Outline

- Scoring Matrices
- Alignment with Affine Gap Penalties
- Local Alignment
- Multiple Alignment

In this part we consider certain generalization of the definition of the concept of 'similarity' leading to scoring matrices that are deterministic or probabilistic. We will also consider 'gaps' in the alignment and also local similarity of a much smaller length string against a larger string. We then consider alignment of a set of strings, the set containing more than two strings.

The Basic Problem

A gene or a protein may be related to another gene or protein. “Relatedness” may mean

1. They are homologous if they shared a common ancestry.
2. They may have common functions.

Analysis of DNA or protein sequences (the sequence of amino acids or “residues”) may reveal certain domains or “motifs” that are shared among a group of molecules. Protein alignments give more information than DNA alignments. This is because certain DNA mutations, particularly at the third location in a codon, do not change the protein. Such mutations are called silent mutations. Also mutations in the intron regions of a DNA has practically no effect on the protein. When a DNA sequence is analyzed, it is common practice to analyze the translated amino acid sequence.

Protein sequence comparison can identify homologous sequences that originated from a common ancestor over 1 billion years ago (BYA) whereas DNA sequences can look back up to 600 MYA (millions of years ago).

But there are situations where the DNA sequence must be identified viz. to locate a gene or a motif, searching for polymorphism or identifying a cloned CDNA fragment.

Need to Develop Scoring Matrices

- Two sequences are either homologous or not homologous. Statements like two sequences are 20% or 50% homologous are wrong. The only relevant criterion to be homologous is that they are originated from a common ancestral sequence. But, it is correct to say that two homologous sequences are 20% or 50% similar if 20% or 50 % of nucleotides or residues are identical (matched). The cost of substitution of one nucleotide for another nucleotide is set arbitrarily in models for DNA comparison, but two amino acids may not “matched” but may still be biochemically or biophysically related and may command a large similarity score. These are called conservative substitutions.
- Thus definition of scoring matrices are essential for comparing amino acid sequences.

Homologous: Orthologous and Paralogous

- Homologous proteins may be orthologous or paralogous. Orthologs are homologous sequences in different species that arose from a common ancestor. For example, humans and rodents diverged 80 MYA ("millions of years ago") when a single ancestral myoglobin gene diverged by speciation. Orthologs have similar biological function viz. the myoglobin transport oxygen in both human or rat.
- Genes are often duplicated to produce multiple copies in the same genome, which often diverge in functions slightly. Where the homology is the result of gene duplications so that the copies have descended side by side during the history of the organism, the genes should be called paralogous (para=parallel). For example, human α-1 globin and β globin are paralogous.

Key Issues

- The scoring system to rank alignments
- The algorithm complexity to find the optimal or good alignments
- Statistical and Biological significance of an alignment score.

Consider the following pairwise alignments (All from same region of the human alpha globin protein sequence SWISS-PORT data base of proteins. The middle line shows identical (red) and 'similar' (+ sign,blue) meaning functionally conservative:

a)

```
HBA_HUMAN :  GS  A QVK G HGKKVADALTNAV AHVDDMPNALSALSDLHAHKL
             G+    +VK +  HGKKV     A+++++ AH+D+ +   + ++++ LS+LH     KL
HBB_HUMAN :   GN P KVK A HGKKVLG  AFSDGLAHLDNL KGTFAT   LSELHCDKL
```

This shows clear similarity of human alpha globin to beta globin.

(b)

HBA_HUMAN : **GSAQVKGHGKKVADALTNAVAHV- - - D- - DMPNALSALSDLHAHKL**

++ + + ++H+ KV + +A ++ +L+ L+ ++H+ K

LGB2_LUPLU : NNPELQAHAGKVFKLVYEAAIQLQVTGVVVT DATLKNLGSVHVSKG

This shows a biologically meaningful alignment between leghaemoglobin and yellow lupin. These two sequences are evolutionary related and have same three Dimensional structure , and function in oxygen binding. Note much fewer match Characters and 'gaps' inserted to maintain alignment

(c)

HBA_HUMAN: **GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSD- - - - LHAHKL**

GS+ + G + +D L ++ H+ D+ A +AL D ++AH+

F11G11.2 GSGYLVGDSLTFVDLL - - VAQHTADLLAANAALLDEFPQFKAHQE

A spurious high-scoring alignment to a nematode glutathione S-tranferase homolog named F11G11.2. The two proteins have totally different structure and function.

How do we differentiate cases like (b) and (c). This calls for a careful definition of the scoring system that we use to evaluate alignments.

Additive Score Model

We consider the mutations at different sites in a sequence to be Independent . A “gap” of arbitrary length considered to be a single mutation.

This assumption is a reasonable approximation for DNA and protein sequences (although we know that interactions between the bases play a significant role for protein structures).

For RNA sequences, this assumption is erroneous since the base pairing gives rise to long-distance dependencies due to the folding structures. We leave out RNA sequences from our discussions.

Substitution Matrices

- Given a pair of aligned sequences, how do we assign a **score** that gives relative likely hood that the sequences are related? Earlier, we discussed the general scoring equations with respect to DNA and protein sequences using constant cost parameters such as μ and σ, but their applications to specific biological context is a bit more involved process. A convenient way to specify scoring is a matrix called the **Substitution Matrix** which is $|\Sigma|+1$ by $|\Sigma|+1$ matrix giving the values of $\delta(a,b)$ for all possible (a,b) except (-,-). Ideally, the scores should capture the underlying evolutionary or bio-chemical properties of the sequences.

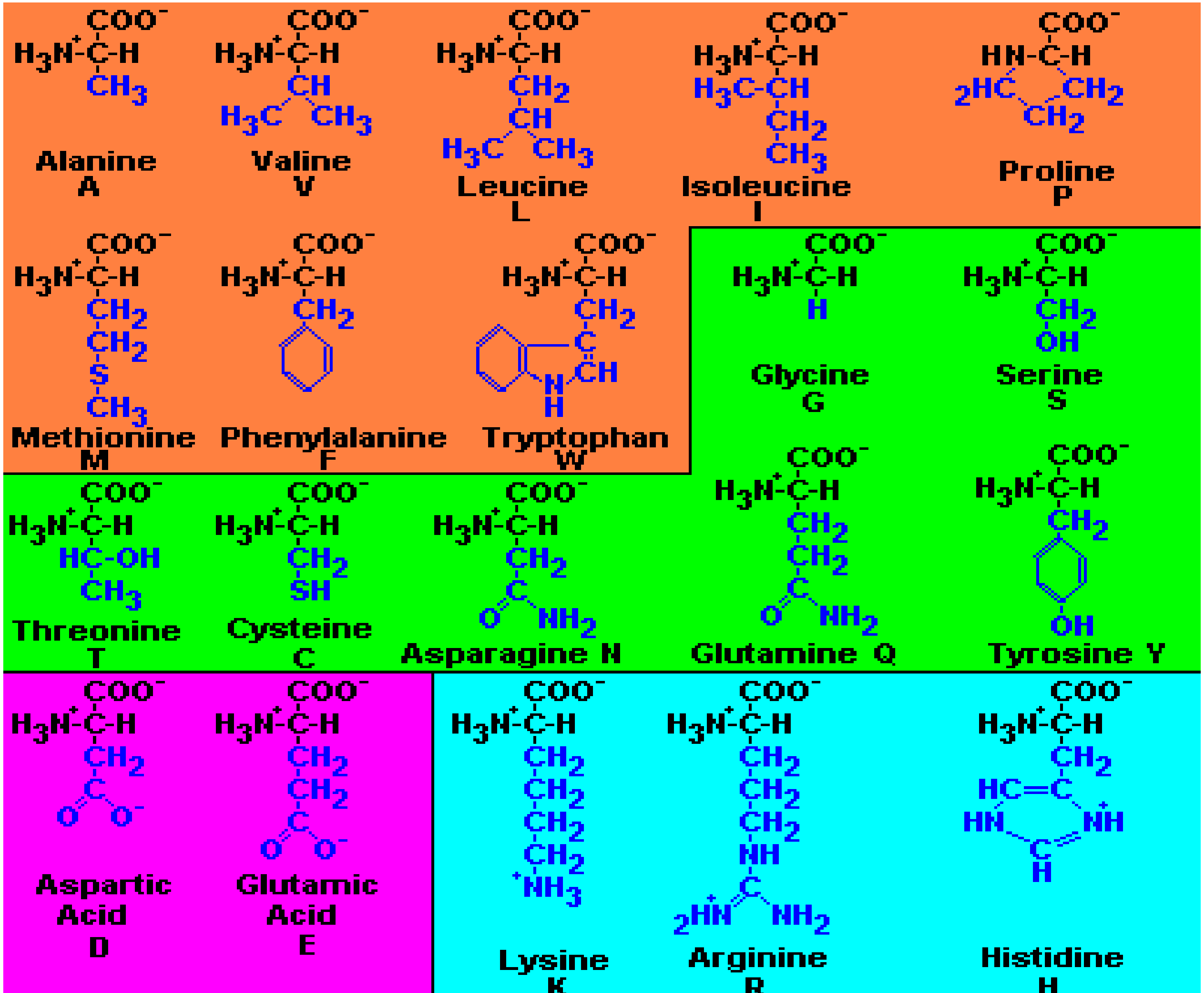
Substitution Matrices

Since nucleotides differ very little in biochemical functions, simple scoring functions are reasonable for DNA sequences. But random mutations of the nucleotide sequence within a gene may change the amino acid sequence of the corresponding protein. Some of these may produce drastic change in the structure and function of the protein while others do not affect the fitness of the organism. The amino acids Asn, Asp, Glu, and Ser are the most mutable amino acids ; Cys and Trp are the least mutable. Knowledge of the frequency of occurrences of most and least common evolutionary events allow Biologists to define models to derive scoring matrices for computing biologically relevant alignments.

Amino acid similarities

- Leucine (L) and Isoleucine (I) biochemically similar
  - High score for subsitution = +2
  - But not as high as no change (L,L) or (I,I) = +4
- Leucine (hydrophobic) and Aspartic Acid (D) (hydrophilic) biochemically different
  - Low score for substitution = -4
  - Conservative Substitutions: T(Threonine) and S(serine),
  - L(Leucine)and V(Valine).
  - Basic amino acids: (K,R,H)
  - Acidic amino acids : (D,E)
  - Hydroxylated amino acids: (S,T)
  - Hydrophobic amino acids : (W,F,Y,L,I,V,M,A)

20 amino acids



Proteins

Proteins are polymers, also called polypeptides consisting of a sequence of amino acids. There are twenty amino acids that are found in proteins.

Hydrophobic Group			Hydrophilic Group		
A	**Alanine**	**ala**	**R**	**Arginine**	**arg**
C	**Cysteine**	**cys**	**N**	**Asparagine**	**asn**
G	**Glycine**	**gly**	**D**	**Aspartic acid**	**asp**
I	**Isoleucine**	**ile**	**Q**	**Glutamine**	**gln**
L	**Leucine**	**leu**	**E**	**Glutamic acid**	**glu**
M	**Methionine**	**met**	**H**	**Histidine**	**his**
F	**Phenylalanine**	**phe**	**K**	**Lysine**	**lys**
P	**proline**	**pro**	**S**	**Serine**	**ser**
T	**Trypyophan**	**trp**	**T**	**Threonine**	**thr**
Y	**Tyrosine**	**tyr**			
V	**Valine**	**val**			

Making a Scoring Matrix

- Scoring matrices are created based on biological evidence.
- Alignments can be thought of as two sequences that differ due to mutations.
- Some of these mutations have little effect on the protein's function, therefore some penalties, $\delta(v_i , w_j)$, will be less harsh than others.

Scoring Matrix: Example

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

AKRANR

KAAANK

-1 + (-1) + (-2) + 5 + 7 + 3 = 11

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acids→ will not greatly change function of protein.

Construction of Scoring Matrix

The entry δ(a,b) in the scoring matrix for proteins usually denotes how often the amino acid ‘a’ substitutes the amino acid ‘b’ in the alignments of related protein sequences. The most commonly used scoring matrices ***point accepted mutation (PAM)*** and ***block substitution matrix (BLOSUM)*** are created by using this principle.

Margaret Dayhoff and her team made significant contributions in this field in the 70’s. They published a book **Atlas of Protein Sequence and Structure** which listed all protein sequences known in late 70’s along with information about their structures and functions. They defined an **accepted point mutation(APM** –changed to **PAM** for ease of pronunciation)as a substitution that has been accepted by natural selection. This happens if

1. A gene undergoes a DNA mutation such that it encodes a different amino acid.
2. The entire species adopts that change as the predominant form of the protein.

Dayhoff,M.O.,Schwartz,R.M. and Orcutt, B. C.,”A model of evolutionarychange in protein “ in “Atlas of Protein Sequence and Structure”, Vol.5, National Biomedical Research Foundation, Washington D.C. Pp.89-90, 1978. (Also read pp.178-180 from text by Jones and Pevzner)

Dayhoff's Method of Construction of Scoring Matrices

Dayhoff and her team used phylogenetic tree analysis technique s on 1572 changes in 71 groups of extremely similar proteins. Such analysis allows comparison of extant amino acid sequences to inferred ancestral sequences. This approach involved phylogenetic analysis rather than comparing the two amino acid sequences directly.

For the PAM 1 matrix, they identified proteins that have undergone 1% change (that is, 1 accepted point mutation per 100 amino acid residues). Such sequences are defined as being one PAM unit diverged. Such alignments are called the **base alignments**.

