# Lecture 5: Sequence Similarity

Lecturer: Serafim Batzoglou

04/19/2006

(All images cribbed from the lecture slides)

## Background

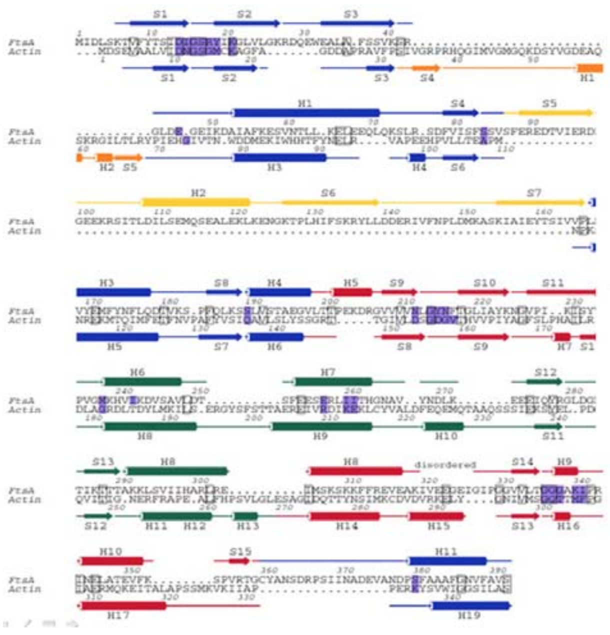
### *The Central Dogma*

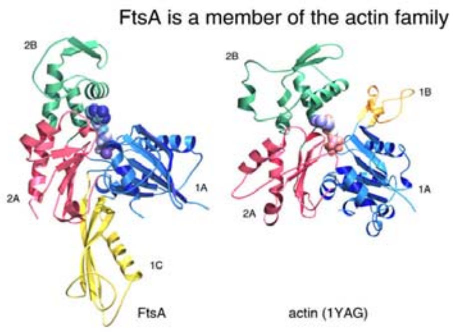
The Central Dogma of Molecular Biology states that DNA is transcribed onto a single stranded messenger-RNA, which in turn gets translated into sequences of amino acid. These strands of amino acid then go on to fold into secondary and tertiary structures, and eventually arrive at its native conformation as a protein. It is in this native conformation that the protein can carry out its function.

More specifically, triplets of nucleotide in a DNA sequence, known as *codons*, are mapped onto one of the 20 amino acid according to the genetic code . Furthermore, it is known that the same sequence of amino acid will almost always fold into the same structure. Thus, sequence similarity in DNA sequences or amino acid sequence is a good indication that the resulting protein will be structurally and functionally similar. Tools that identify sequence similarity are therefore a valuable asset for researchers in the field of biology.

### *Sequence Similarity as a tool*

One major use of sequence similarity is to take a newly discovered nucleotide or polypeptide sequence and compare it against a database of existing sequences with known functionality. Consider for example Actin, a protein found in human muscle cells, and FitsA, a protein found in bacteria. Actin aids in the contraction of muscle cells. Similarly, FitsA help aid contraction during cell division. Comparing the sequence and structure of FitsA and Actin yield high degree of similarity:



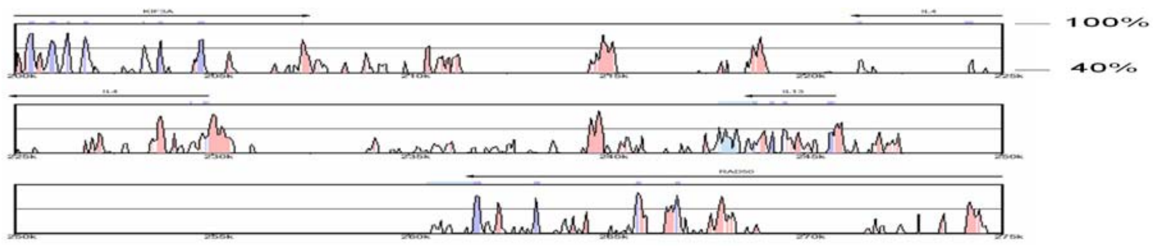


In this way, we can use the information such as Actin and FitsA in a protein or nucleotide database to classify newly discovered sequences.

