Global Multiple Sequence Alignment

Dannie Durand

In multiple sequence alignment, the goal is to align k sequences, so that residues in each column share a property of interest, typically descent from a common ancestor or a shared structural or functional role. Applications of multiple sequence alignment include identifying functionally important mutations, predicting RNA secondary structure, and constructing phylogenetic trees.

Given sequences s_1,...,s_k of lengths n_1,...,n_k, $\alpha = \{s'_1,...,s'_k\}$ is an alignment of s_1,...,s_k if and only if

- $s'_a \in (\Sigma')^*$, for $1 \leq a \leq k$
- $|s'_a| = l$, for $1 \leq a \leq k$, where $l \geq \max(n_1, \ldots, n_k)$
- s_a is the sequence obtained by removing gaps from s'_a
- No column contains all gaps

Scoring a multiple alignment

As with pairwise alignment, multiple sequence alignments (MSAs) are typically scored by assigning a score to each column and summing over the columns. The most common approach to scoring individual columns in a multiple alignment is to calculate a score for each pair of symbols in the column, and then sum over the pair scores. This is called sum-of-pairs or SP-scoring. For global multiple sequence alignment, SP-scoring can be used with either a distance metric or a similarity scoring function. The sum-of-pairs similarity score of an alignment of k sequences is

$$S_{sp}(s'_1, \ldots, s'_k) = \Sigma_{i=1}^{l} \Sigma_{a=1}^{k} \Sigma_{b>a}\ p(s'_a[i], s'_b[i]), \tag{1}$$

where l is the length of the alignment. As before, $p(x, y)$ is a numerical score that represents the similarity of x and y and $p(x, \_)$ is the gap score. Further, we define $p(\_, \_)$ to be zero. In pairwise alignment there is no need to assign a value to $p(\_, \_)$ because the definition of a pairwise alignment specifies that no column may contain two gaps. However, in a multiple alignment, two aligned sequences can have a gap in the same column (i.e., $s'_a[i] = s'_b[i] = \_$), as long as there exists at least one sequence in the MSA that does not have a gap in that column. As an example, let us calculate the SP-score for the alignment of three sequences shown below:

```
s1  A_TT
s2  A_T_
s3  ACAT
```

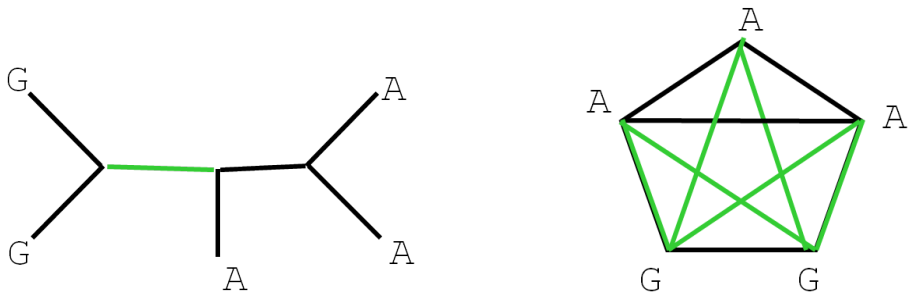
We can calculate the SP-score for each column separately:

	A_TT A_T_ ACAT
s_1, s_2	MOMg
s_1, s_3	MgmM
s_2, s_3	Mgmg

Note that the second column contains two gaps and that these are assigned a score of zero. The total SP-score is $5M + 2m + 4g$. (Is this alignment optimal? If not, how could you improve it?)

We can also use sum-of-pairs with distance scoring for global multiple alignment. This is how we would score the same alignment using unweighted edit distance:

	A_TT A_T_ ACAT
s_1, s_2	0001
s_1, s_3	0110
s_2, s_3	0111



(a) (b)

Figure 1: Two ways of scoring the column (A, A, A, G, G) in a multiple alignment. Green edges represent mismatches. **(a)** Scoring mutations on a tree. **(b)** Sum-of-pairs scoring

Sum-of-pairs scoring tends to overestimate the number of mutations required to explain the data. For example, a single mutation is required to explain the column (A, A, A, G, G), when scored on the tree in Fig. 1a. In contrast, SP-scoring assigns this column a score of six (Fig 1b), because SP-scoring is based on the implicit assumption that each pair of symbols is independent of all other pairs.

Scoring an alignment on a tree, also known as tree alignment, is based on the assumption that the residues in the columns of the multiple sequence alignment share an evolutionary history and that this history can be expressed as a single tree for all columns.