Given a set of base alignments, let

$$f(a,b) = \frac{\#\ \text{of times amino acids } a \text{ and } b \text{ are aligned against each other}}{\text{total number of aligned positions}}$$

Let $f(a)$= frequency of amino acid a in all positions from the data set.

$g(a,b) = \dfrac{f(a,b)}{f(a)}$ = probability that an amino acid a mutates into amino acid b within 1 PAM unit

The (a, b) entry of the PAM-1 matrix is defined as $\delta(a,b) = \log \dfrac{f(a,b)}{f(a)f(b)} = \log \dfrac{g(a,b)}{f(b)}$

The quantity $f(a).f(b)$ is the joint probability of aligning a with b by chance.

The PAM-n matrix is defined as the result of applying PAM-1 matrix n times. If G is the 20X20 matrix of frequencies $g(a,b)$, then G^n (multiplying this matrix by itself n times) gives the probability that amino acid a mutates into amino acid b during n PAM units. The (a,b) entry of the PAM-n matrix is defined as $\log \dfrac{G^n}{f(b)}$

For large n, the resulting PAM matrices often allow one to find related proteins even when there are practically no matches in the alignment. The underlying DNA sequences are so diverged that there comparison will not find any statistically significant biological similarities.

Scoring Two Amino Acid Sequences

- **Random model (R)**: Assumes that every amino acid in the sequence occurs independently. Thus, the probability of the two sequences is the product of probabilities of each amino acid.

$$P(S_1, S_2 \mid R) = \Pi p_i \Pi p_j$$

  where p_i and p_j *denote probabilities of ith and* jth symbols in the two sequences.
- **Match model (*M*)** *:* the aligned pair of bases occur with a joint probability p_{ij}. We can think of p_{ij} as the probability that the residues *i* and *j* have been derived from some unknown original base *k* in their common ancestor *(k* could be same as *i* and/or *j).* Thus

$$P\ (S_1, S_2 \mid M) = \Pi\ p_{ij}$$

Substitution matrices

The ratio of these two likelihood is known as the ***odds-ratio***

$$\frac{P(S_1, S_2 \mid M)}{P(S_1, S_2 \mid R)} = \frac{\Pi p_{ij}}{\Pi p_i p_j}$$

In order to arrive at an additive scoring system, we take the sum of logarithm of this ratio, known as the **log-odds ratio *S***:

$$S = \sum_i s(S_1(i), S_2(i))$$

Where $s = \log \frac{\Pi p_{ij}}{\Pi p_i \Pi p_j}$

is the log likelihood of the residue pair occurring as an aligned pair. These scores can then be arranged as a 20X20 matrix (for protein sequences) with $s\,(a_i, a_j)$ in position *i,j where* a_i and a_j are the *i*th and *j*th amino acids.

PAM10 - see separate posting

The values for p_{ij} are called the "target frequencies" and they indicate the amount of evolutionary change that has taken place. For example, for a pair of closely related proteins, an aligned Serine were to change to threonine 5% of the time, the target frequency for $p_{S,T}$ is 0.05. The quantity $p_i p_j$ represents probabilities of amino acids *I* and *j* occurring by chance. These frequencies are derived by first calculating relative mutabilities (how often a residue is going to change in a short period of evolution) and dividing them by the overall frequencies of occurrences of these residues.
The less mutable residues are those that play a more significant role in the function and structure of the protein (viz. one residue mutation is known to cause cystic fibrosis) whereas more mutable residues like Asp, Ser, Asn and Gly have functions in protein that can be easily assumed by other residues. These phenomena are obviously related to the genetic code and how the mutations in the triplets of the reading frame affect the production of protein

Table 2-6 PAM-250 Matrix in "Log Odds" Form (Ratios expressed as base 10 logarithms)

C	12																			
S	0	2																		
T	-2	1	3																	
P	-3	1	0	6																
A	-2	1	1	1	2															
G	-3	1	0	-1	1	5														
N	-4	1	0	-1	0	0	2													
D	-5	0	0	-1	0	1	2	4												
E	-5	0	0	-1	0	0	1	3	4											
Q	-5	-1	-1	0	0	-1	1	2	2	4										
H	-3	-1	-1	0	-1	-2	2	1	1	3	6									
R	-4	0	-1	0	-2	-3	0	-1	-1	1	2	8								
K	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5							
M	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6						
I	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5					
L	-8	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	8				
V	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4			
F	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9		
Y	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10	
W	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**

BLOSUM

Blocks **Su**bstitution **M**atrix represents a statistical alternative to PAM. It gives a more accurate measure of differences of distantly related proteins. It is more effective for searching sequences that contain relatively sparse regions of close evolutionary relatedness.

PAM assumes that the mutations occur at a constant rate – the same rate that is observed in the short term. BLOSUM is derived by observing all amino acid changes from a protein families in an aligned region without any bias regarding the level of similarity between the two sequences.

PAM matrices are based on scoring of all amino acid positions, but BLOSUM matrices are based on conserved positions that occur in blocks representing the most similar regions of related sequences.

BLOSUM

- **Blo**cks **Su**bstitution **M**atrix
- Scores derived from *observations* of the frequencies of substitutions in blocks of local alignments in related proteins
- Matrix name indicates evolutionary distance
  - BLOSUM62 was created using sequences sharing no more than 62% identity

The Blosum50 Scoring Matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*****	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

The log-odds values have been scaled and rounded To the nearest integers for the purpose of computational efficiency.

Substitution matrix: BLOSUM62

	A	C	D	E
A	**4**	0	−2	−1
C	0	**9**	−3	−4
D	−2	−3	**6**	2
E	−1	−4	2	**5**

Subsititions, e.g. C+E
score depends on how likely the two amino acid types are to substitute for each other.

Identities, e.g. E+E
High scores (main diagonal).

Matrix is symmetrical

BLOSUM62 -- See separate posting

Optimal Alignment with Gap

- Definition:
  - A ***gap*** in an alignment between two strings, is a run of contiguous spaces.
- An insertion or deletion of a character was represented by a space.
- Each occurrence of such a space character in the alignment is considered to be a mutation.
- Sometimes a gap of more than one space can be created by only one mutational or evolutionary event. To handle this kind of situation, we need to develop a model of alignment cost function that does not attribute a negative cost or penalty based on the length of the gap (which is called a linear model).

Biological Significance of Gap

- Examples of 'gaps' in biological context is numerous.
- The case of cDNA is a good biology application.
  - In a genome, not all DNA are responsible for the production of proteins or hormones;
  - those that carry these functions are said to be **expressed**.
- To study this phenomenon, biologists make DNA , called **cDNA**, corresponding to **mRNA** that leaves the nucleolus to cytoplasm for translation, by replacing **uracil (*U*)** in RNA by **thyamine (*T*)** .
- Concatenation of these DNA strands then corresponds to complement of the **exon** of the gene, cDNA.

Biological Significance of Gap(contd.)

- If we now sequence the cDNA and compare this with similar DNA in the chromosomal DNA, we would have obtained a map of chromosomal genes that are expressed.
- While doing this similarity search, the **introns** have to be aligned with long gaps.
- Recall a gene may be distributed over several segments with interleaving introns. If we used our scoring scheme for similarity search here, we would have penalized heavily our total score for the alignment (since gap will translate into a set of contiguous delete operations) and the similarity of the cDNA with some segment of chromosomal DNA would be missed.
- The alignment that best reflect the relationship consists of a few regions of strong similarity interspersed with long regions of gaps.

Linear Gap Penalty

- Gaps subtract a value from the objective score
- Simplest design: “**linear**” penalties
- a fixed parameter (*h*) multiplied by length of gap

  “*h*” for “gap extension”

  h = 4 (fixed penalty)

```
L A K E
I - - E
```

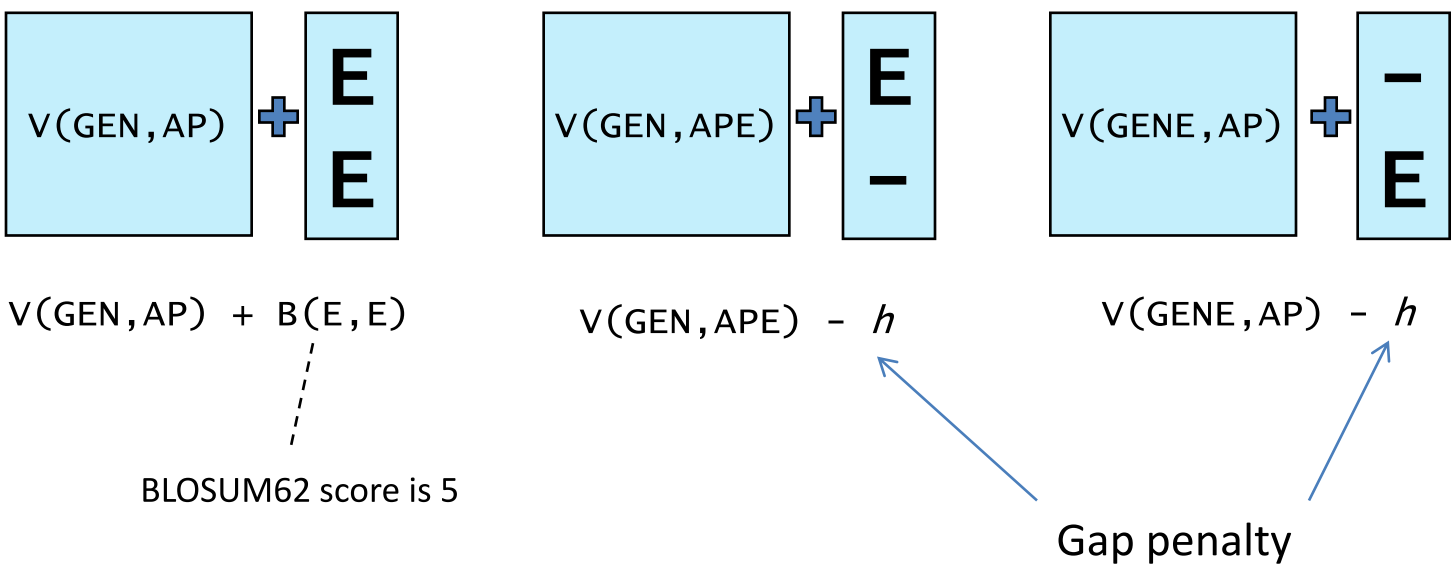
= −1

(L,I) = 2 2*h* = −8 (E,E) = 6

- Subtract e for every “-” in the alignment

Computing Maximum Objective Score

`max Score(GENE,APE) = ?`



Problem with linear gap penalties

```
GRB2_CHICK    ...SVKFGN----D-VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...

GRB2_CHICK    ...SVKFGND-----VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```

- These alignments have same objective score with linear penalties
- But lower alignment is more biologically reasonable
  - One gap instead of two = one insertion / deletion event instead of two

Affine Gap Penalties

- Prefer fewer gaps (parsimony: fewer insert / delete events)
- Gap Penalty= $gap(k) = g + hk$

  g = "gap open" or "per-gap" penalty, typical $g = 9$

  h = "gap extension" penalty, typical $h = 2$

  k = gap length (number of consecutive "-" symbols)

```
GRB2_CHICK    ...SVKFGN----D-VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```

