CS262 Winter 2016 Lecture 16 Notes

Professor: Serafim Batzoglou
Scribe: John Luttig
February 25, 2016

One-sentence version: We learned about the applications of and algorithms behind multiple sequence alignment.

Multiple Sequence Alignment: Definition

In multiple sequence alignment, we want to insert gaps in between letters of N sequences $(x_1, x_2, ..., x_{N-1}, x_N)$ such that all sequences have length L and we reveal all overlaps between multiple sequences.

Note that the our definition of multiple sequence alignment is similar to pairwise alignment. However, multiple sequence alignment is much harder than normal alignment, which we explore below.

Key Concepts

Multiple sequence alignment is a large and active field of research within computational genomics. Its main challenges stem from the fact that even using techniques like dynamic programming with relatively low runtimes, trying to align each sequence with every other sequence will necessarily give us an exponential runtime with respect to the number of sequences we are aligning. Thus, we leverage existing knowledge of evolutionary trees to perform pairwise alignments without redundancy, and use a number of heuristics to prevent error accumulation in our alignments.

Applications

Performing multiple sequence alignment across species tells us which features are most preserved across species, giving us a heuristic for which subsequences are most functional across species. The types of amino acids observed in multiple sequence alignments tell us which regions give us which proteins.

Gene structure

In the beginning of genomics as a field when there were very few sequenced genomes, the main reason for sequencing species related to humans was so that we could perform multiple sequence alignment with human genes.

Today, we have sequenced many human genes, but the question remains: how many ways exist of skipping exons and extending introns?

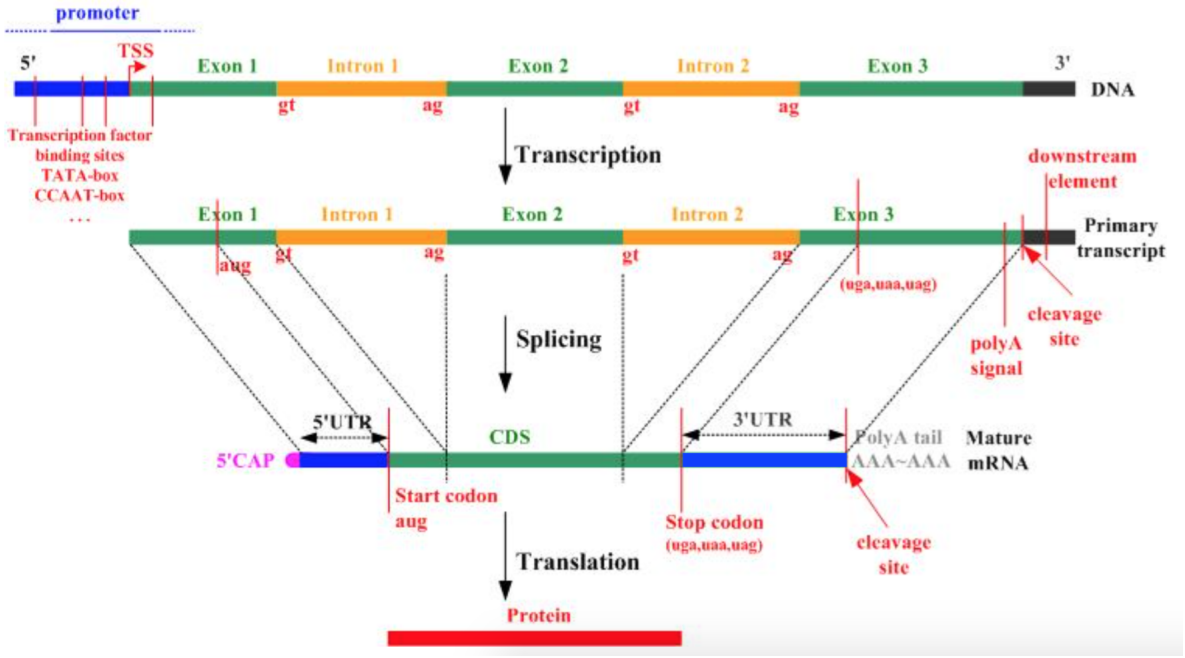

Figure 1: Gene Structure

This is a process during which a DNA molecule is transcribed, spliced, and translated into a protein, while removing introns and keeping exons. The molecule is considered mature once all introns are removed.