Given a known tree topology as input, the k extant sequences are associated with the k leaves of the tree. Sequences for the internal nodes are selected such that the sum of edge costs, i.e. the total number of mutations required along the branches of the tree, is minimized. Under this model, the cost of an edge $(X_i; X_j)$ in the tree is the minimum number of mutations required to transform sequence X_i into sequence X_j.

In order to use this approach, several issues must be resolved. First, a tree topology is needed. In general, the underlying tree is not known. In fact, multiple sequence alignments are generally used to estimate evolutionary trees and not vice versa. Second, it is generally assumed that every column in the alignment has the same underlying tree topology.For many sequences, such as those that that have undergone domain shuffling, this is not the case. Third, in order to compute the branch costs of the tree, we must infer the ancestral sequences associated with the internal nodes. Tree alignment has historically been based exclusively upon the parsimony criterion; that is, on the assumption that mutations are rare and the minimum number of evolutionary steps required to explain the data is the best evolutionary hypothesis. Data that does not happen to be parsimonious can favor the wrong tree model. In addition, column-oriented optimization approaches to MSA usually assume that sequence positions are independent and identically distributed. In general, these assumptions frequently do not hold for biological sequence data.

Is a tree the correct model for describing the relationship between residues in each column? A tree may not be a suitable model because the relationship between residues is functional or structural rather than historical. When alignment is used to study function or structure, residues in a column do not always share a common ancestor. The goal is to align residues that share the same role. Although functional or structural residues usually share an evolutionary history, sometimes functional or structural roles can migrate to neighboring residues.

For all of these reasons, tree alignment is rarely used in practice.

Given two sequences s_a and s_b in a multiple alignment, the pairwise alignment of s_a and s_b induced by the MSA, is the alignment obtained by deleting the other sequences in the MSA and then removing any column that contains two gaps. For example, in the multiple alignment below,

```
AC_T_G
A_GT_G
ACGTAG
```

the induced alignment of the first two sequences is

```
AC_TG
```

```
A_GTG.
```

Further, the pairwise alignment induced by the optimal multiple alignment is not necessarily the optimal pairwise alignment. In this example, the optimal pairwise alignment is

```
ACTG
AGTG.
```

Although the optimal pairwise alignment may have a better score, the induced pairwise alignment may be a biologically more realistic alignment because it reflects properties of the family as a whole.

A dynamic programming algorithm for multiple alignment

The dynamic programming algorithm used for finding the optimal global alignment of two sequences can be extended to the problem of global alignment of k sequences. First, let us consider a dynamic program to design three sequences. When a sum-of-pairs similarity score is used, the dynamic program for three sequences looks like this:

Input:

Sequences s_1, s_2, and s_3 of lengths n_1, n_2, and n_3, respectively.

Initialization:

$$\begin{aligned} a[i_1, 0, 0] &= a[i_1 - 1, 0, 0] + g \\ a[0, i_2, 0] &= a[0, i_2 - 1, 0] + g \\ a[0, 0, i_3] &= a[0, 0, i_3 - 1] + g \end{aligned}$$

Recurrence:

$$a[h, i, j] = \max \begin{cases} a[i_1 - 1, i_2, i_3] + 2g \\ a[i_1, i_2 - 1, i_3] + 2g \\ a[i_1, i_2, i_3 - 1] + 2g \\ a[i_1 - 1, i_2 - 1, i_3] + 2g + p(s_1[i_1], s_2[i_2]) \\ a[i_1 - 1, i_2, i_3 - 1] + 2g + p(s_1[i_1], s_3[i_3]) \\ a[i_1, i_2 - 1, i_3 - 1] + 2g + p(s_2[i_2], s_3[i_3]) \\ a[i_1 - 1, i_2 - 1, i_3 - 1] + p(s_1[i_1], s_2[i_2]) + p(s_1[i_1], s_3[i_3]) + p(s_2[i_2], s_3[i_3]) \end{cases}$$

Store the indices of the entry in a that maximize the right hand side of the recurrence in a trace-back matrix, $\mathcal{T}$.

Trace back:

From $\mathcal{T}[n_1, n_2, n_3]$ to $\mathcal{T}[0, 0, 0]$ to obtain the optimal alignment.

Output:

The optimal alignment score, $a[n_1, n_2, n_3]$.

The optimal alignment of s_1, s_2, and s_3 with respect to similarity function, $\mathcal{S}$.

