

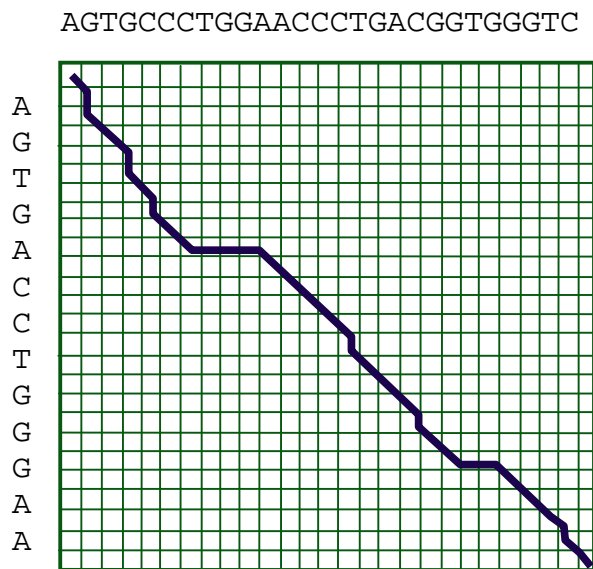
Lecture 17: Multiple Sequence Alignment

This lecture will review the problem of Multiple Sequence Alignment and some techniques to approach the problem. It is one of the fundamental problems of computational genomics and is defined as follows:

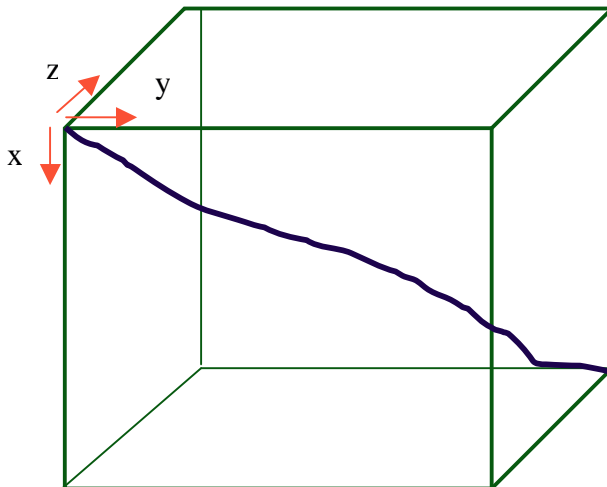
Given N sequences $x^1, x^2 \dots x^N$, insert gaps (-) in each sequence x^i such that

- All sequences have the same length L
- Score of the global map is maximum

With two sequences, the optimal solution can be found using Needleman-Wunsh (NW) algorithm, which is effectively finding the best path (with respect to the scoring function) from top left corner to bottom right corner as depicted in the figure below.



With three sequences it maps to the best path in a cube.



In general, for N sequences the search space is a N-dimensional hypercube with a path such that each co-ordinate either increases or stays the same. The Needleman-Wunsh algorithm generalizes trivially to higher dimensions but is very slow (details follow later). The alignment so produced can be projected to a subset of dimensions giving an alignment of fewer sequences. All possible projections to two dimensions comprise the “induced pairwise alignments”.

Given a pair of sequences from among a set of multiply aligned sequences, “induced pairwise alignment” is the pairwise alignment obtained by retaining only the pair from multiple alignment and removing the positions with a gap-gap alignment.

Example:

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

Induces:

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

A scoring function for multiple alignment is Sum of Pairs (SOP) function. It is defined as the sum of scores of all induced pairwise alignments. If we are given an evolution tree, we can incorporate it by introducing weights in the scoring function.