The gene is subdivided into regions according to the role of the region during transcription splicing / formation of protein; Transcription starts at the TSS site at the top left of the diagram. The promotor region promotes binding of RNA polymerase, starting the transcription process. The ribosome starts from 5 cap until it gets to AUG, then starts translation, in which every third codon becomes an amino acid. This process stops when it reaches the stop codon sequence UGA, UAG, or UAA.

At the splice site, there is a signal of length 8-12 bits that denotes the start of an intron (e.g. $GGAGTAG$); similarly, we know that the end of intron has lots of CTs and ends in AG.

The important information is stored in the exons in the form of codons, where every 3 letters becomes an amino acid. Differing abundances of tRNA means that there are different frequencies of amino acids in a given organism. We can encode these probabilities in a probabilistic model, which constitutes a large and active area of study in genomics.

GENSCAN was the first large HMM for gene finding, used to predict gene locations and

exon-intron boundaries. It has exon states (initial, middle, and terminal) and intron states. Each state had a frame of where the exon ended, and the HMM had 27 total states. The transition probabilities were affected by (see voice).

We can use multiple alignments to improve gene finding. A gene finder using only one sequence is called *ab initio*. This is a good first step in finding genes specific to this organism, and lost in comparable organisms.

We use the term *de novo* to refer to the process of looking at target genome and aligned informant genome, then taking a probabilistic approach to model relative tendency of exons to be preserved across species.

The RNA-sequence based approach is becoming increasingly popular, it extracts mature RNA from a collection of cells, sequences it, and aligns it to the target genome, to find all different splices found in collection of cells.

We know the function of most of the 20,000 genes, which yield over 1 million proteins.

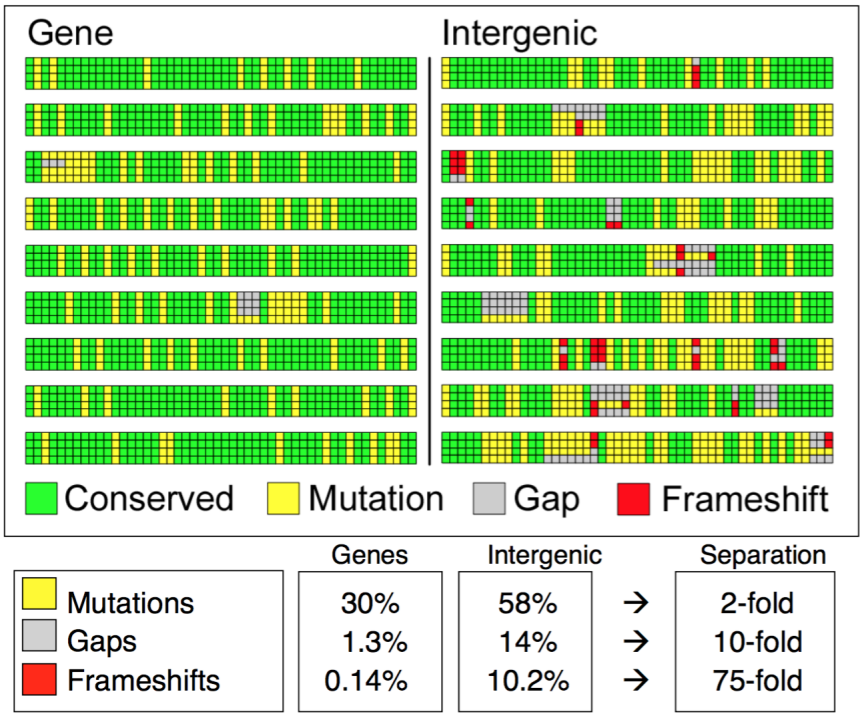

Figure 2: Patterns of Conservation

With excess funds from the NIH, researchers sequences 4 yeasts and wrote a paper on motif finding. Note in the figure above:

1. Lots of frameshifts in Intergenic vs Gene

2. Longer mutations in intergenic

3. Mutations usually spaced apart by 3 in Gene

The pattern of conservation across multiple species can tell you about the function of a given region, and how strong the evolutionary pressure is to conserve a given region.

