Lecture notes on alignment—II

Usman Roshan
Department of Computer Science
New Jersey Institute of Technology

March 22, 2005

1 Multiple sequence alignment

The multiple alignment of sequences is considered by some to be the *holy grail* problem of computational biology and bioinformatics. Multiple sequence alignments (MSAs) have many applications ranging from constructing phylogenies to determining functionally related motifs in proteins. The setup of the problem is the same as that of pairwise alignment except that we are now dealing with many sequences as opposed to just two.

1.1 Problem definition

A multiple alignment of n strings $S_1, S_2, ..., S_n$, is a mapping $S_1 \rightarrow S_1', S_2 \rightarrow S_2', ..., S_n \rightarrow S_n'$ such that

- $|S_1'| = |S_2'| = ... = |S_n'|$, and
- removal of gaps from S_i' gives S_i for all $i = 1, .., n$.

An example aligmnent of $S = AGA$, $T = ACGG$, and $U = AGC$ is

$$\begin{array}{ll} S & A-GA \\ T & ACGG \\ U & A-GC \end{array}$$

1.2 Optimization criteria

There are two popular optimization criteria: sum-of-pairs (SP) and phylogentic tree-alignment (TA).

1.2.1 Sum of Pairs (SP)

Given sequences S_1 and S_2, we have previously defined how to score a pairwise alignment S_1', S_2'. Let $S(S_i', S_j')$ be the score of the pariwise alignment under a given scoring function and gap scoring scheme. Given sequences $S_1, S_2, ..., S_n$, the SP score of a multiple alignment $A = S_1', S_2', ..., S_n'$ is defined as

$$SP(A) = \sum_{i<j} S(S_i', S_j').$$

The sum-of-pairs problem is to find the alignment with the optimal SP score and is NP-hard.

1.2.2 Tree alignment (TA)

Definition 1 *[Tree Alignment] Given an evolutionary tree $T = (V, E)$ for n sequences $(S_1, S_2, \ldots, S_n)$ where T has $m + k$ nodes, a tree alignment of $(S_1, S_2, \ldots, S_m)$ on T is a multiple alignment of $(S_1, S_2, \ldots, S_{m+k})$, where $S_{n+1}, ..., S_m$ are additional sequences assigned to the internal nodes of T. Note that the additional sequences must be over the same alphabet as $S_1, ..., S_n$.*

Give a pairwise alignment score $S(S_1', S_2')$, where S_1', S_2' is a pairwise alignment of S_1 and S_2, a tree alignment score is defined as $TA(T, A) = \sum_{(u,v)\in E(T)} S(T_u, T_v)$ where T_v denotes the sequence assigned to node v of T.

The tree alignment problem is to find the optimal tree alignment for a given tree T. The *generalized tree alignment* problem is to find the tree T (as well as its tree alignment) with the optimal tree alignment score. Both of these problems are NP-hard.

2 Heuristics

Heuristics are widely used in multiple sequence alignment. There are two main approaches, iterative and progressive alignment, which we discuss below. We first describe some basic concepts used in heuristic multiple sequence alignment: profile alignments and alignment paths.

2.1 Basics

2.1.1 Profile alignment—aligning two alignments

Very often we are faced with the problem of aligning a set of two alignments. We solve this problem by computing a *profile* for each alignment and then aligning the two profiles using the dynamic programming approach. We defined a profile previously but do it again here.

Suppose we are given the alignment A and want to compute a profile P. We first show how to compute the profile P_i for the i^{th} A_i of alignment A. Each site A_i is represented by a vector of amino acid frequencies P_i. We denote the frequency of amino acid x in P_i as P_i^x. Then the score between two profile sites P_i and Q_j is

$$\sum_{x,y} P_i^x Q_j^y S(x,y)$$

where x and y range over the 20 amino acids and $S(x,y)$ is the amino acid scoring matrix. We can now align two alignments by computing their profiles then use the optimal dynamic programming algorithm to align them, since we have now defined how to score two sites of a profile.

2.1.2 Alignment paths—retrieving the alignment

An alignment path is a string over the alphabet $\{M, I, D\}$, where M means a match, I means an insertion, and D means a deletion. An alignment path can be constructed during the pairwise alignment algorithm as follows.