gap penalty $= -2g - 5h = -28$

```
GRB2_CHICK    ...SVKFGND-----VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```

gap penalty $= -g - 5h = -19$

Goto's Algorithm

Ref: E.W.Myers and W. Miller, "Optimal Alignments in Linear Space", CABIOS, Vol.4,No.1, 1988, pp.11-17 (posted in this website)

The alignment problem is often formulated as maximizing similarity score rather than minimizing difference score. Earlier, we discussed the replacement cost as δ(*a,b*) with δ(*a,a*)=0 in the context of difference score. For similarity, a bonus score *σ*(*a,b*) is added for every aligned pair (*a,b*) and a gap penalty *gap*(*k*) is subtracted for every k symbol gap. We also know that if we have the minimum cost solution, we can transform it to a solution of maximum similarity.

Definitions

- Recall A_i denote the i-symbol prefix $a_1a_2\ldots a_i$ of n-symbol sequence $A=a_1a_2\ldots a_n$ and B_j denote the j-symbol prefix $b_1b_2\ldots b_j$ of the sequence $B=b_1b_2\ldots b_m$. Define
- $C(i,j)$= minimum cost of conversion of $A_i \rightarrow B_j$
- $D(i,j)$= minimum cost of conversion of $A_i \rightarrow B_j$ that deletes a_i
- $I(i,j)$= minimum cost of conversion of $A_i \rightarrow B_j$ that inserts b_j

If i=0, $D(i,j)$ is not defined since A_0 is a null string and deleting a null symbol does not make any sense. Similarly, $I(i,j)$ for j=0 is not defined since we cannot insert a null symbol. Taking these boundary conditions into account, the recurrence relations to obtain $C(i,j)$ is given in the next slide.

Recurrence Relations for *C*(*i,j*)

$$C(i,j) = \begin{cases} \min\ [\ D\ (i,j),\ \ I\ (i,j),\ \ C\ (i-1, j-1) + \delta\ (a_i, b_j)] & if\ \ i>0\ \ \text{and}\ \ \ j>0 \\ gap\ \ (j) & if\ \ i=0\ \text{and}\ \ \ j>0 \\ gap\ \ (i) & if\ \ i>0\ \text{and}\ \ \ j=0 \\ 0 & if\ \ i=0\ \text{and}\ \ \ j=0 \end{cases}$$

For i,j>o, the first line is obvious because the alignment must end with a delete, insert or a replacement operation.

For i=0 and j>1, *C*(*i,j*)=*gap*(*j*) , since $A_0 \rightarrow B_j$ corresponds to alignment

$$\begin{array}{l} -\ \ -\ \ -\ \ a_1 a_2 .. a_n \\ b_1 b_2 ... b_j b_{j+1}\ \ ...\ \ b_m \end{array}$$

For i>1 and j=0, *C*(*i,j*)=*gap*(*i*) , since $A_i \rightarrow B_0$ corresponds to

$$\begin{array}{l} a_1 a_2 .. a_i\ a_{i+1} a_{i+2} .. a_n \\ -\ \ -\ \ -\ b_{j+1}\ \ ...\ \ b_m \end{array}$$

For i=j=0, *C*(*i,j*)=0, since $A_0 \rightarrow B_0$ corresponds to an empty alignment.

Recurrence Relation for *D*(*i,j*)

Consider $i>0$ and $j>0$. Possible scenarios for minimum cost conversion of A_i to B_j that deletes a_i are shown. This justifies 1st line for D(I,j) below.

	a_{i-1}	a_i
b_j	--	--

a_{i-1}	--	a_i
--	b_j	--

a_{i-1}	a_i
b_j	--

Gap already exists: $D(i-1,j)+h$

New gap created: $C(i-1,j)+g+h$

For $i=0$ and $j>0$, $D(0,j)$ corresponds to $A_0 \rightarrow B_j$. As we noted earlier, if $i=0$, $D(i,j)$ is not defined since A_0 is a null string and deleting a null symbol does not make any sense. So, we are free to pick the function $D(0,j)$ for i=0 and j>0. Since the alignment starts with a "delete", we define $D(0,j)=C(0,j)+g$.

We need not compute $D(i,0)$ for $i >=0$ since other quantities do not depend on these values.

Thus, we can write

$$D(i,j) = \begin{cases} \min\ [D(i-1,j),\ C(i-1,j)+g] + h & \text{for } i>0 \text{ and } j>0 \\ C(0,j) + g & \text{for } i=0 \text{ and } j>0 \end{cases}$$

Recurrence Relation for *D*(*i,j*) (contd.)

For the case i=1, an optimal Conversion of A_1 to B_j ending

- -- --a_1
$b_1 b_2 b_j$--

with a delete, must convert A_0 to B_j and then delete a_1. Thus,

$$\min\ [D\ (0\ ,\ j),\ C\ (0,\ \ j) + g\ \ ] + \ h = \min\ [\ C\ (0, j) + g,\ \ C\ (0, j) + g\ \ ] + h$$

by substituting $D(0, j) = C(0, j) + g$ which yields:

$$D(1, j) = C(0, j) + gap\ \ (1)$$

This shows that it was okay to define $D(0, j) = C(0, j) + g$
so that the recursion terminates properly.

Recurrence Relation for *I*(*i*,*j*)

- *I* is handled like *D.* Thus, if we define $I(i,0)= C(i,0)+g$ for $i>0$ and ignore $I(0,j)$ for $j>=0$, then

$$I(i,j)= \begin{cases} \min\ [I(i, j-1), C(i, j-1)+g] + h & \text{for } i>0 \text{ and } j>0 \\ C(i,0) + g & \text{for } i>0 \text{ and } j=0 \end{cases}$$

The recurrence relations *C,D* and *I* can be used to write an algorithm as presented in the paper cited earlier.

An Example

v=*AGTAC* w=*AAG*

Cost Model: $\delta(a,b)=1$ if a is not equal to b and with $\delta(a,a)=0$

$gap(k)= g + hk= 2+0.5k$ where g is gap open penalty .

We know that for i=0, j>0 $C(0,j)=gap(j)$ for converting $A_0 \rightarrow B_j$

Now, $D(0,j)$ is not defined since an alignment that ends with deleting a null symbol does not make any sense. We are free to define $D(0,j)$. It is "convenient " to define $D(0,j)$ as

$D(0,j)=C(0,j)+g$ for $j>0$. Thus, $D(0,j)=C(0,j)+g=gap(j)+g$. This yields

$$D(0,1)=gap(1)+g = 2+0.5+2=4.5$$

$$D(0,2)=gap(2)+g = 2+1+2=5$$

$$D(0,3)=gap(3)+g = 2+1.5+2=5.5$$

We need not compute $D(0,0)$,$D(1,0)$, $D(2,0)$,$D(3,0)$,$D(4,0)$ and $D(5,0)$

Similarly, since $I(i,0)= C(i,0)+g$ for $i>0$, we can write $I(i,0)= C(i,0)+g= gap(i) + g$ and obtain

$$I(1, 0) = gap(1) + g = 2 + 0.5 + 2 = 4.5$$

$$I(2, 0) = gap(2) + g = 2 + 1 + 2 = 5$$

$$I(3, 0) = gap(3) + g = 2 + 1.5 + 2 = 5.5$$

$$I(4, 0) = gap(4) + g = 2 + 2 + 2 = 6$$

$$I(5, 0) = gap(5) + g = 2 + 2.5 + 2 = 6.5$$

D

	0	1	2	3
0	*	4.5	5	5.5
1	*			
2	*			
3	*			
4	*			
5	*			

$D(0, j) = C\ (0, j) + g = gap\ \ (\mathrm{j}) + \mathrm{g}$

I

	0	1	2	3
0	*	*	*	*
1	4.5			
2	5			
3	5.5			
4	6			
5	6.5			

$I(i,0) = C(i,0) + g = gap(i) + g$

C

	0	1	2	3
0	0	2.5	3	3.5
1	2.5			
2	3			
3	3.5			
4	4			
5	4.5			

$C(0, j) = gap(j)$
$C(i,0) = gap(i)$

D

	0	1	2	3
0	*	4.5	5	5.5
1	*	5	5.5	6
2	*			
3	*			
4	*			
5	*			

$\min\ [D\ (i-1\ , j), C\ (i-1\ ,\ j) + g] + h\ \ (\ i = 1,\ j > 0)$

I

	0	1	2	3
0	*	*	*	*
1	4.5	5	2.5	3
2	5			
3	5.5			
4	6			
5	6.5			

$\min\ [I\ (i\ , j-1), C\ (i\ ,\ j-1) + g] + h$

$(\ i > 0\ ,\ j = 1)$

	0	1	2	3
0	*	4.5	5	5.5
1	*	5	5.5	6
2	*			
3	*			
4	*			
5	*			

	0	1	2	3
0	*	*	*	*
1	4.5	5	2.5	3
2	5			
3	5.5			
4	6			
5	6.5			

Now, we can compute the first row of C(1,j) using the general formula :

$$\min\ [\ D\ (i,j),\ \ I\ \ (i,j),\ \ C\ \ (i-1,j-1)+\delta\ (a_i,b_j)]\quad if\ \ i>0\ \ \text{and}\ \ \ j>0$$

0	1	2	3
0	2.5	3	3.5
2.5	0	2.5	3
3			
3.5			
4			
4.5			

	0	1	2	3
0	*	4.5	5	5.5
1	*	5	5.5	6
2	*	2.5	5	5.5
3	*	3	3.5	2
4	*	3.5	4	4.5
5	*	4	4.5	5

D

	0	1	2	3
0	*	*	*	*
1	4.5	5	2.5	3
2	5	5.5	5	3.5
3	5.5	6	5.5	6
4	6	6.5	6	5.5
5	6.5	7	6.5	7

I

	0	1	2	3
0	0	2.5	3	3.5
1	2.5	0	2.5	3
2	3	2.5	1	2.5
3	3.5	3	3.5	2
4	4	3.5	3	4.5
5	4.5	4	4.5	4

C

Thus the computation proceeds as follows:

1. ***Compute 0-th row and 0-th column of matrices D, I, C***
2. ***Compute the 1^{st} rows of D and I***
3. ***Compute the 1^{st} row of C***
4. ***Compute the 2^{nd} row of D and I***
5. ***Compute the 2nd row of C***

Repeat until you finish computation of the last rows of D, I, C

Global alignment (Summary)

- Global alignment
  - all letters from both sequences
- Objective score: substitution matrix + affine gap penalties
- Three cost matrices *C,D,I*
- Three trace-back matrices (if alignment needed as well.)
- Convert to dual problem to get similarity.

Needleman-Wunsch Algorithm

- Global alignment by dynamic programming often called “the Needleman-Wunsch algorithm”
  - Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* **48**(3): 443-53.
  - Paper describes an algorithm with fixed gap penalty (independent of length)
  - First application of dynamic programming to biological sequences

Local Alignments: Smith- Waterman Algorithm

- A particularly interesting variant of similarity search is local alignment or similarity.
- Suppose we have two long DNA sequences in which there is a particularly interesting subsequence representing a gene that are common between the sequences.
  - Doing a global alignment or similarity search will not be able to identify this because there may be a lot of dissimilarity in the rest of the sequence which yield a low value for similarity and a large value of edit distance, none of which say anything about this interesting region.
- If the regions of highly similar local alignment are small, they might get lost in the context of global alignment.
- If we need to detect similarity between two protein sequences which are highly diverged but share a common conserved evolutionary sequence in a particular domain, doing a global alignment or similarity search does not help much.

Local Alignments: Why?

- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.
- Example:
  - Homeobox genes have a short region called the *homeodomain* that is highly conserved between species.
  - A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence

Local alignment

- Often called "the Smith-Waterman algorithm"
  - Smith, T.F. and Waterman, M.S. (1981) Identification of common molecular subsequences. *J Mol Biol* **147**(1): 195-7.
  - Introduces the critical "all prefixes of all suffixes" trick.
- Surprisingly, only small modification of global case will yield an algorithm for local alignment
  - Many more local alignments than global alignments
  - Prior to Smith-Waterman paper, algorithms were much slower

Local vs. Global Alignment

- The <u>Global Alignment Problem</u> tries to find the longest path between vertices *(0,0)* and (*n,m*) in the edit graph.
- The <u>Local Alignment Problem</u> tries to find the longest path among paths between **arbitrary vertices** (*i,j*) and (*i', j'*) in the edit graph.
- In the edit graph with negatively-scored edges, Local Alignment may score higher than Global Alignment

Local vs. Global Alignment (cont’d)

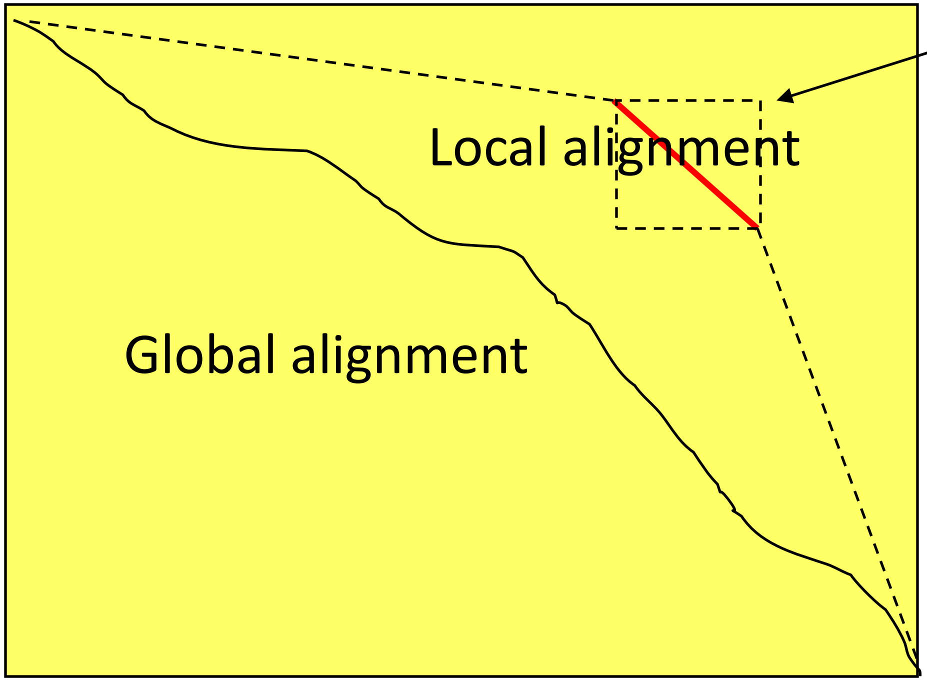
- Global Alignment