Scoring Multiple Sequence Alignment

We can perform multiple sequence alignment by aligning them sequentially, pairwise. However, every time you do a pairwise alignment, some positions are wrong. As you pair more sequences, errors add up and it becomes very hard to handle these error regions.

We could build a probabilistic model using an existing evolutionary tree with molecular evolution approach, then build an alignment that maximizes the likelihood of sequences coming from evolution or finding gene sites. However, this is currently intractable given its time complexity.

Sum of Pairs
In practice, we simply use a pairwise scoring function, and sum up the scores. One problem with this approach is that if the evolutionary tree is dense in some subtrees and sparse in others (sum of pairs slide), a.k.a tree not evenly distributed by evolutionary history, you may want to weight the tree.

We defined **induced pairwise alignment** as a pairwise alignment induced by the multiple alignment. For example, consider the following alignment:

```
x:  AC-GCGG-C
y:  AC-GC-GAG
z:  GCCGC-GAG
```

Given this alignment, the induced pairwise alignment is as follows:

```
x:  ACGCGG-C; x:  AC-GCGG-C
y:  AC-GCGAG; y:  ACGC-GAC
z:  GCCGC-GAG; z:  GCCGCGAG
```

We derive the induced pairwise alignment of two sequences by removing gaps and gluing the sequences together.

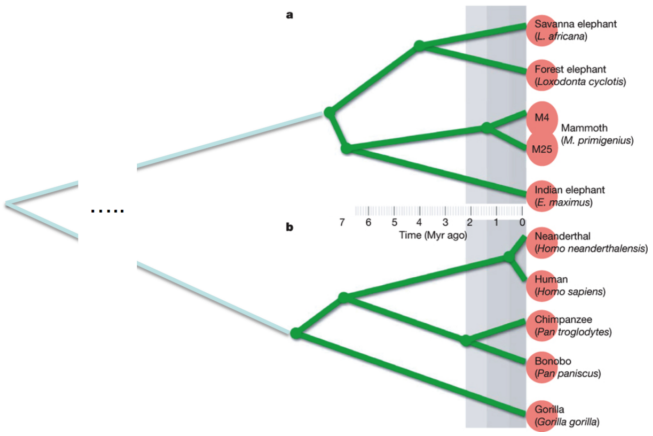

Figure 3: Evolutionary Tree.

For scenarios in which some species are densely populated on a branch and others are sparsely populated, we should weight the pairs of sequences. For example, in the diagram above, we do not want redundant information caused by the densely populated elephants branch relative to the human branch, so we can weight humans higher than elephants.

We can thus define the weighted Sum of Pairs score as follows to account for weightings between all pairs:

$$S(m) = \Sigma_{k<l} w_{kl} s(m^k, m^l)$$

Multiple Sequence Alignment Algorithms

We could convert multiple sequence alignment, losing some information, to a sequence but in difference space, by converting the letters $\Sigma = \{A, C, G, T\}$ into a new alphabet:

$$\Sigma = (p_1, p_2, ..., p_k, p_{GAP} \mid 0 \leq p_i \leq 1, \Sigma\ p_i = 1)$$

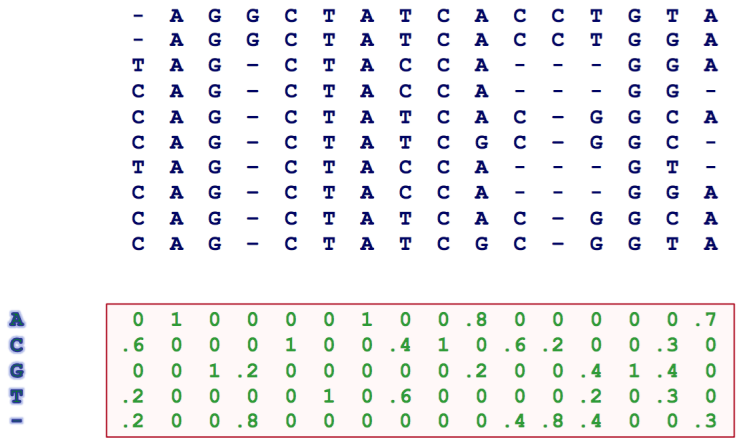

Figure 4: Profile Representation

Profile Representation
We create a profile representation of a multiple alignment and compute the probabilities of each letter at a given position, shown in the red box above. What do we lose here?