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

w_{kl} : weight decreasing with distance

This can help reduce bias in the multiple alignment if there are many similar species. Now, we define the concept of a profile used in multiple alignment. Consider the following example:

	-	A	G	G	C	T	A	T	C	A	C	C
T	A	G	-	C	T	A	C	C	A	-	-	
C	A	G	-	C	T	A	C	C	A	-	-	
C	A	G	-	C	T	A	T	C	A	C	-	
C	A	G	-	C	T	A	T	C	G	C	-	
A		1				1			.8			
C	.6			1		.4	1		.6	.2		
G			1	.2					.2			
T	.2				1	.6						
O	.2		.8						.4	.4		
E										.4		
C	.2		.8							.4		

Given a multiple alignment $M = m_1 \dots m_n$, replace each column m_j with profile entry p_j :

- Frequency of each letter in S
- Number of gaps
- Optional: number of various kinds of gaps (openings, extensions, closings)

Multidimensional Dynamic Programming

As mentioned before Needleman-Wunsh can be generalized to N dimensions ($N > 2$) with the scoring function defined as the sum of column scores.

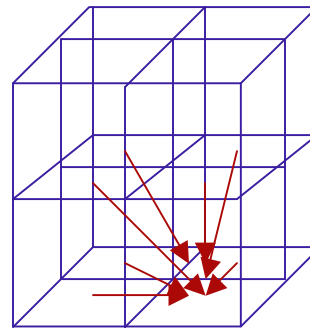
$S(m) = \sum_i S(m_i) = \text{sum of column scores}$

$F(i_1, i_2, \dots, i_N)$: Optimal alignment up to (i_1, \dots, i_N)

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

For example, with a constant gap penalty model on three sequences the search space is a cube with 7 neighbours per cell and the recursion is given below considering all the various possibilities of alignment in last column.

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



Lets analyze the running time of the algorithm. If there are N sequences of length L each, the size of the hypercube is L^N . For the last column of alignment, we have 2 choices for each sequence, whether its last character forms a part of the last column or is already aligned before giving 2^N possibilities for its neighbors. However since we do not allow gap-only columns, the exact number of neighbors per cell is $2^N - 1$. The running time therefore is $O(L^N 2^N)$.

If we try to generalize to affine gaps we would need additional matrices as with 2 sequences. So, before we move from one cell to another in the hypercube we would additionally need the type of cell we are coming from. This could be any of the 2^N possibilities required for exact computation of score under affine gap model. Essentially, each cell has proliferated in 2^N different types of cells and optimal score would be the best among all possibilities. This increases the running time by a factor of 2^N to $O(L^N 4^N)$.

Progressive Alignment

When the evolutionary tree is known for the given sequences, the closest sequences are aligned first, in the order of the tree. In each step, align two sequences x, y , or profiles p_x, p_y , to generate a new alignment with associated profile p_{result} . If the tree edges have weights (proportional to the divergence in that edge), new profile is a weighted average of two old profiles. The alignment of two profiles shouldn't make any changes within the profiles. Only the following possibilities are considered, alignment of p_x against a gap in

the other and vice-versa, or alignment of p_x against p_y . The computation of scores for these possibilities is illustrated in the following example.

Profile: (A, C, G, T, -)

$p_x = (0.8, 0.2, 0, 0, 0)$

$p_y = (0.6, 0, 0, 0, 0.4)$

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

Result: $p_{xy} = (0.7, 0.1, 0, 0, 0.2)$

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

Result: $p_{x-} = (0.4, 0.1, 0, 0, 0.5)$

When the evolutionary tree is not known, we first perform all pairwise alignments, and define distance matrix D , where $D(x, y)$ is a measure of evolutionary distance based on pairwise alignment. Construct a tree using one of the several techniques discussed later in the course and align on the tree.

Aligning two alignments

Given two alignments m_1 and m_2 we want to find the optimal alignment under SOP scoring with affine gaps. Consider the following example:

```

m1      x GGGCACTGCATGTAGTCAGTCG
         y GGTTACGTC-----GTCACGTG

m2      v GGACCT-----CGCCAGGGA
         w GGACGTACC-----CGCCAGGGG
         z GGGAACTGCAGGTCGTCAGTCG
  
```

The induced pairwise alignment can make certain gaps disappear. If the gap in both sequences start at same position and end at same position, there will be none in the pairwise alignment. If both start and end position mismatch then there will be one gap in both sequences. Similarly for other different cases of starting and ending positions.