- Set $path(A,B) = \phi$.
- Compute the optimal scoring matrix and the traceback matrix.
- When constructing the alignment using the traceback matrix we
  - concatenate M to $path(A,B)$ if there is a match between them,
  - concatenate D to $path(A,B)$ if a letter of A matches with a gap,
  - and concatenate I to $path(A,B)$ if a letter of B matches with a gap

Given two sequences A and B their alignment path can be used to retrieve their pairwise alignment A', B'. Assume $|A| = m$, $|B| = n$, and $P(A, B) = p$, where $P(A, B)$ is the profile of their alignment. We can construct A' and B' from the alignment path $path(P(A, B))$ using the following algorithm.

- Set A' and B' to be the empty strings, and set $j = 0, k = 0$.
- For $i = 1...p$ do {

  If $path_i(A, B) == M$ $\{concatenate(A', A_j); concatenate(B', B_k); j++; k++;\}$
  Else if $path_i(A, B) == D$ $\{concatenate(A', A_j); concatenate(B', '-'); j++;\}$
  Else if $path_i(A, B) == I$ $\{concatenate(A', '-'); concatenate(B', B_k); k++;\}$

  }

2.2 Progressive alignment

Progressive alignment is a heuristic for the tree alignment problem. We assume as input a phylogenetic tree T leaf-labeled with sequences from S which are to be aligned. We outline the steps below.

1. Compute a distance matrix of optimal pairwise alignment scores using the dynamic programming algorithm.
2. Compute a rooted phylogenetic guide-tree T using the using the UPGMA algorithm (NJ could also be used).
3. We now assign profiles and alignment paths to each node of T by performing a post-order traversal. For a given node v with children u and w, let $P(v) = profile(alignment(P(u), P(w)))$ be the alignment profile of $P(u)$ and $P(w)$ and let $path(v) = path(alignment(P(u), P(w)))$ be the corresponding alignment path. We use the following recursion to compute $P(root)$ and $path(root)$, the profile and alignment path at the root. Let S_v be the input sequence associated with leaf v.

   If node v is a leaf $\{P(v) = S_v; path(v) = \phi;\}$
   else {let u and w be the children of v;
   let p_{uw} be the pairwise alignment of u and w;
   set $P(v) = profile(p_{uw}); path(v) = path(p_{uw});\}$

   Since we are performing a post-order traversal, every child will have a valid path and profile assigned to it.

4. We now perform a traversal to compute the alignment for each sequence. For each leaf v we perform the following traversal towards the root.

If $parent(v) = \phi$	{ output v'; }
else	$\{p(v) = parent(v)$;
	set v' to the alignment obtained using the path $path(p(v))$;
	set $v = p(v)$; }

ClustalW is a popular alignment software which uses this approach for alignment. It, however, uses the neighbor-joining tree instead of the UPGMA one as the guide-tree. One drawback of this approach is that once a gap is inserted in a profile it cannot be corrected later on.

2.3 Iterative alignment

Another approach for alignment independent of a guide-tree is called iterative alignment. In the iterative approach an alignment is built by aligning pairs of profiles until all the sequences have been aligned to the profile. A simple iterative alignment strategy would be:

- Let S to be the set of sequences to be aligned
- Randomly select a sequence $s \in S$ and set its profile to be P
- While $S \neq \phi$ {
  - Let $s' \in S$ be such that it has the highest scoring alignment with P
  - Align s' to P and update P to be the profile of the new alignment containing s'
  - $S = S - \{s'\}$

A drawback of this approach is that it doesn't take a phylogenetic tree into consideration and it can be very slow.

2.4 Other approaches

We look at two different approaches.

2.4.1 Tree improvement

The main idea in tree improvement is to get a better phylogenetic tree which in turn can yield a better progressive alignment. The tree improvement heuristic is outline below.

- Let S be the sequences to be aligned, and let T be a guide-tree
- Perform a tree alignment A of S on T
- While *not done* do {
  - Compute a tree T' on A
  - Perform a tree alignment A of S on T'

  }

The accuracy and speed of this approach depends upon the method used for the tree alignment and the method used to compute a tree on the alignment. One may iterate this blindly until a specified number of iterations have been met or stop when the tree alignment score or the sum-of-pairs score doesn't improve.

2.4.2 Hybrid—MUSCLE

A recent approach which has performed well on structurally aligned benchmark alignments is MUSCLE—multiple sequence alignment using log expectation score. MUSCLE is a combination of progressive and iterative profile alignment strategies. It has three main stages.