1. Can't see conservation across groups of species
2. Lose lengths of gaps
   (a) if you take two multiple alignments and want to align them using an affine gap scoring function, when you decide which positions to gap, you make it such that certain gaps are not penalized.

1 Multidimensional Dynamic Programming

We can generalize the Needleman-Wunsch algorithm as follows. Note that this is effectively the sum of the column scores:

$$S(m) = \Sigma_i S(m_i)$$

$F(i_1, i_2, ..., i_N)$ represents the optimal alignment up to $(i_1, ..., i_N)$. This tells us the optimal score of aligning the first i_1 letters of sequence 1, i_2 letters of sequence 2, and so on.

$$F(i_1, i_2, ..., i_N) = max_{\text{(all neighbors of cube)}} * (F(nbr) + S(nbr))$$

In 2-dimensional dynamic programming, there are 3 previous neighbors. in 3-dimensional dynamic programming, however, there are 7 previous neighbors (generalizable to $2^k - 1$ for k-dimensional):

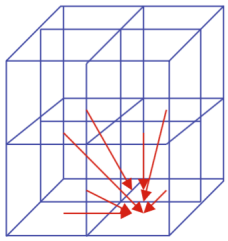

Figure 5: Pointers to current cell in 3-dimensional dynamic programming.

$$F(i,j,k) = max \begin{cases} F(i-1,j-1,k-1) + S(x_i, x_j, x_k) \\ F(i-1,j-1,k) + S(x_i, x_j, -) \\ F(i-1,j,k-1) + S(x_i, -, x_k) \\ F(i-1,j,k) + S(x_i, -, -) \\ F(i,j-1,k-1) + S(-, x_j, x_k) \\ F(i,j-1,k) + S(-, x_j, -) \\ F(i,j,k-1) + S(-, -, x_k)s \end{cases}$$

The problem with multidimensional DP is that it is completely unscalable for 3 or more species.

Runtime

The size of the dynamic programming matrix is L^N, where:

L = length of each sequence
N = number of sequences

There are $2^N - 1$ neighbors per cell. This gives us a runtime of $O(2^N L^N)$, which is infeasible for $N > 3$.

2 Progressive Alignment

This is the main method used for multiple sequence alignment. It assumes knowledge of the evolutionary tree. We can align the closest profiles or sequences first, in the order that the tree denotes. Note that at each alignment, we are aligning two sequences x,y or profiles p_x, p_y, to generate a new alignment with a profile p_{result}. We can also use a weighted version of the tree, in which the weight of the edge is proportional to evolutionary divergence in that

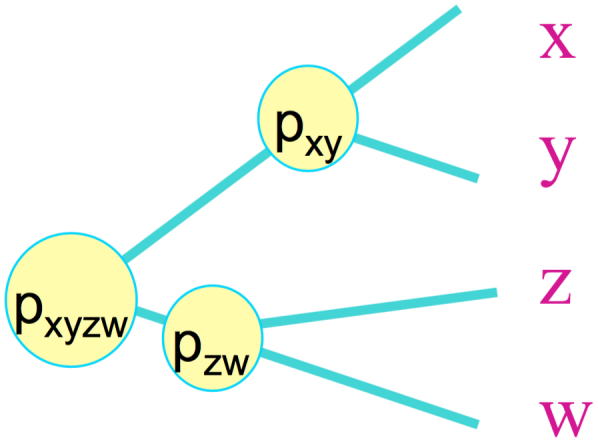

Figure 6: Progressive Alignment Tree

edge, and the new profile is a weighted average of the two joined profiles.

A **profile** is when you take pairwise multiple alignment and convert it into a vector of probabilities (from earlier). The following profiles are defined as a vector of probabilities; the probabilities of A, C, G, T, and $-$, respectively:

$p_x = (0.8, 0.2, 0, 0, 0)$
$p_y = (0.6, 0, 0, 0, 0.4)$

For example, we could get the above profiles if p_x referred to the sequence $AAAAAAAACC$ and p_y referred to the sequence $AAAAAA----$.