The dynamic program for multiple sequence alignment has the same structure as the algorithms for pairwise sequence alignment, but the initiation and recurrence steps are more complex. Since the alignment matrix, a, is a 3-dimensional matrix, the first row in each of the three dimensions must be initialized. The algorithm calculates the entries in a according to the recurrence proceeding diagonally from $a[0, 0, 0]$ to $a[n_1, n_2, n_3]$. As in the pairwise case, a trace-back matrix, $\mathcal{T}$, is used to record the indices that gave the optimal score for each i_1, i_2, i_3 prefix. Once the entire matrix has been filled in, the optimal score is found in $a[n_1, n_2, n_3]$. Note that this is analogous to the pairwise global alignment algorithm, where $a[n, m]$ contains the optimal score.

It is straightforward, if messy, to generalize the dynamic program for three sequences to a dynamic program for k sequences. To convince yourself that you understand how this works, try writing down the algorithm for four sequences. For three sequences, the recurrence has seven entries. How many entries will there be in the recurrence when $k = 4$? How many entries will there be for arbitrary k?

The computational complexity of the dynamic programming algorithm to align k sequences is $O(n^k 2^k k^2)$. To see this, note that for k sequences, the alignment matrix has $O(n^k)$ entries. For each entry in a, the recurrence relation considers $O(2^k)$ neighboring cells. Calculating the SP-score for each of those neighbors requires $O(k^2)$ time. Thus, the time complexity of the dynamic program for multiple sequence alignment is exponential in the number of sequences. Given 10 sequences of length at most 500, it is possible to calculate the optimal alignment using dynamic programming. For larger problem instances, it is necessary to use a heuristic.

Heuristics for global multiple alignment

The dynamic programming approach to global multiple sequence alignment is framed as an optimization problem. In this approach, we design an optimization criterion over the set of all possible MSAs and then to seek the MSA that optimizes this crterion. The advantage of this optimization approach is that the optimization criterion makes explicit the assumptions upon which the optimization is based. Because they are explicit, these assumptions are open to scrutiny and falsification.

However, this formal optimization approach has disadvantages as well. One, as we have already seen, is that the computational complexity is exponential in the number of sequences. A second problem is the selection of an optimization criterion. In computational biology, the optimization

criterion must follow a specific biological model relating the data to the evolutionary, structural, or functional question at hand. If this is not the case, then the optimal solution may not reflect biological relationships. As we have seen, both sum-of-pairs and tree alignment have limitations in how well they capture the underlying biology.

In practice, the most widely used multiple alignment programs are based on heuristic methods, not only because of the exponential running time of the exact algorithm, but also because heuristics often give MSAs that are better biologically, even though they do not guarantee mathematically optimal alignments. A survey of multiple alignment software based on heuristic methods, *Protein multiple sequence alignment* by Do and Katoh, 2008, is posted in the "Optional reading" section of the course syllabus.

The performance of MSA programs is typically evaluated empirically using benchmarks based on curated or automated structural alignments. BALIBase (http://www.lbgi.fr/balibase/), a collection of "high quality, manually refined, reference alignments based on 3D structural superpositions"[1], is one of the most widely used benchmarks. The BALIBase reference data sets are designed to mimic properties of different types of data sets encountered in practice, especially those that are challenging to align. Examples of challenging data sets include highly divergent sequences that are variable in length and have less than 50% identity, related sequences combined with several outlier, or "orphan", sequences, and related sequences that differ due to large insertions, deletions or terminal extensions.

In class, we reviewed the general structure of *progressive alignment*, one of the most commonly used heuristic strategies. This approach is used in a number of programs, including the widely-used CLUSTAL family of multiple alignment programs. Given k sequences, $s_1, \ldots, s_k$, of lengths $n_1, \ldots, n_k$, progressive methods construct an alignment as follows:

1. Construct pairwise alignments for all pairs of sequences
2. Compute $\hat{D}$, the matrix of pairwise distances, where $\hat{D}[a, b]$ is the distance between sequences s_a and s_b. Note that $\hat{D}$ is a symmetric matrix with zeros on the diagonal.
3. Construct a "guide tree", T, from $\hat{D}$. T is a rooted tree with k leaves corresponding to the k sequences.
4. Construct an MSA by repeatedly merging intermediate multiple alignments to obtain progressively larger alignments, until all k sequences have been incorporated in the alignment. The order of merging is determined by the guide tree, T.

