out how these different versions of the sequence relate to each other.

Specifically, we want to create a matrix where each sequence is represented as a row, and each column represents the alignment of characters from the original sequences, also including gap characters as necessary. Note: we generally require that at least one character in each column is not a gap.

```
Seq1  ACC-AGTGA-CT
Seq2  -CGTAGCGA-CT
Seq3  ACC-AGTGT-CT
Seq4  ACGTAGTCATCT
```

Unlike the pairwise sequence alignment problem, the multiple alignment problem is a bit harder to formalize. A commonly used theoretical framework states the problem as follows:

Multiple Alignment Problem. Given a set of n sequences s_1 .. s_n, find the multiple alignment M that minimizes the sum of the pairwise distances between these sequences:

$$\sum_{s_i, s_j \in S} d_M(s_i, s_j)$$

$d_M(s_i, s_j)$ is the distance between sequences s_i and s_j as implied by the multiple alignment M, i.e. the sum of the distances between the characters of the two sequences that are aligned to each other in the multiple alignment. Generally, alignments between gap characters are assigned score 0. The score of alignments between individual nucleotides or amino-acids depends on the specific context: in protein alignments the scores can be derived from the BLOSUM matrix scores, while in phylogenetic analysis of nucleotide sequences, the alignment scores are determined by an appropriately defined evolutionary model.

Note that $d_M(s_i, s_j)$ is generally larger than $D(s_i,s_j)$ - the optimal pairwise distance between two sequences (as defined by the pairwise alignment problem discussed earlier in the class).

The Multiple Alignment Problem can be "easily" solved through an extension of the dynamic programming algorithm for pairwise sequence alignment. The intuition is to extend the V(i,j) values from the 2-sequence context, to V(i, j, k) for 3 sequences, etc. The resulting algorithm runs in $O(L^n)$ where L is the length of the aligned sequences, and n is the number of sequences. Clearly, this algorithm is impractical, though some run-time improvements can be obtained by carefully pruning the dynamic programming table.

In general, the Multiple Alignment Problem is NP-hard, leading to the need for heuristic approximation algorithms.

One such algorithm is called Star Alignment - deriving its name from the fact that the multiple alignment is constructed by aligning all the sequences to a same "center" sequence. The alignment process is called "progressive alignment" and proceeds as follows:

1. Construct alignment of sequence 1 with the center sequence (using standard pairwise alignment)
2. Construct alignment of sequence 2 with center sequence (using standard pairwise alignment). Any gaps inserted within the center are propagated to the already constructed alignment with sequence 1.
3. repeat 2 until all sequences have been aligned

Note: the implementation can be a bit tricky as you need to avoid having spurious gaps inserted in the alignment - e.g. if two sequences both lead to the insertion of a gap at the same location in the center sequence, the alignment algorithm should ensure a single gap is inserted (rather than two adjacent gaps).

Choice of the center sequence. For reasons that will become apparent below, the center sequence s_c is chosen to be the sequence that minimizes the equation

$$\sum_{i \neq c} D(s_i, s_c)$$

over all possible choices of s_c.

Theorem: If the distance between sequences satisfies the triangle inequality, the star alignment has a score at most 2 * OPT, where OPT is the score of the optimal multiple alignment.

Proof:

Let S* be the "sum of pairs" score of the star alignment.

$$S^{star} = \sum_{i \neq j} d_{star}(s_i, s_j) \leq \sum_{i \neq j} (D(s_i, s_c) + D(s_j, s_c)) = 2(n-1) \sum_i D(s_i, s_c)$$

where n is the number of sequences and d_{star} is the distance implied by the star alignment.