```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment

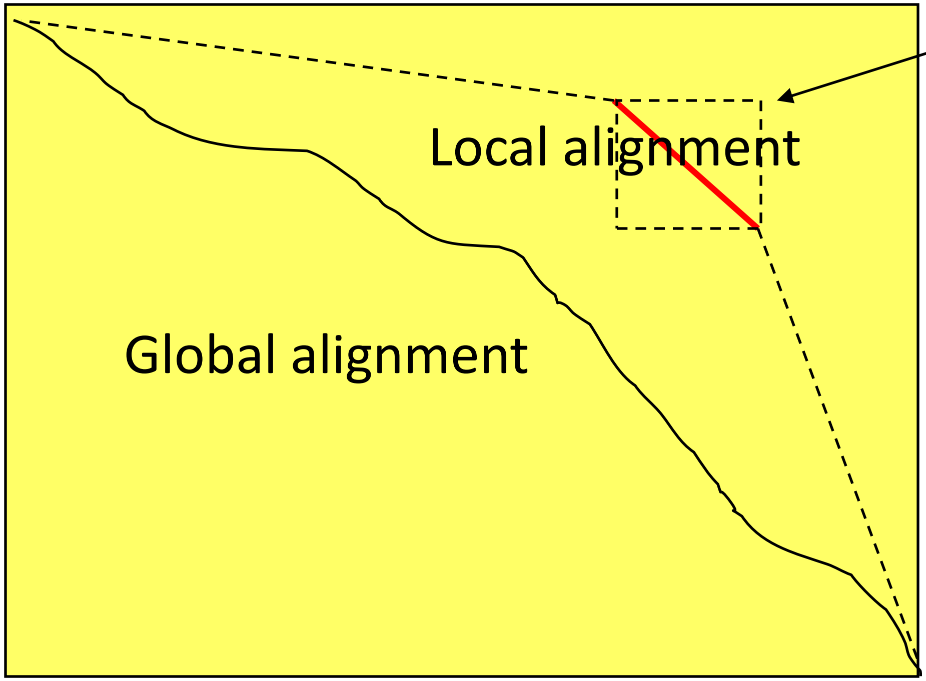
```
                  tccCAGTTATGTCAGggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Local Alignment: Example



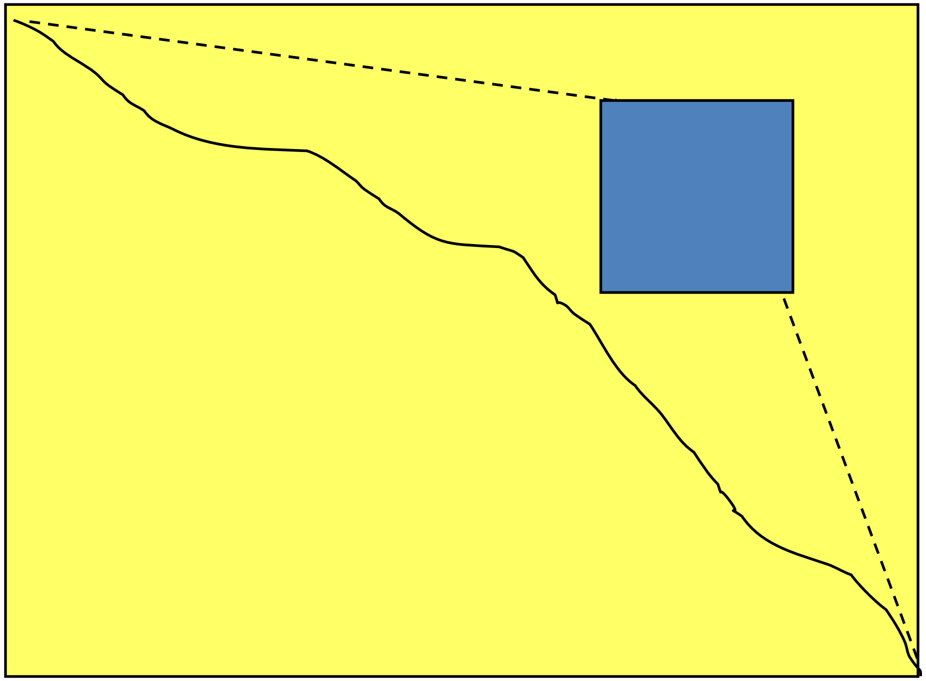
Compute a “mini” Global Alignment to get Local

Local Alignment: Example

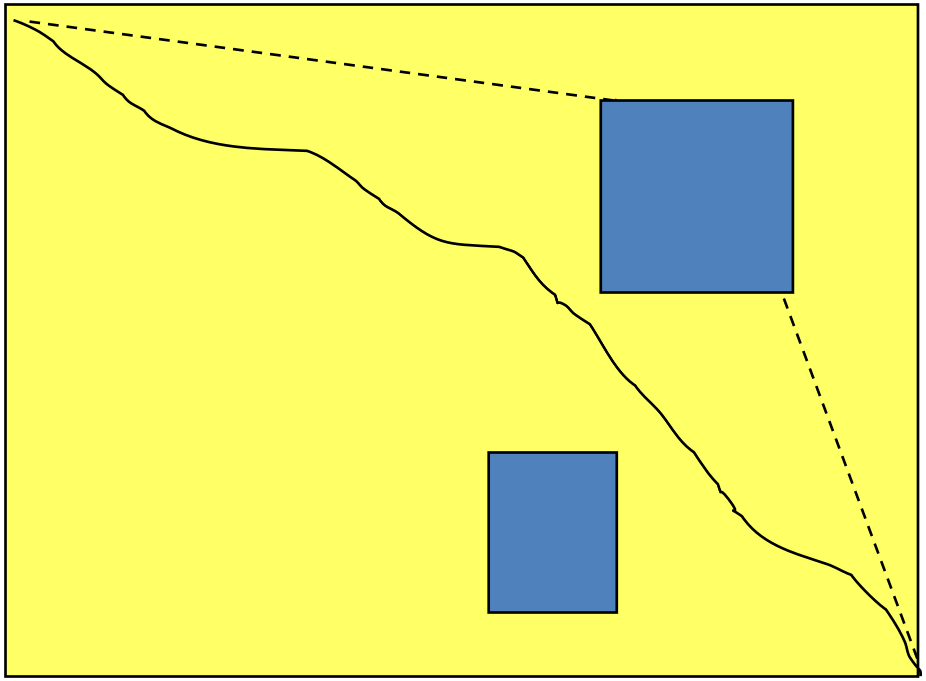


Compute a “mini” Global Alignment to get Local

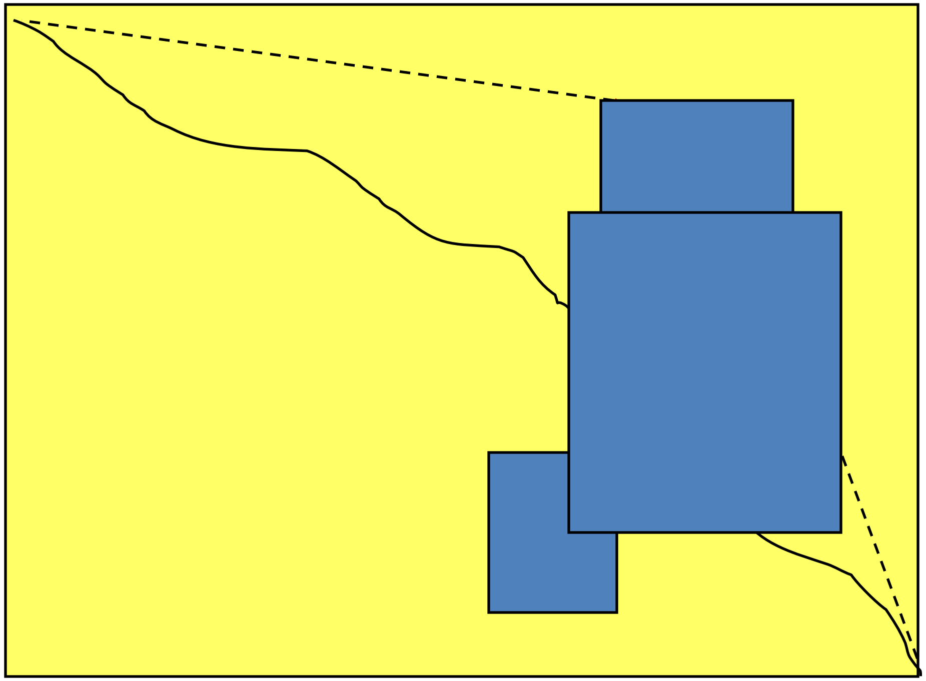
Local Alignment: Example



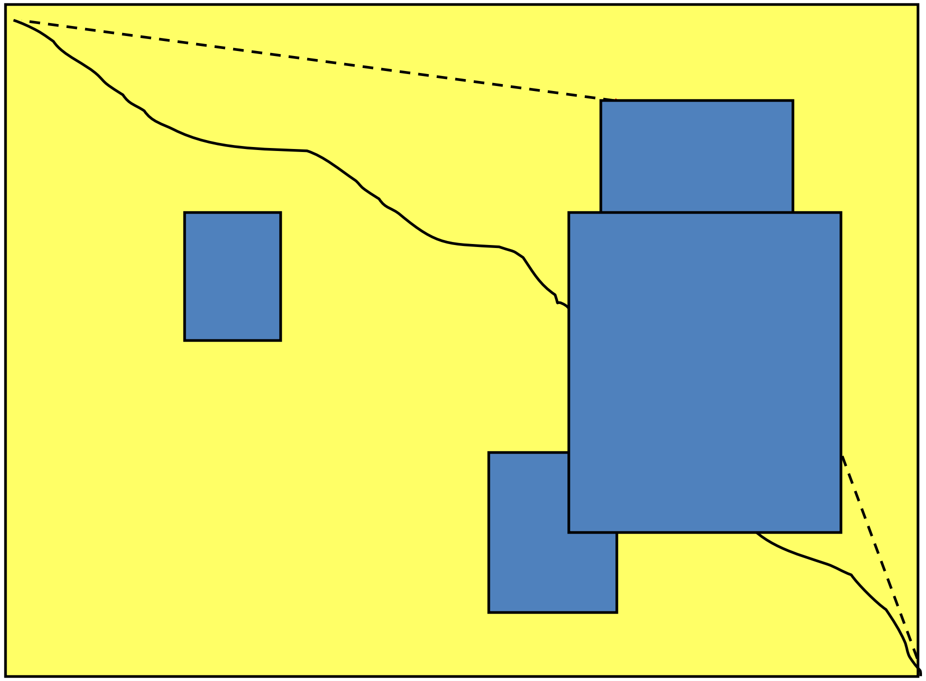
Local Alignment: Example



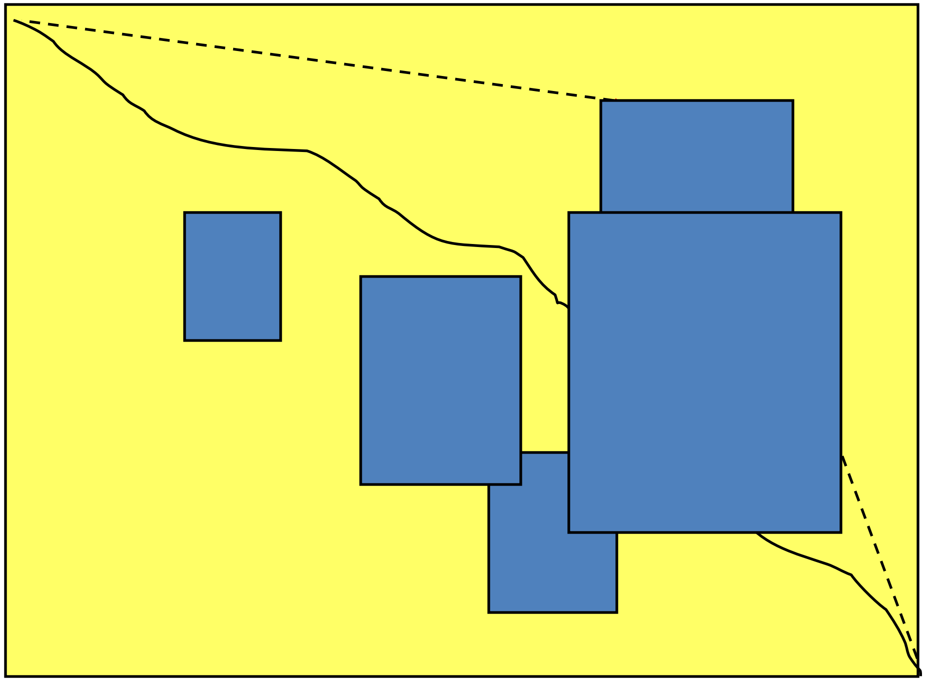
Local Alignment: Example



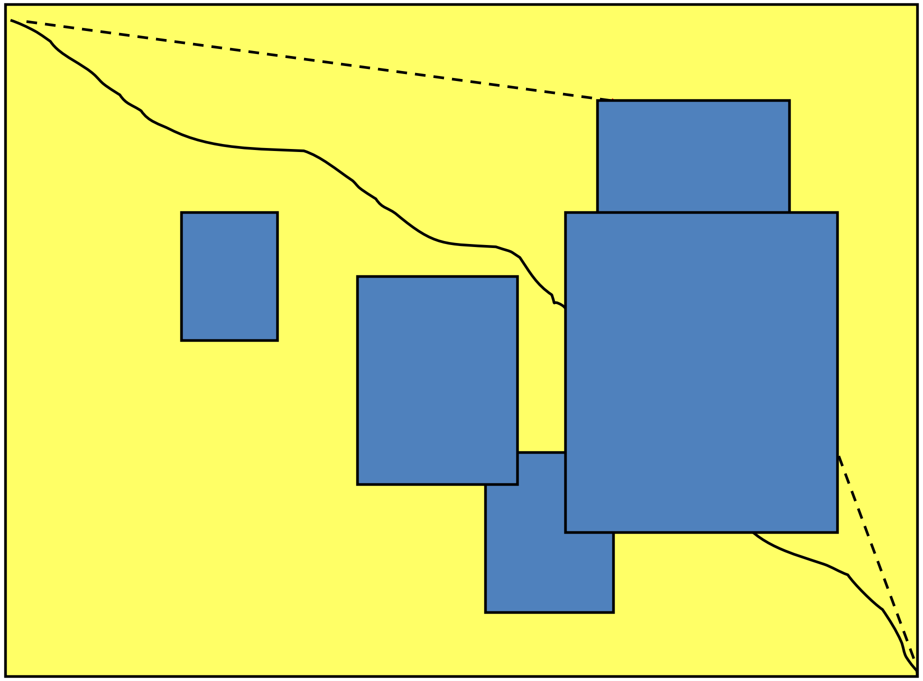
Local Alignment: Example



Local Alignment: Example



Local Alignment: Running Time



The Local Alignment Problem

- Goal: Find the best local alignment between two strings
- Input : Strings **v, w** and scoring matrix δ
- Output : Alignment of substrings of **v** and **w** whose alignment score is maximum among all possible alignment of all possible substrings

The Problem with Exhaustive Algorithm

- An obvious exhaustive algorithm is to enumerate all the substrings of *S1* and *S2* and execute a dynamic programming algorithm on each pair.
  - There are $O(n^2m^2)$ such pairs.
- For one string, a substring is defined by two positions the string which can be chosen in $O(n^2)$ and $O(m^2)$ ways for *S1* and *S2*, respectively.
- For each pair, dynamic programming takes $O(nm)$ time. Thus, the complexity of such an approach is $O(n^3m^3)$.

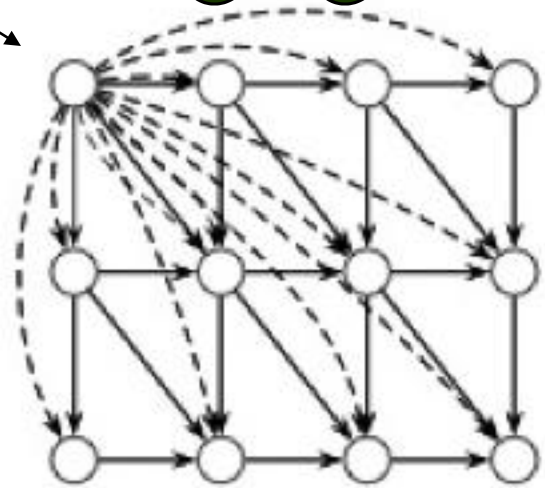
Problem Formulation

The algorithm is essentially a minor modification in the dynamic programming equations for the global alignment with two differences:

1. In each cell in the dynamic programming matrix, an extra possibility is added to allow the value to be 0 if all other options lead to a negative value for that cell. Essentially, it means starting a new computation if the best alignment gives a negative value. This also implies that the first row and column are set to value 0 at the beginning of the computation.
2. The new alignment can end anywhere in the matrix not necessarily at point (n,m) in the matrix. Whenever a local maxima is encountered. The trace back starts and it ends when it meets the first 0 in the path.

Local Alignment: Free Rides

Vertex (0,0)



The dashed edges represent the free rides from (0,0) to every other node.

Algorithm to find value of optimal *V*(*i,j*)

- The algorithm is very similar to the algorithm to determine maximum similarity of two strings.
- Use again recurrence relations.
- Make reasonable assumptions about insert and delete operations as $\delta(-, x) \leq 0$ and $\delta(x,-) \leq 0$, respectively.
- Since the optimal suffix to align with an empty suffix is a string of length zero, we can write the basis as:

$$V(i,0)=0$$

$$V(0,j)=0$$

The Recurrence Realtion

- For *i*>0 and *j*>0, the recurrence relations are:

$$V(i, j) = \max[\,0, V(i-1, j-1) + \delta(S_1(i), S_2(j)),$$

$$V(i-1, j) + \delta(S_1(i), -),$$

$$V(i, j-1) + \delta(-, S_2(j))]$$

Example

Let S = ***ABCLDEL*** and T = ***LLLCDE***, a match score +2, and a mismatch or space score -1. Initialization step:

			ϵ	***L***	***L***	***L***	***C***	***D***	***E***
	j i		0	1	2	3	4	5	6
ϵ	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
A	**1**	**0**							
B	**2**	**0**							
C	**3**	**0**							
L	**4**	**0**							
D	**5**	**0**							
E	**6**	**0**							
L	**7**	**0**							

Example: Let S = ***ABCLDEL*** and T = ***LLLCDE***, a match score +2, and a mismatch or space score -1.

		ϵ	L	L	L	C	D	E
		0	1	2	3	4	5	6
ϵ	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	0	0
B	2	0	0	0	0	0	0	0
C	3	0	0	0	0	2	1	0
L	4	0	2	2	2	1	1	0
D	5	0	1	1	1	1	3	2
E	6	0	0	0	0	0	2	5
L	7	0	2	2	2	1	1	4

The value of optimal alignment is $V(6,6) = 5$. We can construct optimal alignments by retracing from any maximum entry to any zero entry:

		ε	L	L	L	C	D	E
		0	1	2	3	4	5	6
ε	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	0	0
B	2	0	0	0	0	0	0	0
C	3	0	0	0	0	2	1	0
L	4	0	2	2	2	1	1	0
D	5	0	1	1	1	1	3	2
E	6	0	0	0	0	0	2	5
L	7	0	2	2	2	1	1	4

The Optimal Local Alignment

- The optimal local alignments corresponding to these paths are:

C	L	D	E
C	-	D	E

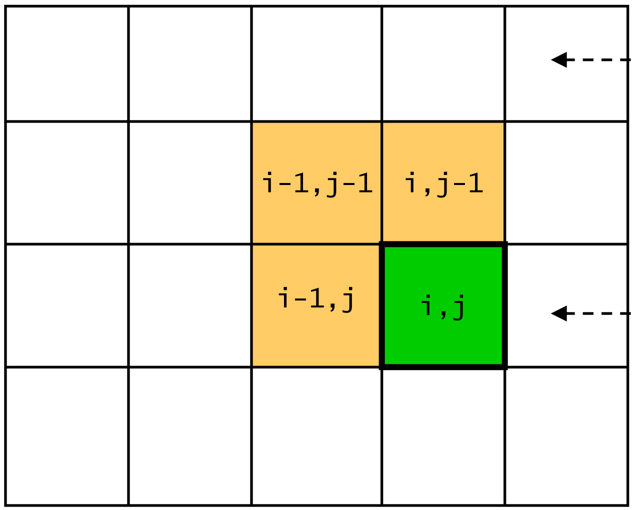
L	-	D	E
L	C	D	E

Space Complexity

- It is easy to see that the time complexity of the algorithm is ***O(nm)*** , as in the general case of dynamic programming.
- The algorithm takes ***O(nm)*** space. This is quite expensive if the sequences are large.
- If one were interested only in the value of the alignment and not obtaining a trace, this could easily be done by keeping only the last two rows of the matrix to compute the next row.
- This will need only ***O(n+m)*** space.
- Is it possible to reconstruct an alignment using only linear space?

Saving space

Compute matrix left-to-right
and top-to-bottom



This row no longer needed

Current row depends only
on previous row and current row

Need only store two rows to
compute score of best alignment
= O(L) space, where L denotes
number of cells in a row or column
whichever is minimum.

(Can it done with space for one row
or column only?).

Trace-back in O(L) space

- Trace-back is harder
- Myers-Miller algorithm
  - Myers, E.W. and Miller, W. (1988) Optimal alignments in linear space. *Comput Appl Biosci* **4**(1): 11-7.
- Repeatedly divides similarity matrix in half

Faster speed

- Speed improvements require approximation
  - give up guarantee that an objective score is optimized
- Global alignment: k-difference
- Local and global alignment: seeds

Freely available source code

- FASTA package
  - align: Myers-Miller global alignment
  - lalign: Smith-Waterman local alignment
  - fasta: fast database search by k-mer matching and d. p. extension
- BLAST (NCBI)
  - Fast database search
  - Seeds by “neighborhood” method
  - Match seeds by lookup in pre-computed index
  - Extend seeds by d. p. with score threshold

Multiple Alignment

Outline

- Dynamic Programming in 3-D
- Progressive Alignment
- Profile Progressive Alignment (ClustalW)
- Scoring Multiple Alignments
- Entropy
- Sum of Pairs Alignment

Multiple Alignment versus Pairwise Alignment

- **Up until now we have only tried to align two sequences.**

Multiple Alignment versus Pairwise Alignment

- **Up until now we have only tried to align two sequences.**
- **What about more than two? And what for?**




Multiple Alignment versus Pairwise Alignment

- **Up until now we have only tried to align two sequences.**
- **What about more than two? And what for?**
- **A faint similarity between two sequences becomes significant if present in many**
- **Multiple alignments can reveal subtle similarities that pairwise alignments do not reveal**




Multiple Sequence Alignment

- Generalization of two sequence similarity problem, the problem of determining the similarity among multiple sequences.
- The purpose is to discover ‘faint but widely dispersed’ common sequences which might represent biologically important information.
- These common sequences might reveal evolutionary history, conserved motifs in the genome of divergent species, common chemical structure that give rise to similar folding or 3-D structures of proteins giving rise to similar functions.

- An example is the notion of ***protein family*** which is a collection of proteins having
  - similar 3-D structure,
  - similar functions,
  - and similar evolutionary history.
- If a new protein is discovered and if one is interested in classifying which family it belongs, comparison with individual members in the family might produce conflicting or confusing results.

Multiple Alignment of Several Amino Acid Sequences of Globin Proteins

- The example below shows how common features are dispersed faintly among a group of proteins which may not be apparent when two sequences in the family are compared.
- The abbreviations on the left denote the organisms that the globin sequences are from. The sequences are displayed in several rows since they are longer than a page can accommodate. Columns containing highly similar residues in regions of known secondary structures are marked by “v” and columns with identical residues are marked by *. Two residues are considered similar if they are from any one of the folowing classes: (F,Y), (M,L,I,V), (A,G),(T,S),(Q,N),(K,R) and (E,D).

An Example of Multiple Alignment