Another use of sequence similarity is to determine from it phylogeny of proteins and species, as well as bring to attention important regions of a sequence. Proteins evolve by both duplication and species divergence by means of mutations in the DNA which encode them. Before continuing, we will introduce a few definitions:

- *Orthologs* are two proteins in species **X** and species **Y** that evolved from one protein in an ancestor species **Z**.
- *Paralogs* are two proteins that evolved from one protein through duplication.
- *Homologs* are orthologs or paralogs that exhibit sequence similarity.

Evolution, through process of elimination, weeds out the homologs deemed unfit for survival. Thus, important regions of DNA such as areas that encode protein tend to be *conserved*, i.e. these regions tend to stay the same over time. It is therefore not surprising that highly conserved region in a sequence implies some important functionality. Indeed, most protein coding regions of the DNA are over 80% similar between humans and rodents:



With this background, we will continue on to the topic of sequence alignment.

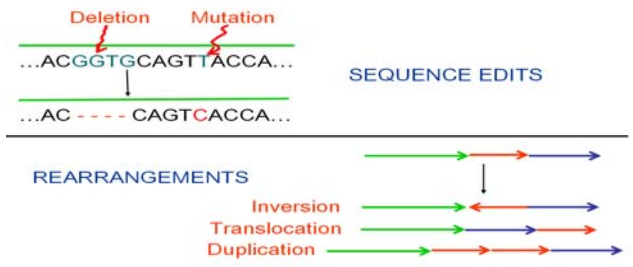
## Sequence Alignment

### *Definition*

Given two strings **X** and **Y**,

$$\mathbf{X} = x_1 \quad x_2 \quad \ldots \quad x_M$$
$$\mathbf{Y} = y_1 \quad y_2 \quad \ldots \quad y_N$$

An *alignment* is an assignment of gaps to positions 0...M in **X** and 0...N in **Y** so as to line up each letter in one sequence with either a letter or a gap in the other sequence. In other words, we want to place gaps in between letters of the two sequences to make the length the same, and so as to reveal their sequence similarity. The resulting alignment can be thought of as a hypothesis that the two sequences come from a common ancestor through *sequence edit* such as *mutations* or *deletions*, or *rearrangements* such as *inversion*, *translocation*, or *duplication*. See the below diagram for an illustration of the various types of edits that can occur.



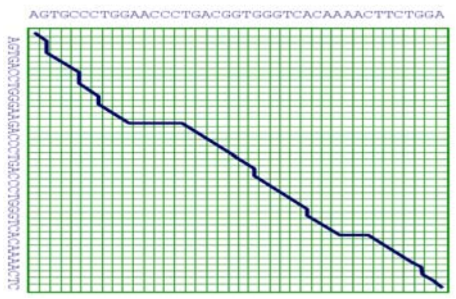
For the purposes of our sequence similarity algorithms, we will only consider sequence edits.

### *Scoring Function*

In order to determine a best alignment, we need to assign a score to the various types of alignment that we can generate from two sequences. Say we assign (**+m**) to matches, (**-s**) for mismatches, and (**-d**) for gaps. We can then define a scoring function for alignment as:

$$\text{Score } F = (\#\text{ of matches})*m - (\#\text{ of mismatches})*s - (\#\text{ of gaps})*d$$

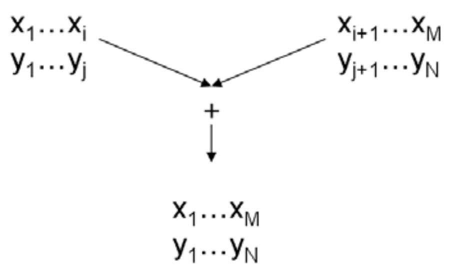
Now that we have a scoring function, we can search the alignment space for an optimal solution. This can be accomplished by creating an M by N matrix and finding a path through that matrix—a diagonal in cell (i, j) represents a match of $x_i$ and $y_j$, a vertical line in cell (i, j) represents aligning $x_i$ to a gap in sequence **Y**, and a horizontal line in cell (i, j) represents aligning $y_j$ to a gap in sequence **X**. An alignment can then be defined as a path through the matrix:



How do we find an optimal path? A naïve approach would be to generate all possible paths in the matrix and take the best. It should be evident that such a solution will only work for very short sequences, because the number of possible alignment grows exponentially with the length of the sequence. Thus, we need a better method to generate an optimal solution.