Let OPT be the score of the optimal alignment.

$$OPT = \sum_{i \neq j} d_{OPT}(s_i, s_j) = \sum_j \sum_i d_{OPT}(s_i, s_j) \geq \sum_j \sum_i D(s_i, s_c) = n \sum_i D(s_i, s_c)$$

Combining the two equations above we get $\frac{S^{star}}{OPT} \leq 2 - \frac{2}{n} < 2$

Tree-guided alignment

Neither the optimization problem implied in the definition of the multiple alignment problem, nor the approximation provided by the star alignment have a biological interpretation. As a corollary, the resulting multiple alignments may not capture interesting biological phenomena. A more "biologically meaningful" approach to multiple alignment relies on the construction of a guide tree that, intuitively, captures the evolutionary relationship between the sequences being aligned. The guide tree stores the sequences at its leaves, and the progressive alignment approach described in the context of the star alignment can be modified to allow sequences to be merged together in a bottom-up fashion into a single multiple alignment.

Virtually all "popular" multiple alignment programs rely on this paradigm. Two heavily used

programs are ClustalW and Muscle (papers linked from the class website).

Multiple alignment of genomes

The problem: so far we've discussed global multiple alignment - the sequences have to be aligned end-to-end. This approach simply doesn't scale to whole genomes. Also, whole genomes have rearrangements, events that are not captured in the typical edit-distance alignment algorithms.

Example: the Mauve system - http://asap.ahabs.wisc.edu/mauve/

Basic idea:

1. Find a collection of MUMs shared by 2 or more genomes (multi-MUMs).
Can think of a variety of ways of doing this, but in principle the basic idea is to first find exact k-mer matches across multiple genomes (making sure none is a repeat in any of the genomes) then extend these matches along groups of genomes as long as the corresponding sequences agree.

2. Find locally collinear blocks (LCB) among the MUMs - these are sets of MUMs that occur in the same order in all genomes. Note: the LCB algorithm requires all MUMs to occur in all genomes so it's only applied to those MUMs.

The idea of the LCB algorithm is to repeatedly sort the MUMs based on their occurrence in each genome, and mark breakpoints in this sorting - places when the MUMs sorted by genome i occur out of order in genome j.

3. A global guide-tree is constructed from all genomes, using the weight of the shared MUMs as a similarity/distance measure. This tree is then used to align the regions between the MUMs using ClustalW.

Local multiple alignment (motif finding)

How could one define a local multiple alignment in the same way that we talk about local inexact alignment? In Mauve, the MUMs are in effect short local multiple alignments, but they are exact. Can this concept be extended to inexact matching?

Problem: *Given k sequences of length L, find a sequence of length M that occurs nearly the same in all sequences*

Note: We ignore gaps here.

Note: This looks fairly difficult - we need to pick from among $(L-M+1)^k$ different possible multiple alignments and find the best one.

One solution: Gibbs sampling.

Algorithm.

1. Pick a random subsequence from each of the k sequences

2. Construct a multiple alignment - represented as a "profile" - frequency of each nucleotide at each position ($p_{i,j}$ = probability of observing nucleotide i at position 1<=j<=M in the alignment)

3. Pick a random sequence s from among the k sequences and find the length M subsequence that best matches the alignment profile computed at 2.

Score of the match is $\sum_{1 \le j \le M} p_{ij} \log \frac{p_{ij}}{q_i}$ where q_i is the frequency of nucleotide i in the whole sequence, and nucleotide i in the sum above is simply the nucleotide from sequence s that is aligned to the profile.

4. Replace the best matching sequence from s within the multiple alignment and repeat step 3 until convergence.

Profile representation of multiple alignments

The basic idea from the Gibbs sampler is that a multiple alignment can be represented as a profile - simply the frequencies of observing each nucleotide in each column of the alignment.

Note: It is easy to modify the traditional dynamic programming sequence alignment algorithm to compute an alignment between a sequence and a profile (use as alignment scores the weighted average of the scores for the nucleotides found in the profile column).

Note: It is equally easy to modify the traditional dynamic programming sequence alignment algorithm to align two profiles to each other.

Thinking of a multiple alignment as a profile is useful in the "search" context when we want to find a sequence that matches an existing multiple alignment. It is also useful in guiding the progressive multiple-alignment process.