```
                                                 *vvvvv*

HUMA       VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF DLSH    GS
HAOR       MLTDAEKKEVTALWGKAAGHGEEYGAEALERLFQAFPTTKTYFSHF DLSH    GS
HADK       VLSAADKTNVKGVFSKIGGHAEEYGAETLERMFIAYPQTKTYFPHF DLSH    GS
HBHU      VHLTPEEKSAVTALWGKV  NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGN
HBOR      VHLSGGEKSAVTNLWGKV  NINELGGEALGRLLVVYPWTQRFFEAFGDLSSAGAVMGN
HBDK      VHWTAEEKQLITGLWGKV  NVADCGAEALARLLIVYPWTQRFFASFGNLSSPTAILGN
MYHU       GLSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDKFKHLKSEDEMKAS
MYOR       GLSDGEWQLVLKVWGKVEGDLPGHGQEVLIRLFKTHPETLEKFDKFKGLKTEDEMKAS
IGLOB     SPLTADEASLVQSSWK    AVSHNEVEILAAVFAAYPDIQNKFSQFA1GKDLASIKDT
GPUGNI     ALTEKQEALLKQSWEVLKQNIPAHSLRLFALIIEAAPESKYVFSFLKDSNEIPE  NN
GPYL      GVLTDVQVALVKSSFEEFNANIPKNTHRFFTLVLEIAPGAKDLFSFLKGSSEVPQ  NN
GGZLB      MLDQQTINIIKATVPVLKEHGVTITTTFYKNLFAKHPEVRPLF    DMGRQE  SL
```

Example of Multiple Alignment (contd.)

```
                    VVVVV                                      VVVV*

HUMA        AQVKGHGKKVADALTNAV         AHVDDM   PNALSALSDLHAHKLRVDPVNFKLLS
HAOR        AQIKAHGKKVADALSTAA         GHFDDM   DSALSALSDLHAHKLRVDPVNFKLLA
HADK        AQIKAHGKKVAAALVEAV         NHVDDI   AGALSKLSDLHAQKLRVDPVNFKFLG
HBHU        PKVKAHGKKVLGAFSDGL         AHLDNL   KGTFATLSELHCDKLHVDPENFRLLG
HBOR        PKVKAHGAKVLTSFGDAL         KNLDDL   KGTFAKLSELHCDKLHVDPENFNRLG
HBDK        PMVRAHGKKVLTSFGDAV         KNLDNI   KNTFAQLSELHCDKLHVDPENFRLLG
MYHU        EDLKKHGATVLTALGGIL         KKKGHH   EAEIKPLAQSHATKHKIPVKYLEFIS
MYOR        ADLKKHGGTVLTALGNIL         KKKGQH   EAELKPLAQSHATKHKISIKFLEYIS
IGLOB       GAFATHATRIVSFLSEVIAL1SGNTSNAAAV  NSLVSKLGDDHKARGVSAAQ1FGEFR
GPUGNI      PKLKAHAAVIFKTICESA   TELRQKGHAVWDNNTLKRLGSIH LKNKITDPHFEVMK
GPYL        PDLQAHAGKVFKLTYEAA    IQLEVNGAVASDATLKSLGSVHVSKGVVDA HFPVVK
GGZLB       EQPKALAMTVLAAAQNI          ENLPAI   LPAVKKIAVKHC QAGVAAAHYPIVG
```

Example of Multiple Alignment (contd.)