We then define the substitution score between the profiles p_x and p_y. We want this to obey the Sum of Pairs score. Note that we don't care about pairs of sequences within p_x or within p_y; when aligning two profiles, we only care about optimizing sequences that cross both profiles. Thus we consider the letters that occur in both profiles:

$$s(p_x, p_y) = 0.8 * 0.8 * s(A, A) + 0.2 * 0.6 * s(C, A) + 0.8 * 0.4 * s(A, -) + 0.2 * 0.4 * s(C, -)$$

This gives us the result $p_{xy} = (0.7, 0.1, 0, 0, 0.2)$.Now, if we define our scoring function as follows:

$$s(p_x, -) = 0.8 * 1.0 * s(A, -) + 0.2 * 1.0 * s(C, -)$$

Then, our result will be $p_{x-} = (0.4, 0.1, 0, 0, 0.5)$. Note that if the profiles differ in length, we can use a weighted version of the above method and adjust the weights accordingly.

Unknown Evolutionary Tree

We can still perform all pairwise alignments as before. Then, we can define a distance matrix D, where $D(x, y)$ is a measure of evolutionary distance, based on the pairwise alignments we calculated. Then, we can construct the tree using a method like UPGMA or Neighbor Joining. Then, we can proceed as before by aligning based on the tree we have created.

Error-prevention Heuristics

The problem with progressive alignment is that every mistake you make accumulates. Thus, by the time you have joined 100+ sequences, they have compounded mistakes and are less clear, making it harder to align two profiles. We have several heuristics to protect against this type of behavior:

1. Iterative refinement (most basic and widely used)
2. A*-based search
3. Consistency
4. Simulated Annealing

Iterative Refinement

Iterative refinement addresses the problem in progressive alignment that the initial alignments are "frozen", despite new evidence coming in that should change the original alignments. For example, consider the following sequences:

```
x:  GAAGTT
y:  GAC-TT
z:  GAACTG
w:  GTACTG
```

In this example, x and y are frozen, but if you observe z and w, it is clear that the gap in y should be placed such that $y = GA - CTT$.

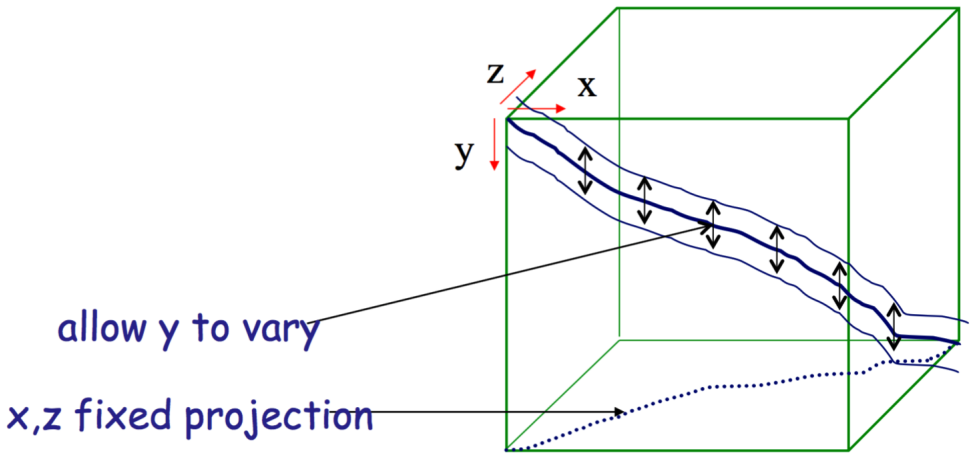

Figure 7: Barton-Stenberg Algorithm.

Barton-Stenberg Algorithm

The Barton-Stenberg show above can be defined as follows:

1. For $j = 1$ to N:
   (a) Remove x^j
   (b) Realign x^j to a profile of the other aligned sequences $(x^1 \dots x^{j-1} x^{j+1} \dots x^N)$.
2. Repeat until the alignment score converges.

Iterative Alignment - Example

In this example, we will align (x, y), (z, w), (xy, zw), where each sequence is as follows:

```
x:  GAAGTTA
y:  GAC-TTA
z:  GAACTGA
w:  GTACTGA
```

Realigning y yields the following:

```
x:  GAAGTTA
y:  G-ACTTA
z:  GAACTGA
w:  GTACTGA
```

This is because y is better aligned with both z and w (look at the AC subsequence).