Different variants of the profiles are called Position Weight Matrices (PWM), or Position Specific Score Matrices (PSSM). These usually are a bit more sophisticated than simply counting the frequency of each symbol (nucleotide or amino-acid) in each column. Specifically, the "score" of nucleotide i in column j is represented as a log-odds score:

$$\log(\frac{p_{ij}}{q_i})$$

where p_{ij} is the probability of observing nucleotide i in column j, and q_i is the overall frequency of nucleotide i.

The p_{ij} values are also not simply the observed frequency of a nucleotide at a particular column in the alignment. Pseudo-counts are often added to each column's symbol counts to account for the fact that certain symbols have not been observed in the sample represented by the sequences

present in the alignment. An example (used by the program PSI-Blast) is to compute

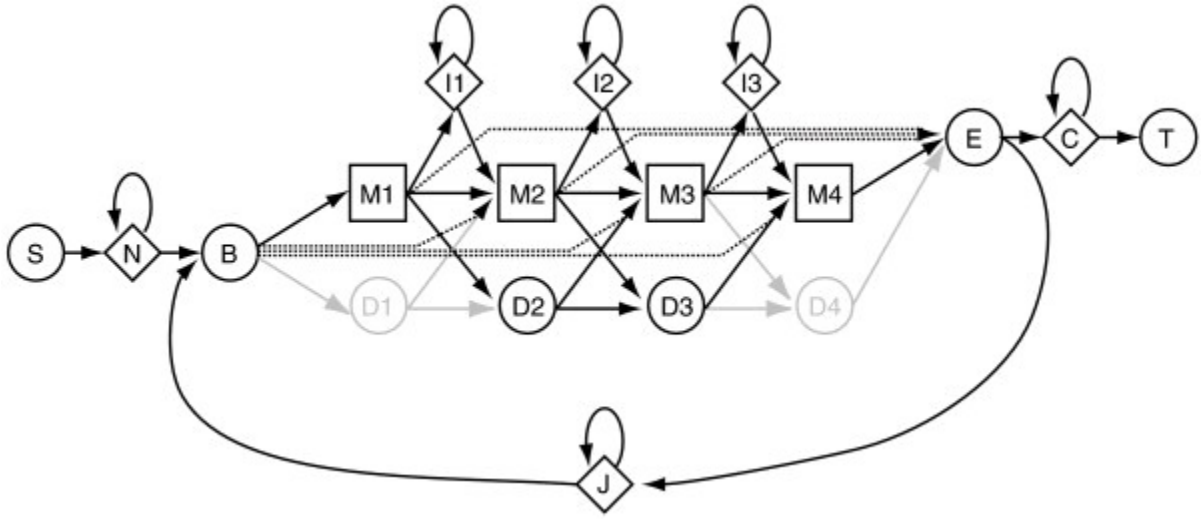
$$p_{ij} = \frac{f_{ij} + b\,q_i}{n + b}$$

where f_{ji} is the frequency of symbol i within alignment column j, b is the pseudo-count (a small number), q_i is the overall frequency of symbol i, and n is the number of sequences (rows) in the multiple alignment.

Profile HMMs.

Implicit in the alignment profiles described above is the fact that columns in the alignment are independent of each other. This is not necessarily true in biologically-relevant alignments, where we might want to model some contextual dependence between alignment columns. Profile Hidden Markov Models (HMMs) handle this specifically in the context of aligning one sequence against an existing multiple alignment (e.g. find a gene from a given gene family - represented as a multiple alignment - within a genome)

I will skip for now an introduction to Hidden Markov Models and simply show the structure of the HMM used for alignment (see figure below)



(image from Newberg *BMC Bioinformatics* 2009 **10**:212 doi:10.1186/1471-2105-10-212)

In the HMM shown above you can distinguish three types of states (just focus in the center region until I can draw a better picture).

The square states represent columns in the multiple alignment, corresponding to a match between characters of the sequence being aligned and the alignment. The emission probabilities for these states are determined from the nucleotide/amino-acid frequencies at the corresponding column.

The circle states indicate deletions - skipped columns in the alignment. These have no emission probabilities.

The diamond states indicate insertions - characters in the aligned sequence that do not fit in the

alignment.

The alignment of a sequence to the multiple alignment is computed by a simple application of the Viterbi algorithm.