```
              VVVVV             VVV

HUMA        HCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR
HAOR        HCILVVLARHCPGEFTPSAHAAMDKFLSKVATVLTSKYR
HADK        HCFLVVVAIHHPAALTPEVHASLDKFMCAVGAVLTAKYR
HBHU        NVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH
HBOR        NVLIVVLARHFSKDFSPEVQAAWQKLVSGVAHALGHKYH
HBDK        DILIIVLAAHFTKDFTPECQAAWQKLVRVVAHALARKYH
MYHU        ECIIQVLQSKHPGDFGADAQGAMNKALELFRKDMASNYKELGFQG
MYOR        EAIIHVLQSKHSADFGADAQAAMGKALELFRNDMAAKYKEFGFQG
IGLOB       TALVAYLQANVS  WGDNVAAAWNKAL1DNTFAIVVPRL
GPUGNI      GALLGTIKEAIKENWSDEMGQAWTEAYNQLVATIKAEMKE
GPYL        EAILKTIKEVVGDKWSEELNTAWTIAYDELAIIIKKEMKDAA
GGZLB       QELLGAIKEVLGDAATDDILDAWGKAYGVIADVFIQVEADLYAQAVE
```

Family Membership

If the faint similarity of the members in the family can be represented by what is called a '**consensus sequence**', it will be more efficient to find an alignment of the new protein with this consensus sequence to determine whether it belongs to this family.

Definition

Given a set of multiple sequences $S_1, S_2, \ldots. S_k$ a (global) alignment maps them to sequences $S_1', S_2', \ldots. S'_k$ that may contain spaces, where $|S_1'| = |S_2'| =, \ldots = |S'_k|$, and the removal of all spaces from S_i' leaves S_i intact, for $1 \le i \le k$.

Multiple Alignment

- Although the generalization of definition from two sequences to multiple sequences seems straightforward, it is not that obvious how to ***score*** or ***assign value*** to a multiple alignment.
- There are various scoring methods such as ***sum-of –pairs (SP) functions, consensus functions***, and ***tree functions***.
- For the sake of mathematical ease, SP functions have been widely used and good approximation algorithms have also been developed.

Multiple Alignment Methods

1. Exact Approach to Multiple Sequence Alignment
2. Greedy Approach
3. Progressive Sequence Alignment
4. Center Star Algorithm

Other Approaches

5. Consistency Based Approach
6. Structure Based Approach

Exact Approach

- Alignment of 2 sequences is represented as a 2-row matrix
- In a similar way, we represent alignment of 3 sequences as a 3-row matrix

A	**T**	**_**	**G**	**C**	**G**	**_**
A	**_**	**C**	**G**	**T**	**_**	**A**
A	**T**	**C**	**A**	**C**	**_**	**A**

- Score: more conserved columns, better alignment

Alignments = Paths in 3-D Edit Graph

- Align 3 sequences: ATGC, AATC,ATGC

	A	--	T	G	C

	A	A	T	--	C

	--	A	T	G	C

Alignment Paths

0	1	1	2	3	4
	A	--	T	G	C

x coordinate

	A	A	T	--	C

	--	A	T	G	C

Alignment Paths

- Align the following 3 sequences:

ATGC, AATC,ATGC

0	1	1	2	3	4	*x* coordinate
	A	--	T	G	C	
0	1	2	3	3	4	*y* coordinate
	A	A	T	--	C	
	--	A	T	G	C	

-

Alignment Paths

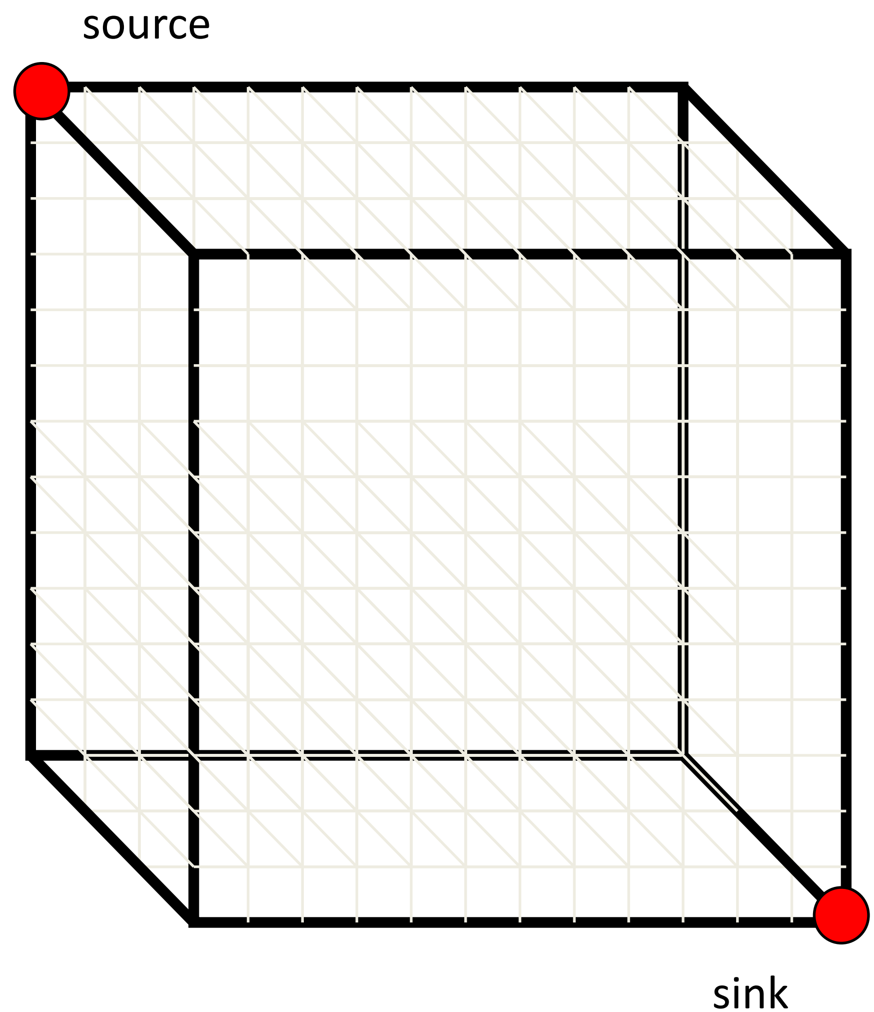
0	1	1	2	3	4	*x* coordinate
	A	--	T	G	C	
0	1	2	3	3	4	*y* coordinate
	A	A	T	--	C	
0	0	1	2	3	4	*z* coordinate
	--	A	T	G	C	

- Resulting path in *(x,y,z)* space:

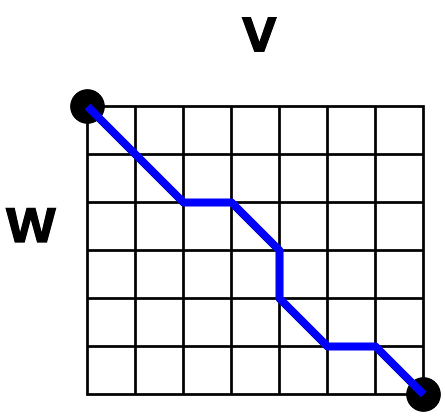
$$(0,0,0)\rightarrow(1,1,0)\rightarrow(1,2,1)\rightarrow(2,3,2)\rightarrow(3,3,3)\rightarrow(4,4,4)$$

Aligning Three Sequences

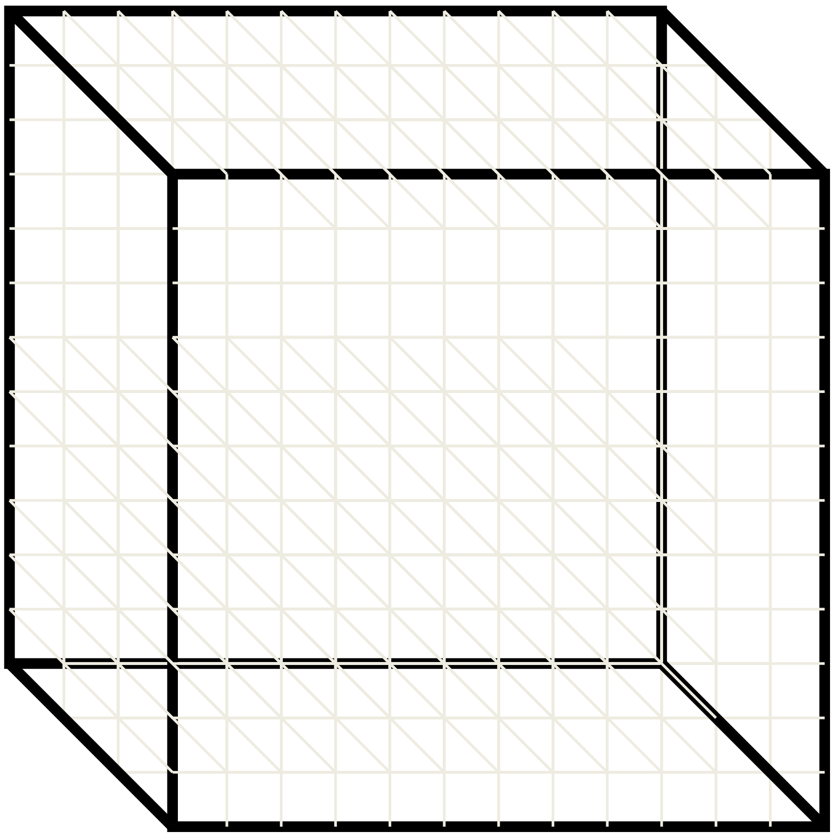
- Same strategy as aligning two sequences
- Use a 3-D “Manhattan Cube”, with each axis representing a sequence to align
- For global alignments, go from source to sink



2-D vs 3-D Alignment Grid



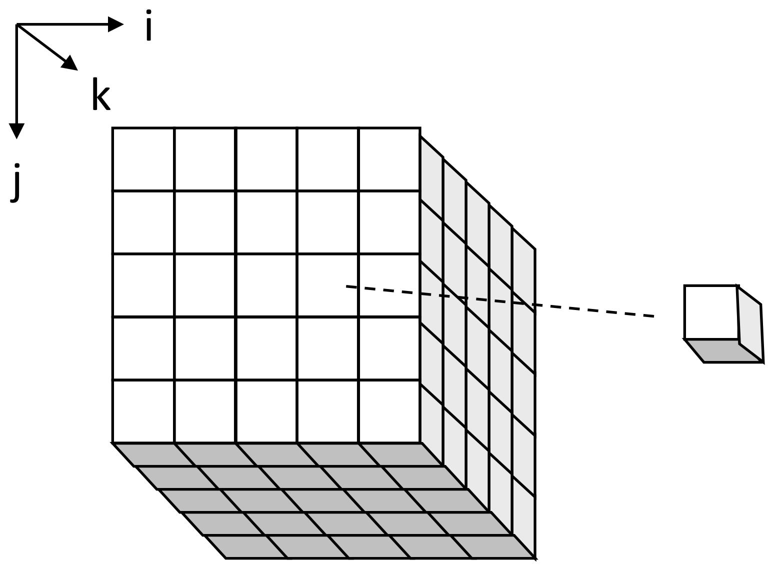
2-D edit graph



3-D edit graph

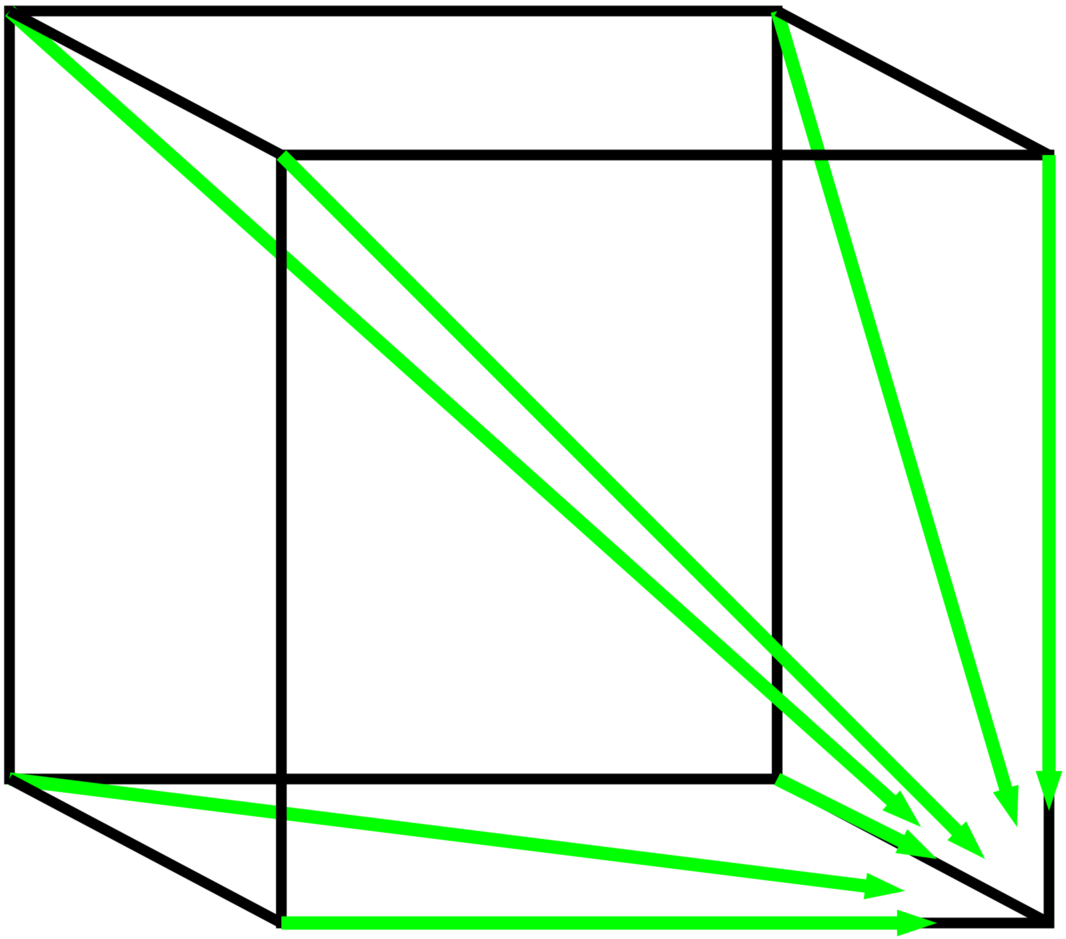
Optimize SP for N sequences

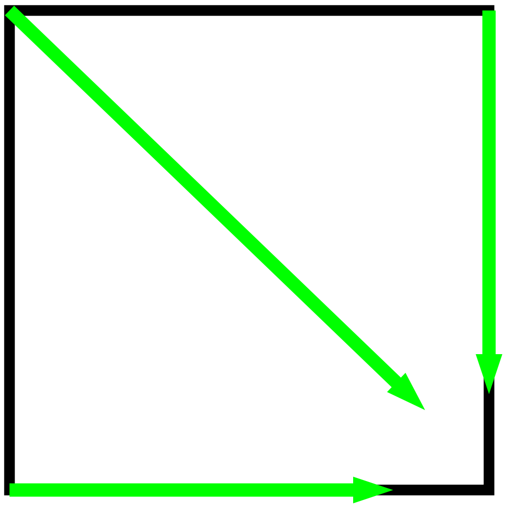
- Similarity matrices become N-dimensional
- E.g., for 3 sequences it will be cubes.



M[i,j,k] =
score of best alignment of
first i letters in A
first j letters in B
first k letters in C

3-D cell versus 2-D Cell

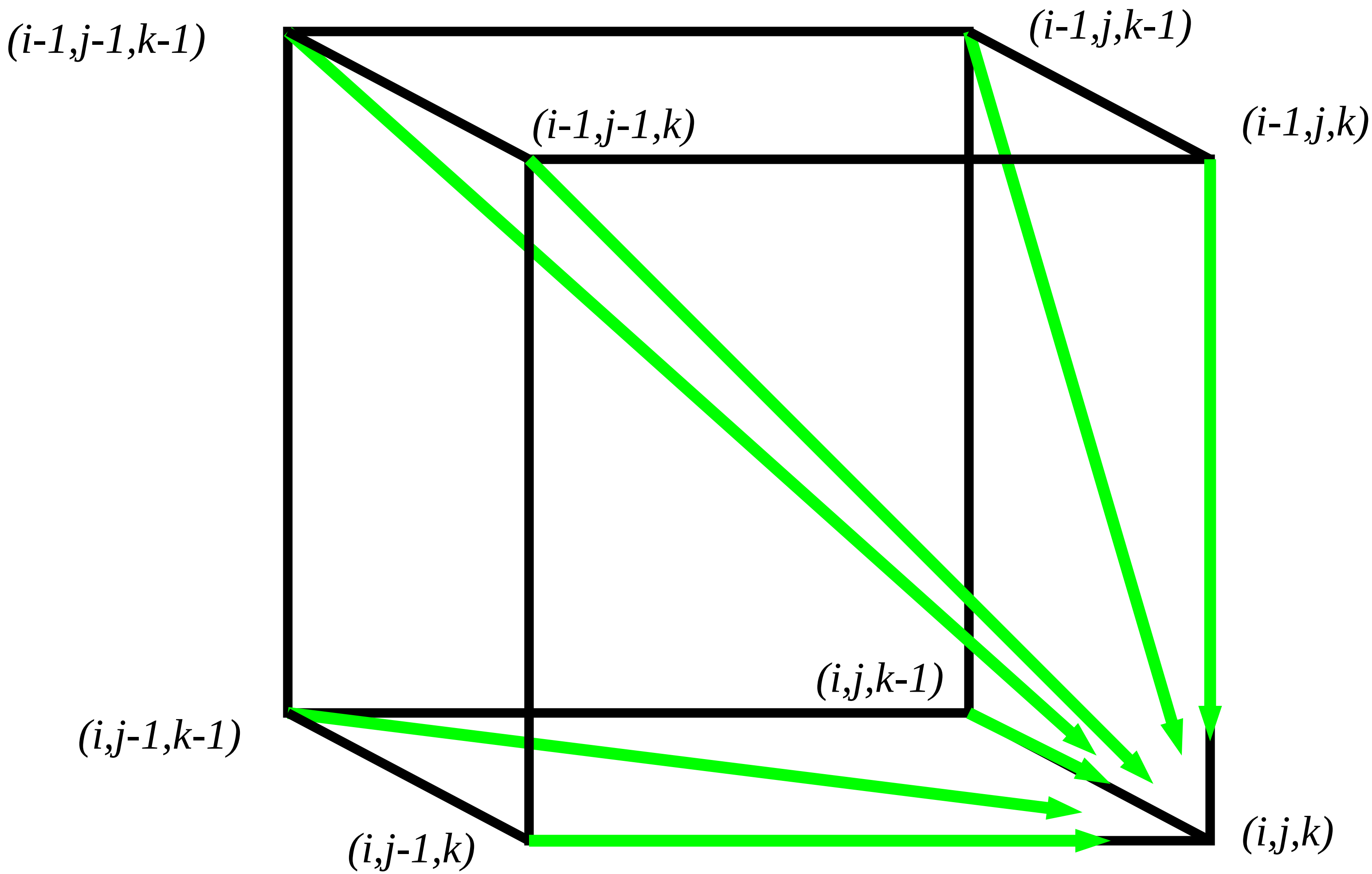




In **2-D**, 3 edges in each unit square

In **3-D**, 7 edges in each unit cube

Architecture of 3-D Alignment Cell



Multiple Alignment: Dynamic Programming

- $$s_{i,j,k} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) & \text{cube diagonal: no indels} \\ \left.\begin{array}{l} s_{i-1,j-1,k} + \delta(v_i, w_j, \_) \\ s_{i-1,j,k-1} + \delta(v_i, \_, u_k) \\ s_{i,j-1,k-1} + \delta(\_, w_j, u_k) \end{array}\right\} & \text{face diagonal: one indel} \\ \left.\begin{array}{l} s_{i-1,j,k} + \delta(v_i, \_, \_) \\ s_{i,j-1,k} + \delta(\_, w_j, \_) \\ s_{i,j,k-1} + \delta(\_, \_, u_k) \end{array}\right\} & \text{edge diagonal: two indels} \end{cases}$$

- $\delta(x, y, z)$ is an entry in the 3-D scoring matrix

Multiple Alignment: Running Time

- For 3 sequences of length $\boldsymbol{n}$, the run time is $O(7m\boldsymbol{np})$ *or* $O(\boldsymbol{n}^3)$ if all sequences have same length n.

- For $\boldsymbol{k}$ sequences, build a $\boldsymbol{k}$-dimensional edit graph, with run time $(2^k-1)(\boldsymbol{n}^k)$ or $O(2^k\boldsymbol{n}^k)$

- Conclusion: dynamic programming approach for alignment between two sequences is easily extended to $\boldsymbol{k}$ sequences but it is impractical due to exponential running time
- It will be a difficult task to define score matrices with real biological significance. See later sum-of-pairs score.

Very slow

- Time and space is $O(n^k)$
- Is NP-complete
  - Wang, L. and Jiang, T. (1994), “On the complexity of multiple sequence alignment” *J Comput Biol* **1**(4): 337-48.
- Totally impractical for most biologically interesting problems
- Faster methods needed

An Objective Scoring Function of Multiple Alignment

- Objective score: Sum-of-pairs (SP)
- Sum of objective score for alignment of each pair of sequences

SP(
```
SEQVENCE
SDQVE-CR
TEQVEACE
```
) =

Score(
```
SEQVENCE
SDQVE-CR
```
) +

Score(
```
SEQVENCE
TEQVEACE
```
) +

Score(
```
SDQVE-CR
TEQVEACE
```
)

Multiple Alignment Induces Pairwise Alignments

Every multiple alignment induces pairwise alignments