### *Dynamic Programming*

Fortunately, alignment score is *additive*. That is, the score of aligning $x_1 \ldots x_M$ with $y_1 \ldots y_N$ is equivalent to the sum of the scores of aligning $x_1 \ldots x_i$ to $y_1 \ldots y_j$ and $x_{i+1} \ldots x_M$ to $y_{j+1} \ldots y_N$. Below is a graphical depiction:



This property of alignment score allows us to perform a technique known as *dynamic programming* to find an optimal score. The general idea behind dynamic programming is to find a solution to a larger problem by using solutions to smaller sub-problems as building blocks. Needleman and Wunsch first published a paper on applying dynamic programming to the problem of finding an optimal alignment, and therefore have an algorithm named after them.

## Needleman-Wunsch Algorithm

### *The algorithm*

Needleman-Wunsch algorithm is a dynamic programming algorithm which computes the optimal alignment between two strings. The key to Needleman-Wunsch algorithm is realizing that when constructing an alignment from two sequences, we have one of the three options:

$x_i$ aligns to $y_j$

$x_1 \ldots\ldots x_{i-1} \quad x_i$
$y_1 \ldots\ldots y_{j-1} \quad y_j$

$$F(i,j) = F(i-1, j-1) + \begin{cases} m, \text{ if } x_i = y_j \\ -s, \text{ if not} \end{cases}$$

$x_i$ aligns to a gap

$x_1 \ldots\ldots x_{i-1} \quad x_i$
$y_1 \ldots\ldots y_j \quad -$

$$F(i,j) = F(i-1, j) - d$$

$y_j$ aligns to a gap

$x_1 \ldots\ldots x_i \quad -$
$y_1 \ldots\ldots y_{j-1} \quad y_j$

$$F(i,j) = F(i, j-1) - d$$

Thus, an alignment is defined to be the path from the top-left corner to the bottom-right corner of the Needleman-Wunsch Matrix (the DP matrix). The path is composed of one of three moves:

| | |
|---|---|
| **Diagonal** | if $x_i$ aligns with $y_j$, either via a match or a mismatch. |
| **Right** | if $x_i$ aligns to a gap. |
| **Down** | if $y_j$ aligns to a gap. |

```
1.  Initialization.
    a.  F(0, 0)   = 0
    b.  F(0, j)         = - j × d
    c.  F(i, 0)         = - i × d

2.  Main Iteration. Filling-in partial alignments
    a.  For each        i = 1……M
          For each      j = 1……N
                        F(i, j)   =  max { F(i-1,j-1) + s(x_i, y_j)   [case 1]
                                           F(i-1, j) – d              [case 2]
                                           F(i, j-1) – d              [case 3]

                        Ptr(i,j)  =  { DIAG,   if [case 1]
                                       LEFT,   if [case 2]
                                       UP,     if [case 3]

3.  Termination. F(M, N) is the optimal score, and
    from Ptr(M, N) can trace back optimal alignment
```

After the main iteration, the bottom right corner of the DP matrix will contain the score of the optimal alignment which we can reconstruct by following the backpointers from that cell.

### *Space and Time Complexity Analysis*

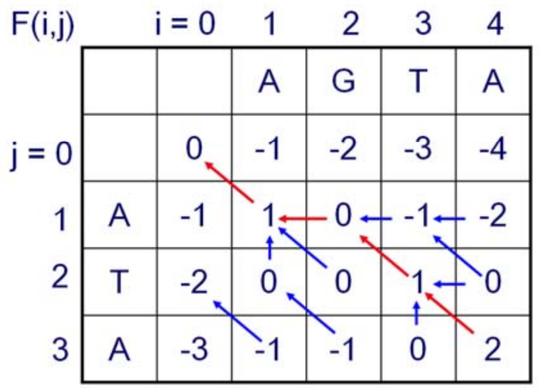
It should be clear that Needleman-Wunsch takes quadratic time O(MN) and uses quadratic space O(MN). History: The original algorithm formulated by Needleman and Wunsch as a cubic time algorithm. The algorithm has been refined by Gotoh to be the quadratic time solution that we know today.

### *An example*

In the following example, we will produce an optimal alignment between the sequence **AGTA** and **ATA**:

x = AGTA
y = ATA

m = 1
s = -1
d = -1



Optimal Alignment:

F(4,3) = 2

AGTA
A - TA

F(1, 1): A matches with A, so we will add m=1 to our score and place a **diagonal** backpointer.