Consistency

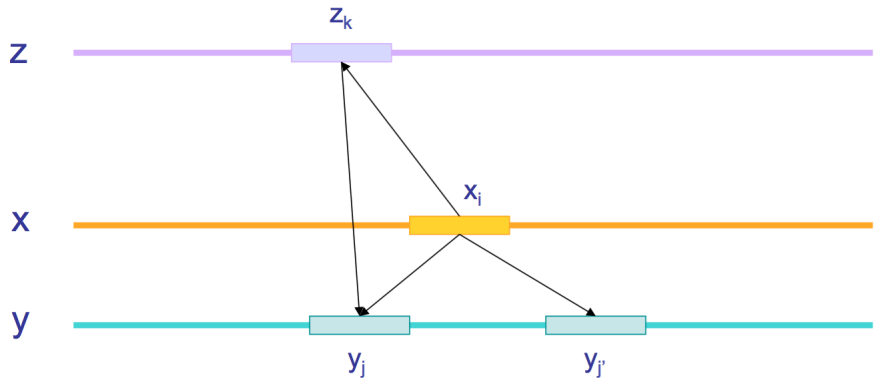

Figure 8: Scenario in which consistency is beneficial.

On a high level, consistency helps us avoid making bad choices when we are unsure of how to pairwise align two sequence by using the information we have from other paired alignments. We can see the benefits of consistency using the above figure, described in the following scenario. Imagine a situation where there is ambiguity whether x_i should align to y_j and $y_{j'}$. If there is a third sequence z that x and y must align to, and z_k aligns to x_i and really likes y_j, x_i will now want to align to y_j.

Algorithm

1. Compute all pair of alignments. In the above scenario, this would be xy, xz, and yz.
2. When aligning xy during progressive alignment:
   (a) For each (x_i, y_j), let $s(x_i, y_j)$ be a function of x_i, y_j, a_{xz}, a_{yz}.
   (b) Now, align x and y using dynamic programming with the modified s function.

MUSCLE

MUSCLE stands for **MU**ltiple **S**equence **C**omparison by **L**og-**E**xpectation. It is an algorithm that has high throughput and accuracy; it can align hundreds of sequences in seconds.

We define the MUSCLE algorithm as follows:

1. $D_{DRAFT}(x, y)$ is defined in terms of number of common k-mers (k is about 3). This can be performed in $O(N^2 L log L)$ time.
2. Now, build a tree T_{DRAFT} based on those distances, using the UPGMA algorithm.

3. Perform progressive alignment over T_{DRAFT}, which gives us the multiple alignment M_{DRAFT}.
4. Measure the new Kimura-based distances $D(x, y)$ using M_{DRAFT}.
5. Build tree T using D.
6. Perform progressive alignment on T, which gives us the multiple alignment M.
7. User iterative refinement (defined above) several times
   (a) Tree Partitioning: Split M along one branch, then realign the two resulting profiles.
   (b) If new alignment M? has a better sum-of-pairs score (defined above) than M, accept M'

Note: we did not get to the ProbCons part of the lecture slides during lecture.

MAFFT

Paper: https://mbe.oxfordjournals.org/content/30/4/772.full

The MAFFT program was not discussed in lecture, but is an alternative to the MUSCLE and PROBCONS algorithms. It is closely related to MUSCLE in that it performs multiple sequence alignment with high throughput and accuracy. It has higher accuracy than MUSCLE, but lower accuracy than PROBCONS. Conversely, it is much faster than PROBCONS, but slower than MUSCLE. It can also perform local alignment, which is a strategic advantage over other programs.

One version of the MAFFT algorithm is shown below, although the iterative refinement step is optional:

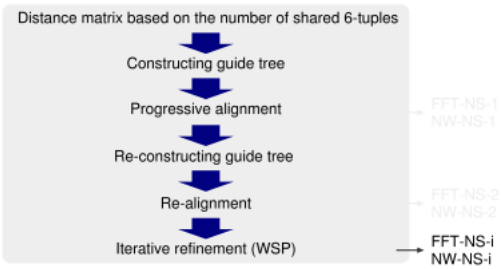

Figure 9: MAFFT with iterative refinement.