```
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
```

Reverse Problem: Constructing Multiple Alignment from Pairwise Alignments

Given 3 arbitrary pairwise alignments:

```
x: ACGCTGG-C;   x: AC-GCTGG-C;   y: AC-GC-GAG
y: ACGC--GAC;   z: GCCGCA-GAG;   z: GCCGCAGAG
```

can we construct a multiple alignment that induces them?

Reverse Problem: Constructing Multiple Alignment from Pairwise Alignments

Given 3 arbitrary pairwise alignments:

```
x: ACGCTGG-C;   x: AC-GCTGG-C;   y: AC-GC-GAG
y: ACGC--GAC;   z: GCCGCA-GAG;   z: GCCGCAGAG
```

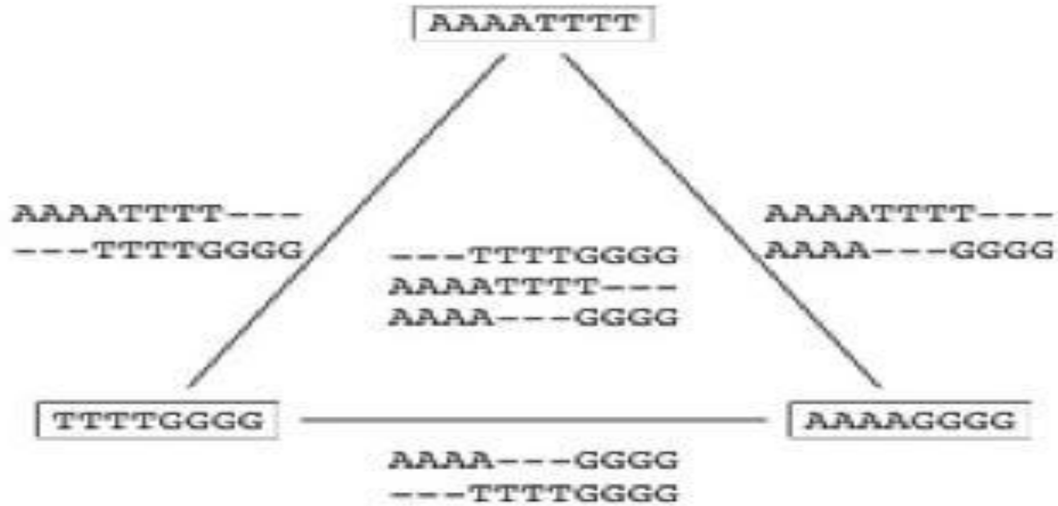
can we construct a multiple alignment that induces them?

NOT ALWAYS

Pairwise alignments may be inconsistent

Combining Optimal Pairwise Alignments into Multiple Alignment:

Can combine pairwise alignments into multiple alignment



(a) Compatible pairwise alignments

Can *not* combine pairwise alignments into multiple alignment

(b) Incompatible pairwise alignments

Inferring Multiple Alignment from Pairwise Alignments

- From an optimal multiple alignment, we can infer pairwise alignments between all pairs of sequences, but they are not necessarily optimal
- It is difficult to infer a ``good” multiple alignment from optimal pairwise alignments between all sequences

Multiple Alignment: Greedy Approach

- Choose most similar pair of strings and align.
- Choose the next sequence that gives maximum score with the existing sequences and insert this sequence with possible insertion of additional space characters. **Repeat.**
- This is a heuristic greedy method

Greedy Approach: Example

- Consider these 4 sequences

s1	GATTCA
s2	GTCTGA
s3	GATATT
s4	GTCAGC

Greedy Approach: Example (cont'd)

There are $\binom{4}{2} = 6$ possible alignments

Cost Model: sub -1, indel -1 and match 1