If we consider pairwise alignment of y with w , and decide to simply add gap open penalty everytime a gap is aligned to a character we correctly account for the resulting one gap in their alignment. Similarly y with v has two gap events and it correctly accounts in this case too. However, consider the following case,

```

x GGGCACTGCA--AGTCAGTCG
y GGTTACGTC--GTCACGTG
  
```

Here we incorrectly account twice for the boxed positions. This illustrates that the order of gap openings is required to decide the correct score contribution at each step. But since there are exponentially many such orderings, this seems non-trivial. In practice, one of the two assumptions are made during score calculation. “Optimistic” assumes no gap i.e.

don't pay gap open penalty. "Pessimistic" assumes gaps i.e. pay gap-open penalty. In general, this may produce alignments which are very different from optimal alignment. Further, this problem has shown to be NP-Hard in general.

Heuristics to improve multiple alignments

- Iterative Refinement
- A* based search
- Consistency
- Simulated Annealing
- ...

We will discuss the first three here.

Iterative Refinement

A problem with progressive alignment is that initial alignments are "frozen" even when new evidence comes in. For example,

```

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

```

} Frozen!

When z and w are added after alignment of x and y, we can clearly see that the correct y=GA-CTT, but progressive alignment doesn't allow any change to the initial alignments. Iterative refinement tries to overcome this drawback. The following algorithm is due to Barton-Stenberg.

1. Align most similar x^i, x^j
2. Align x^k most similar to $(x^i x^j)$
3. Repeat 2 until $(x^1 \dots x^N)$ are aligned

Refinement Step

4. For $j = 1$ to N ,
Remove x^j , and realign to $x^1 \dots x^{j-1} x^{j+1} \dots x^N$
5. Repeat 4 until convergence

This procedure is guaranteed to give equal or better score since the original solution is part of the search space. Further, since at each step we either improve the score or stay the same the iteration is guaranteed to converge. Other variants of the algorithm split the set of sequences into subsets and realign sets of sequences. The following example shows improvement due to refinement step.

Align (x,y), (z,w), (xy, zw):

```

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

```

After realigning y:

```

x:   GAAGTTA
y:   G-ACTTA      + 3 matches
z:   GAACTGA
w:   GTACTGA

```

However, there may be cases when the refinement step fails to correct the alignment. Consider:

```

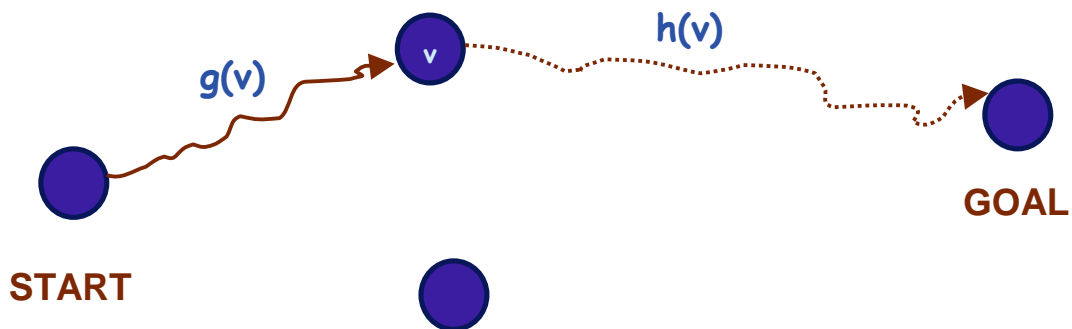
x:   GAAGTTA
y1:  GAC-TTA
y2:  GAC-TTA
y3:  GAC-TTA

z:   GAACTGA
w:   GTACTGA

```

Realigning any single y_i doesn't change anything since other y's keep the 'C' in the same column.

A* for Multiple Alignments



A* is an algorithm to find the shortest path from start node to the goal node in a graph with weighted edges. It is usually employed on graphs with large number of edges since it finds the optimal solution faster in practice compared to Dijkstra that always runs $O(V \log V + E)$ and would be too slow if the number of edges is huge.

In the above figure, $g(v)$ is the cost of the path from START to Node v. $h(v)$ is a (heuristic) estimate of the minimum cost from v to GOAL such that it is an underestimate and hence the minimum cost of a path passing by v, $f(v) \geq g(v) + h(v)$. The algorithm

expands v with the smallest $f(v)$. Also, it never expands v , if $f(v) \geq$ shortest path to the goal found so far.