F(2,1): max { F(1,1)-d=0, F(1,0)-s=-2, F(2,0)-d=-3} = 0; place a **left** backpointer.

And so on. Once the F matrix is completely filled, we can trace back the back pointers, to produce the optimal alignment. Note: if there are multiple backpointers from a particular cell, you can just choose one—they will all result in the same score.

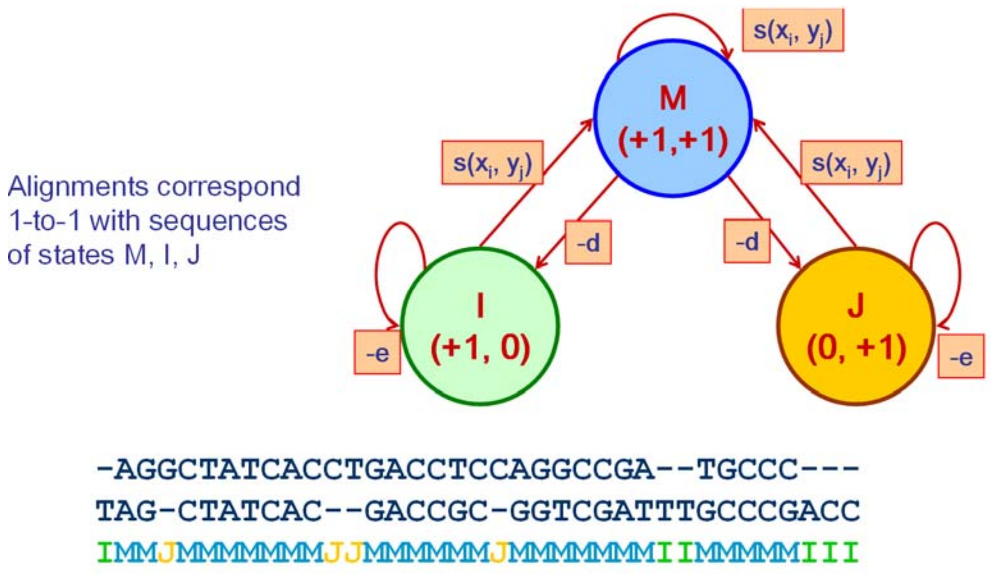
### *BLOSUM Matrices*

We have thus far treated match/mismatch score as some constant, m, s, and d. For a typical DNA alignment algorithm, the value 1 for a match and -1 for a mismatch is used. For a typical protein alignment algorithm, these constants are obtained from a substitution matrix, where the constants in cell (i, j) corresponds to the score aligning letters $x_i$ and $y_j$. The BLOSUM matrix, which is typically used, is computed by giving structurally similar amino acids a higher score than others.

## Pair HMM for alignments

### *State Machine representation*

Another way of representing alignment is as a finite automaton as shown below where **M** represents the match state, **I** represents placing a gap in sequence **X**, and **J** represents placing a gap in sequence **Y**. The score **s($x_i$, $y_j$)** is associated with the transition that brings the automata back to a match state. A penalty **–d** is associated with transitioning out from a match state and a penalty **–e** is associated with continuing a gap. An alignment then corresponds to a sequence of states **M**, **I**, and **J**.



### *Pair HMM*

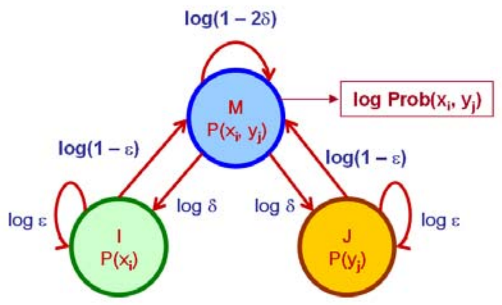
We can create a probabilistic model, known as Pair Hidden Markov Models, for alignments by assigning probabilities to the transition instead of assigning scores. Probabilities of mutations reflect amino acid similarities, and different probabilities can be used for opening and extending a gap. In order to find the likeliest alignment, we can use a dynamic programming method known as the Viterbi algorithm, which is very similar in concept to Needleman-Wunsch.