1. **Stage I (draft progressive)** In Stage I an approximate distance matrix is first estimated using *k-mer* counting between pairs of sequences. MUSCLE then computes a UPGMA tree and a progressive alignment on it.
2. **Stage II (improved progressive)** In the second stage a Kimura distance matrix is estimated from the previous alignment and a new UPGMA tree is constructed. The new tree is then used to construct a new progressive alignment and this process is iterated until there is no topological change in the tree structure or a specified number of iterations have elapsed.

3. **Stage III (refinement)** The third stage is called *refinement* and is performed in the following manner. Let T be the tree from Stage II. A given edge e of T is deleted from T which yields two subtrees T_1 and T_2. The profiles of the alignment of the sequences restricted to the two subtrees T_1 and T_2 is computed and sites which contain only gaps are removed. The two profiles are aligned and if the sum-of-pairs score of the new alignment is better than the previous one, the new alignment is kept. Edges are visited in order of decreasing distance from the root. This process iterates until a specified number of iterations have been met or the sum-of-pairs score doesn't improve after the full tree has been traversed.

Log expectation score One of the main difference between MUSCLE and previous approaches is that MUSCLE uses a new profile vector scoring function called the log-expectation score. This is defined as

$$LE = (1 - P_i^{gap})(1 - P_j^{gap}) \log(\textstyle\sum_x \sum_y P_i^x P_j^y \frac{p_{xy}}{p_x p_y}).$$

where P_i^x is the normalized frequency of amino acid x (or gap if x = gap) in profile site P_i. In the summation x and y loop over amino acids and not gaps. The multiplicative factor containing gap frequencies downweights profile sites with a high frequence of gaps.

3 Evaluation of alignment programs

3.1 Simulation—ROSE

There is no well-accepted biological model for the evolution of biomolecular sequences with insertions or deletions. The only amino-acid subsitution model with insertions and deletions that we are aware of is the one in the ROSE software package. We call this the ROSE model. ROSE allows the user to specify many parameters of the evolutionary process. We first overview how sequences are evolved in ROSE.

Given a rooted tree with branch lengths, ROSE first generates a root sequence v. For each child c of v, this sequence is first mutated to yield subsitutions, and then insertions and deletions are performed. These two steps are repeated in a pre-order traversal. Throughout the process ROSE keeps track of the set of insertions and deletions performed, i.e., the true

alignment. We now describe how the root sequence is generated and how the mutations, insertions, and deletions are performed.

Root sequence Given a model tree with edge lengths, a root sequence of specified length k is generated. Each of the k sites is independently filled with an amino acid a with probability f_a. The default probabilities are given by the normalized frequencies of amino acids computed by Margaret Dayhoff (Atlas of Protein Sequence and Structure, 1979).

Mutation We describe how amino acid x mutates to y on a given edge of length b. The probability of change of x to y is given by a probability mutation matrix which can be specified by the user. The default matrix is the 1 PAM matrix (as computed by Dayhoff '79) which is the probability of one accepted subsitution per hundred sites. We call this the 1 PAM* matrix M and describe how mutation occurs per unit branch length. For one unit of branch length, amino acid x changes to y with probabiliy $M(x, y)$, and we say that the edge has an evolutionary rate of 1 PAM*. For b units of branch length this process is repeated b times; therefore the edge would have a rate of b PAM* units.

Insertions and deletions We only describe how insertions take place since deletions occur in the exact same way. Associated with an insertion are two parameters p_{ins} and l_{ins}. l_{ins} is a probability vector of length q_{ins}. A site is selected for insertion with probability $\frac{p_{ins}}{k}$ where k is the length of the sequence (before the insertion). If an insertion is to take place then the length i of the new sequence to be inserted is selected with probability l^i_{ins}, where i is the i^{th} entry of l_{ins} and $0 \leq i < q_{ins}$. Deletions occur in the exact same way using the analogous parameters p_{del}, l_{del}, and q_{del}. The default values of these parameters are $p_{ins} = p_{del}, q_{ins} = q_{del}, l_{ins} = l_{del}, p_{ins} = 0.00005, l_{ins} = \{.1, .1, .1, .1, .1, .1, .05, .05, .05, .05, .05, .05, .05, .05\}$, and $q_{ins} = 14$.