The merge operation in Step 4 takes as input two multiple alignments of k_1 sequences and k_2 sequences and returns a multiple alignment of $k_1 + k_2$ sequences. This is repeated until all k sequences are incorporated into the alignment. The order in which sequences are merged is determined by a bottom up traversal of the guide tree. For example, if the tree in Fig. 2 were the guide tree, then

[1] "BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark." Thompson JD, Koehl P, Ripp R, Poch O., Proteins. 2005 Oct 1;61(1):127-36.

the pairwise alignment of s_1 and s_2 would be merged with pairwise alignment of s_3 and s_4, yielding an intermediate MSA of four sequences. A similar merging of two pairwise alignments would result in the MSA of s_5, s_6, s_7 and s_8. Finally, these two alignments, of four sequences each, would be merged to obtain a full alignment of eight sequences.

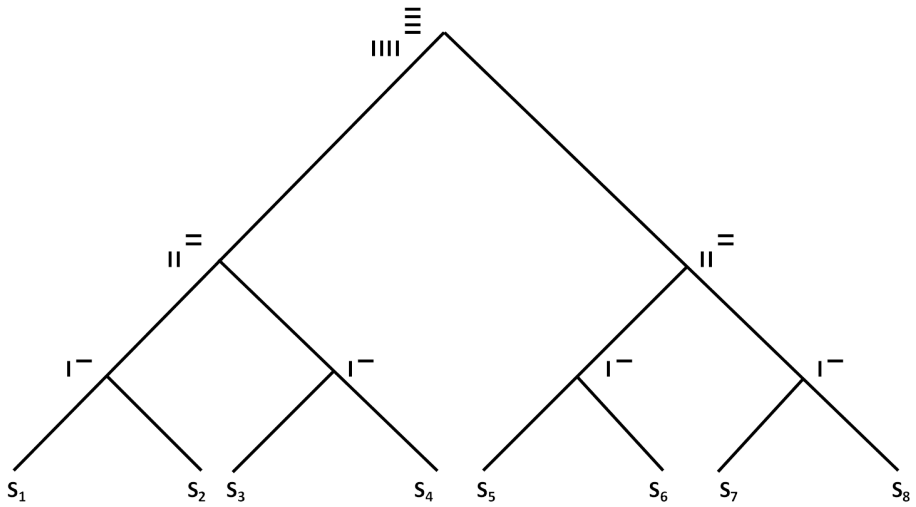


Figure 2: A balanced guide tree for 8 sequences.

The actual merge operation is carried out by using the pairwise global alignment algorithm to align the two alignments, where the input alignments are treated as sequences over an expanded alphabet. Durbin calls this a "profile." For example, two aligned sequences can be viewed as a sequence over the alphabet $\Sigma' \times \Sigma' \setminus \{(\_\ \_)\}$. When $\Sigma = \{A, C, G, T\}$, this expanded alphabet contains 24 symbols ($\{AA, AC, AG, AT, A\_, CA, \ldots\}$).

We illustrate profile alignment with the case where a multiple alignment of three sequences is obtained by aligning a profile $s[1, .., m]$ with a single sequence $t[1, .., n]$. The elements in $s[i]$ are of the form $\binom{x}{y}$, $\binom{x}{-}$, or $\binom{-}{y}$, $x, y \in \Sigma$. The score for aligning $s[i]$ with $t[j]$ is the sum of the scores for aligning the first character in $s[i]$ with $t[j]$ and the second character in $s[i]$ with $t[j]$. For example, when similarity scoring is used and $s[i]$ is of the form, $\binom{x}{y}$, the recurrence relation for calculating the alignment matrix $a[i, j]$ is

$$a[i,j] = \max \left\{ \begin{array}{l} a[i-1, j-1] + p(x, t[j]) + p(y, t[j]), \\ a[i, j-1] + 2g, \\ a[i-1, j] + 2g \end{array} \right\}. \tag{2}$$

When s[i] contains an indel (i.e., $s[i]$ is of the form $\binom{x}{-}$), the recurrence relation for calculating $a[i, j]$ is

$$a[i,j] = \max \left\{ \begin{array}{l} a[i-1, j-1] + p(x, t[j]) + g, \\ a[i, j-1] + 2g, \\ a[i-1, j] + g \end{array} \right\}. \tag{3}$$

Note that $p(x, y)$, the similarity of x and y, is not included in the right hand side of Equation 2. Similarly, the gap score for the alignment of $\begin{smallmatrix} x \\ - \end{smallmatrix}$ is not included in the recurrence in Equation 3. This is because the two symbols in $s[i]$ were compared and scored during the pairwise alignment in a previous step in the progressive alignment procedure.