The multiple alignment problem is mapped to this as follows:

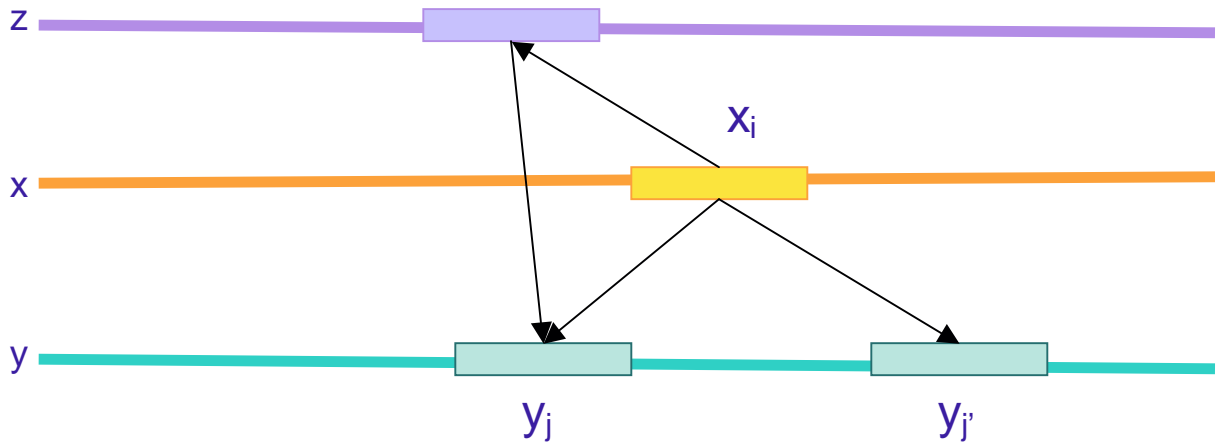
Graph Nodes: Cells in dynamic programming matrix

$g(v)$: alignment cost so far

$h(v)$: sum-of-pairs of individual optimal pairwise alignments

Note, that $h(v)$ satisfies the criterion for being an underestimate of the actual cost, since the optimal pairwise alignment score has to be less than or equal to the score obtained from the induced pairwise alignment. To compute $h(v)$, for each pair of sequences x, y , compute $F^R(x,y)$ the DP matrix of scores of aligning a suffix of x to a suffix of y . Then, at position (i_1, i_2, \dots, i_N) , $h(v)$ becomes the sum of $\binom{N}{2}$ F^R scores.

Consistency



The basic idea of consistency is to incorporate other pairwise alignment preferences during progressive alignment.

Basic method for applying consistency:

- Compute all pairs of alignments xy, xz, yz, \dots
- When aligning x, y during progressive alignment,
 - For each (x_i, y_j) , let $s(x_i, y_j) = \text{function_of}(x_i, y_j, a_{xz}, a_{yz})$
 - Align x and y with DP using the modified $s(.,.)$ function

The substitution function is usually obtained from the product of pairwise substitution matrices with 1/0's where we take logical AND instead of sum to obtain the the product matrix representing common alignment preferences.

Some Resources

Genome Resources

Annotation and alignment genome browser at UCSC

<http://genome.ucsc.edu/cgi-bin/hgGateway> Specialized VISTA alignment browser at LBNL

<http://pipeline.lbl.gov/cgi-bin/gateway2ABC>—Nice Stanford tool for browsing alignments

<http://encode.stanford.edu/~asimenos/ABC/>

Protein Multiple Aligners

<http://www.ebi.ac.uk/clustalw/>

CLUSTALW – most widely used

http://phylogenomics.berkeley.edu/cgi-bin/muscle/input_muscle.py

MUSCLE – most scalable

<http://probcons.stanford.edu/>

PROBCONS – most accurate

Whole genome alignment Rat-Mouse-Human

In next 2 yrs, we can expect to see complete genome sequencing and alignment for several mammals. This would require an increased application of multiple sequence alignment to identify functional regions in the genomes using the known information for existing sequences.