Compute the following matrices (DP)

- $M(i, j)$: most likely alignment of $x_1 \ldots x_i$ with $y_1 \ldots y_j$ ending in state M
- $I(i, j)$: most likely alignment of $x_1 \ldots x_i$ with $y_1 \ldots y_j$ ending in state I
- $J(i, j)$: most likely alignment of $x_1 \ldots x_i$ with $y_1 \ldots y_j$ ending in state J

$$M(i, j) = \log(\text{Prob}(x_i, y_j)) + \max\{ M(i-1, j-1) + \log(1-2\delta),\ I(i-1, j) + \log(1-\varepsilon),\ J(i, j-1) + \log(1-\varepsilon) \}$$

$$I(i, j) = \max\{ M(i-1, j) + \log \delta,\ I(i-1, j) + \log \varepsilon \}$$

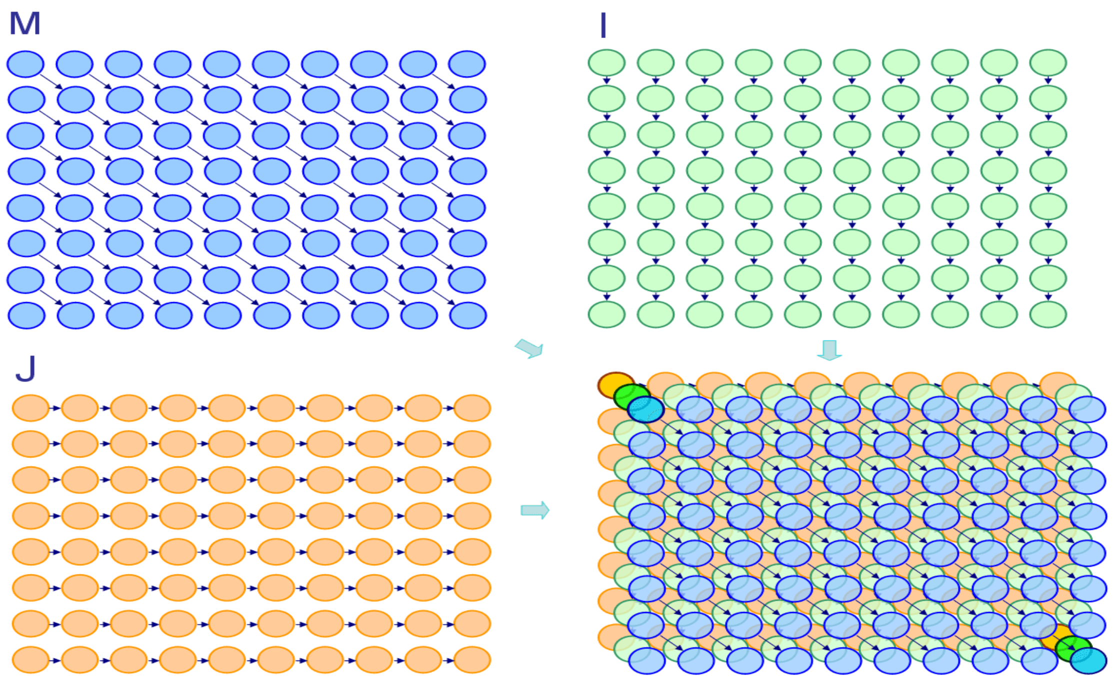


The full Viterbi algorithm for alignment is as follows:

```
For each i = 1, …, M
    For each j = 1, …, N
      $M(i, j) = \log(\text{Prob}(x_i, y_j)) +$
                 max {   $M(i-1, j-1) + \log(1-2\delta)$,
                         $I(i-1, j) + \log(1-\varepsilon)$,
                         $J(i, j-1) + \log(1-\varepsilon)$
                 }
      $I(i, j) =$  max {   $M(i-1, j) + \log \delta$,
                         $I(i-1, j) + \log \varepsilon$
                 }
      $J(i, j) =$  max {   $M(i-1, j) + \log \delta$,
                         $I(i-1, j) + \log \varepsilon$
                 }
```

When matrices are filled, we can trace back from (M, N) the likeliest alignment

Note the use of three matrices, M, I, and J, to keep track of which states we are currently in. A way to visualize the state path is to imagin a lattice composed of the three matrices, M, I, and J. In the lattice, cells in M are connected diagonally, the cells in I are connected vertically, and the cells in J are connected horizontally. Cell I(i, j) is connected with cells J and M(i-1, j). Across the lattice, Cell J(i, j) is connected with cells I and M(i-1, j). Cell M(i, j) is connected with cells J and I(i-1, j-1). The most likely solution is then the best scoring path from the top-left to the bottom-right corner.



This concludes the lecture five.