A key aspect of the merging operation is that we are not allowed to modify the profiles being aligned. For the example above, that means we cannot change juxtaposition of the two symbols in $s[i]$, even if it results in a better alignment with t. This is called the "once a gap, always a gap" rule, although it also applies to mismatches. The consequence of this rule is that a bad decision with regard to the placement of gaps that is made early in the procedure, propagates through subsequent iterations and cannot be corrected. It is this rule that makes progressive alignment a fast heuristic; that is, this rule underlies both the improvement in running time and the possibility of a suboptimal result.

The complexity of progressive alignment is $O(k^2n^2)$, where $n = \max\{n_a\}, 1 \leq a \leq k$. Calculating the distance matrix in Step 2 requires $O(k^2)$ pairwise alignments. The cost of each pairwise alignment is $O(n^2)$. The merging process also requires $O(k^2n^2)$ time. The computational complexity of merging depends on the number of profile alignments required, the number of cells in the alignment matrix for each profile alignment, and number of comparisons required for each cell. The size of alignment matrix is $O(n^2)$ in all profile alignments. The number of comparisons for a single cell in an alignment of two profiles with k_1 and k_2 sequences, respectively, is $O(k_1 \cdot k_2)$.

The number of profile alignments and the values of k_1 and k_2 depend on the shape of the guide tree. At one extreme, we have a completely unbalanced guide tree with k sequences (e.g., Fig. 3, where $k = 8$). In this case, each merge represents an alignment of a single sequence ($k_1 = 1$) with a profile of size $k_2 = h$, $h = 1 \ldots k - 1$, resulting in a complexity of

$$\sum_{h=1}^{k-1} n^2 h.$$

This reduces to

$$n^2 \cdot \frac{(k-1)(k-2)}{2},$$

which is $O(k^2n^2)$.

At the other extreme, we have a balanced guide tree with k leaves (e.g., Fig. 2). For convenience, we assume that k is a power of 2. We leave the case where k is not a power of two to the masochistic reader. In a balanced guide tree, there are $k/2^h$ merges at height, h. Each of these represents an alignment of two profiles, each comprising 2^{h-1} sequences. The computational complexity is the sum of the complexity of the merges at height h, $1 \leq h \leq \log_2 k$, or

$$\sum_{h=1}^{log_2 k} n^2 \cdot (2^{h-1})^2 \cdot \frac{k}{2^h}.$$

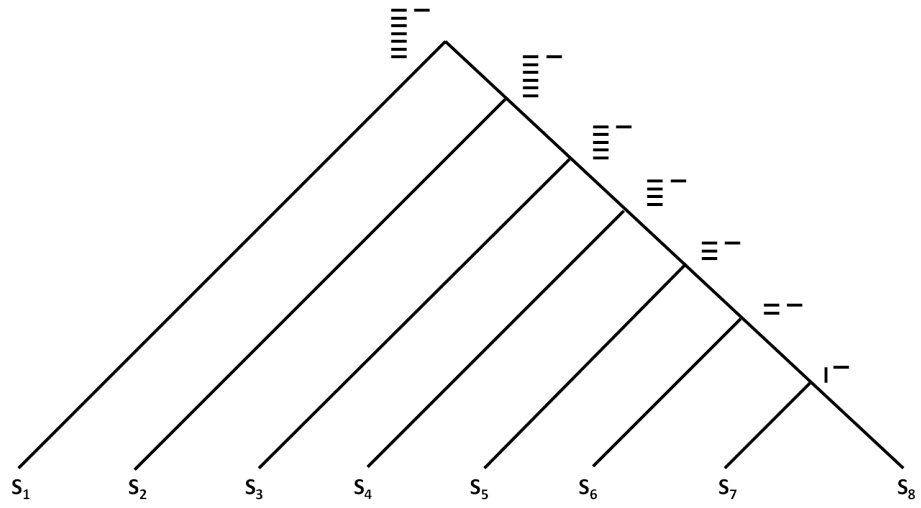


Figure 3: An unbalanced guide tree for 8 sequences.

This reduces to

$$kn^2 \cdot \sum_{h=1}^{log_2 k} 2^{h-2}. \tag{4}$$

It is easy to verify that the series

$$\sum_{i=1}^{N} 2^{(i-1)} = 2^N - 1. \tag{5}$$

Substituting the right hand side of Equation 5 into Equation 4, where $N = \log_2 k$, we obtain

$$\frac{1}{2}\, kn^2(2^{(\log_2 k)} - 1).$$

This reduces to $(k-1)kn^2/2$, so we again obtain a computational complexity of $O(k^2n^2)$.