```
s2   GTCTGA
s4   GTCAGC (score = 2)

s1   GAT-TCA
s2   G-TCTGA (score = 1)

s1   GAT-TCA
s3   GATAT-T (score  = 1)

s1   GATTCA--
s4   G–T-CAGC(score = 0)

s2   G-TCTGA
s3   GATAT-T (score = -1)

s3   GAT-ATT
s4   G-TCAGC (score = -1)
```

Greedy Approach: Example (cont'd)

s_2 *and* s_4 are closest; combine:

s2	GTCTGA
s4	GTCAGC

s1	GAT-TCA
s2	G-TCTGA
s4	G-TCAGC

s1	GAT-TCA
s3	GATAT-T
s2	G-TCTGA
s4	G-TCAGC

Now take the alignment with max score from (*s1,s2*) and (*s1,s4*), which is (*s1,s2*) and add *s4* in it with inserted gaps if necessary. Now, find the best alignment between *s3* and (*s1,s2,s4*) which is (*s1,s3)*

s1	GAT-TCA
s3	GATAT-T

Now, add s3 to the existing alignment with s1.Fortunately, here we do not need to insert additional space characters for s2 or s4

Progressive Alignment

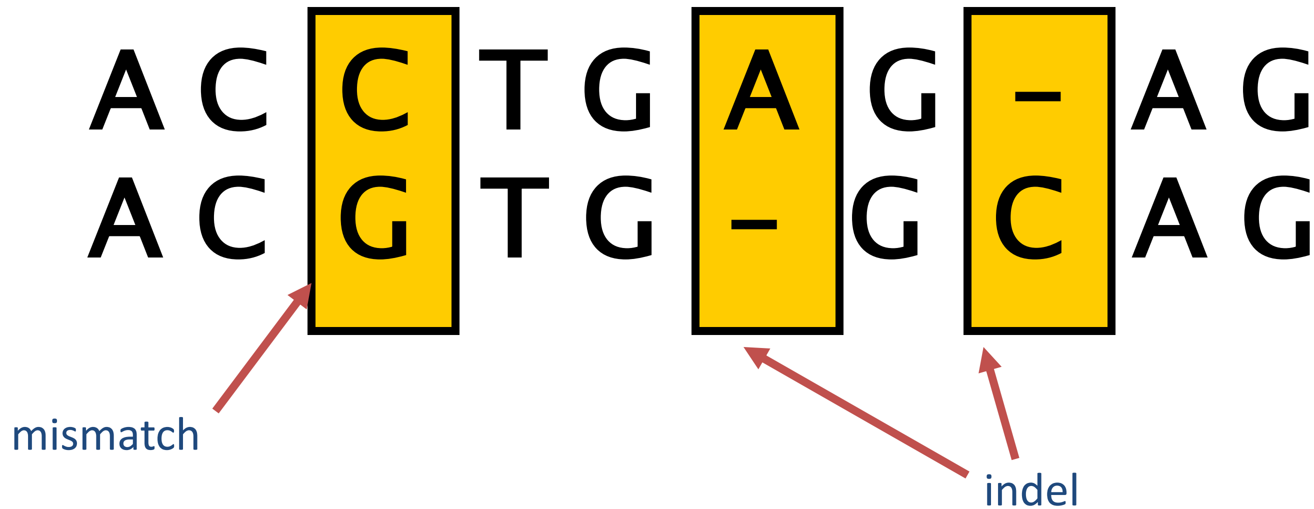
- *Progressive alignment* is a variation of greedy algorithm with a somewhat more intelligent strategy for choosing the order of alignments.
- Progressive alignment works well for close sequences, but deteriorates for distant sequences
  - Gaps in consensus string are permanent
  - Use profiles to compare sequences

ClustalW

- Popular multiple alignment tool today
- ‘W’ stands for ‘weighted’ (different parts of alignment are weighted differently).
- Three-step process
  1.) Construct pairwise alignments
  2.) Build Guide Tree
  3.) Progressive Alignment guided by the tree

Percent Sequence Identity

- The extent to which two nucleotide or amino acid sequences are invariant



70% identical

Step 1: Pairwise Alignment

- Aligns each sequence again each other giving a similarity matrix
- Similarity = exact matches / sequence length (percent identity)

	v_1	v_2	v_3	v_4
v_1	–			
v_2	.17	–		
v_3	.87	.28	–	
v_4	.59	.33	.62	–

(.17 means 17 % identical)

Step 2: Guide Tree

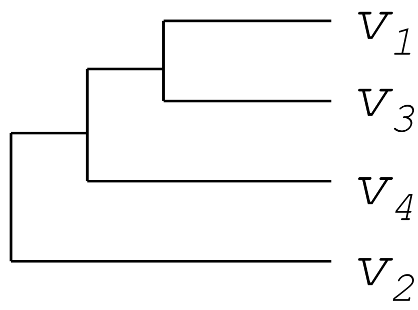
- Create Guide Tree using the similarity matrix

- ClustalW uses the neighbor-joining method

- Guide tree roughly reflects evolutionary relations

Step 2: Guide Tree (cont'd)

	v_1	v_2	v_3	v_4
v_1	-			
v_2	.17	-		
v_3	.87	.28	-	
v_4	.59	.33	.62	-



Calculate:

$v_{1,3}$ = alignment (v_1, v_3)

$v_{1,3,4}$ = alignment$((v_{1,3}),v_4)$

$v_{1,2,3,4}$ = alignment$((v_{1,3,4}),v_2)$

Step 3: Progressive Alignment

- Start by aligning the two most similar sequences
- Following the guide tree, add in the next sequences, aligning to the existing alignment
- Insert gaps as necessary

```
FOS_RAT         PEEMSVTS-LDLTGGLPEATTPESEEAFTLPLLNDPEPK-PSLEPVKNISNMELKAEPFD
FOS_MOUSE       PEEMSVAS-LDLTGGLPEASTPESEEAFTLPLLNDPEPK-PSLEPVKSISNVELKAEPFD
FOS_CHICK       SEELAAATALDLG----APSPAAAEEAFALPLMTEAPPAVPPKEPSG--SGLELKAEPFD
FOSB_MOUSE      PGPGPLAEVRDLPG-----STSAKEDGFGWLLPPPPPPP-----------------LPFQ
FOSB_HUMAN      PGPGPLAEVRDLPG-----SAPAKEDGFSWLLPPPPPPP-----------------LPFQ
                .   . :   ** .     :..  *:.*   *   . *                   **:
```

Dots(less conservative substitution), colon(conservative substitution) and stars(exact match) show how well-conserved a column is.

Multiple Alignments: Scoring

- Number of matches (multiple longest common subsequence score)
- Entropy score
- Sum of pairs (SP-Score)

Multiple LCS Score

- A column is a “match” if all the letters in the column are the same

AAA
AAA
AAT
ATC

- Only good for very similar sequences

Entropy

- Define frequencies for the occurrence of each letter in each column of multiple alignment
  - $p_A = 1, p_T=p_G=p_C=0$ (1st column)
  - $p_A = 0.75, p_T = 0.25, p_G=p_C=0$ (2nd column)
  - $p_A = 0.50, p_T = 0.25, p_C=0.25\ p_G=0$ (3rd column)
- Compute entropy of each column

AAA
AAA
AAT
ATC

$$-\sum_{X=A,T,G,C} p_X \log p_X$$

Entropy: Example

$$entropy\begin{pmatrix} A \\ A \\ A \\ A \end{pmatrix} = 0$$

Best case

Worst case

$$entropy\begin{pmatrix} A \\ T \\ G \\ C \end{pmatrix} = -\sum \frac{1}{4}\log\frac{1}{4} = -4(\frac{1}{4} * -2) = 2$$

Multiple Alignment: Entropy Score

Entropy for a multiple alignment is the sum of entropies of its columns:

$$\sum_{\text{cols.}} \sum_{X=A,C,G,T} -p_X \log p_X$$

Entropy of an Alignment: Example

column entropy:

$$-(\, p_A \log p_A + p_C \log p_C + p_G \log p_G + p_T \log p_T)$$

A	A	A
A	C	C
A	C	G
A	C	T

- Column 1 = -[1*log(1) + 0*log0 + 0*log0 +0*log0]
  = 0
- Column 2 = $-[(^1/_4)*\log(^1/_4) + (^3/_4)*\log(^3/_4) + 0*\log 0 + 0*\log 0]$
  $= -[\ (^1/_4)*(-2) + (^3/_4)*(-.415)\ ] = +0.811$
- Column 3 = $-[(^1/_4)*\log(^1/_4)+(^1/_4)*\log(^1/_4)+(^1/_4)*\log(^1/_4) +(^1/_4)*\log(^1/_4)]$
  $= 4* -[(^1/_4)*(-2)] = +2.0$
- Alignment Entropy = 0 + 0.811 + 2.0 = +2.811

Multiple Alignment Induces Pairwise Alignments

Every multiple alignment induces pairwise alignments

```
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
```

Sum of Pairs Score(SP-Score)

- Consider pairwise alignment of sequences

  a_i and a_j

  imposed by a multiple alignment of *k* sequences
- Denote the score of this suboptimal (not necessarily optimal) pairwise alignment as

  $s^*(a_i, a_j)$
- Sum up the pairwise scores for a multiple alignment:

$$s(a_1,\dots,a_k) = \Sigma_{i,j}\, s^*(a_i, a_j)$$

Computing SP-Score

Aligning 4 sequences: 6 pairwise alignments

Given a_1, a_2, a_3, a_4:

$$
\begin{aligned}
s(a_1 \ldots a_4) = \Sigma s^*(a_i, a_j) = {} & s^*(a_1, a_2) + s^*(a_1, a_3) \\
& + s^*(a_1, a_4) + s^*(a_2, a_3) \\
& + s^*(a_2, a_4) + s^*(a_3, a_4)
\end{aligned}
$$

SP-Score: Example

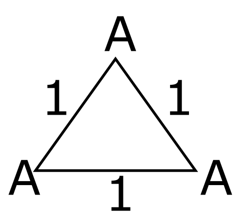
a_1 ATG–C–AAT

. A–G–CATAT

a_k ATCCCATTT

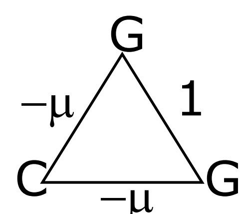
To calculate each column:

$$s(a_1 \ldots a_k) = \sum_{i,j} s^*(a_i, a_j) \longleftarrow \binom{n}{2} \text{Pairs of Sequences}$$



Column 1

Score=3



Column 3

Score = $1 - 2\mu$

Center Star Alignment Algorithm

- Gusfield proposed this algorithm, called **Center Star Alignment Algorithm . It can be proved that the** SP **values are less than twice the optimal value.** We sketch this algorithm now.

Center Star Alignment Algorithm

- We make the following assumptions:
  - $s(x,x)=0$, for all characters x.
  - Symmetric: $s(x,y)=s(y,x)$, for all characters x and y.
  - Triangle inequality: $s(x,y) \le s(x,z)+s(z,y)$, for all characters x, y and z.
- We have used the symbol $D(S_1,S_2)$ to denote the ***edit distance*** or ***minimum global alignment distance*** of S_1 and S_2.

Algorithm

The input is a set Γ of *k* strings.

1. First find $S_1 \varepsilon\, \Gamma$ that minimizes . This can be done by running the dynamic programming algorithm on each of the $\binom{k}{2}$ pairs of sequences in Γ.
   - Note this *S1* is not necessarily the first string specified in the input set Γ. Call the remaining sequences in Γ to be $S_2, S_3, \ldots, S_k$.
2. Now add these strings $S_2, S_3, \ldots, S_k$ one at a time to a multiple alignment that so far has only one sequence viz. S_1. Suppose $(S_1, S_{2,} \ldots\ldots\ldots S_{i-1})$ are already aligned as

$$(S_1^{'}, S_2^{'} \ldots\ldots\ldots S_{i-1}^{'})$$

3. To add S_i, run the dynamic programming algorithm again on S_1' and S_i to produce S_1'' and S_i'.
4. Then adjust $S_2^{'}.........S_{i-1}^{'}$ by adding spaces to those columns where spaces were added to get S_1'' from S_1'.
5. Replace S_1' by S_1''.

Example

- Γ=(*AGTGC, ATC, ATTC, ATC, AGC*)
- **Step1**. S_1 is *ATC* (any one of them) since the edit distance between *ATC* and *ATC* is zero.
  - For all other pair the edit distance is more than 0. Call the remaining set S_2=*ATTC*, S_3=*ATC*, S_4=*AGAGC* and S_5=*AGC*.
- **Step2** and **3**: Add *S2=ATTC*. The alignment between *S1'* and *S2* is:

S1''= A T -- C

S2' = A T T C

- Step4 and 5: We only have one S_1' which is now replaced by $S_1'' = \; A \; T \; \text{--} \; C$.
  - To add *ATC* , the new alignment is

$$S_1'' = \quad A \;\; T \;\; \text{--} \;\; C$$

$$S_3' = \quad A \;\; T \;\; \text{--} \; C$$

- Since no extra space has been inserted in S_1'', we don't have to do anything. So the alignment at this point look like.

$$\begin{array}{cccc} A & T & \text{--} & C \\ A & T & T & C \\ A & T & \text{--} & C \end{array}$$

- Next we add *S4=AGTGC.* The alignment is now

$$\begin{array}{l} A - T - C \\ A\ G\ T\ G C \end{array}$$

- Now, we have introduced a '–' in the second column of $S_1' = S_1''$. So the new multiple alignment have to be "adjusted" giving

$$\begin{array}{l} A - T - C \\ A -- T\ T\ C \\ A - T - C \\ A\ G\ T\ G\ C \end{array}$$

- Finally, we have to add S_5=*AGC.* Since the latest S_1'= S_1''= *A – T – C, S_5=AGC* can be aligned in two different ways by putting *G* aligned with any one of the spaces for S_1'.
- Thus, one of the solutions is

A – T – C

A --T T C

A – T – C

A G T G C

A G -- -- C

Time Complexity

- Theorem:
  - The algorithm just described above has a time complexity ***$O(k^2n^2)$,*** where *k* is the number of sequences and each sequence has a maximum length of *n*.
  - It can be proved that the total SP cost of the solution obtained by the above algorithm is not worse than the twice the optimal cost.

Multiple Alignment: History

1975 Sankoff

Formulated multiple alignment problem and gave dynamic programming solution

1988 Carrillo-Lipman

Branch and Bound approach for MSA

1990 Feng-Doolittle

Progressive alignment

1994 Thompson-Higgins-Gibson-ClustalW

Most popular multiple alignment program

1998 Morgenstern et al.-DIALIGN

Segment-based multiple alignment

2000 Notredame-Higgins-Heringa-T-coffee

Using the library of pairwise alignments

2004 MUSCLE