

7.93 – Lecture #3

Pairwise Sequence Comparisons

ARDFSHGLLENKLLGCDSMRWE
GRDYKMALLEQWILGCD-MRWD

Outline

- Definitions of related sequences
- Concepts and types of alignments – the good, the bad, and the ugly
- Dot matrix alignments
- Computational efficiency
- Recursion and dynamic programming
- New additions 1 – Substitution matrices:
PAM, BLOSUM, Gonnet

Outline (cont)

- New additions 2 – Gaps
- Applied dynamic programming: global alignments: Needleman-Wunsch
- Applied dynamic programming: local alignments – Smith-Waterman
- Basic statistics of sequence alignments

Why align sequences?

- Functional predictions based on identifying homologues.

Assumes:

conservation of
sequence



conservation of
function

BUT: Function carried out at level of proteins, i.e.

3-D structure

Sequence conservation carried out at level of DNA


1-D sequence

Implicit Assumption of Evolution in Models of Sequence Homology

Assume:

Sequence conservation  Structure conservation

Note that the converse is NOT necessarily true!

Structure conservation  Sequence conservation

Definitions

- Homologue:

a relatively non-specific term (meaningless?) that conveys the idea that two sequences are somehow related

- Orthologue:

Ortho = (*greek*) straight....implies direct descent, 1 ancestor

Vav human



Vav bovine

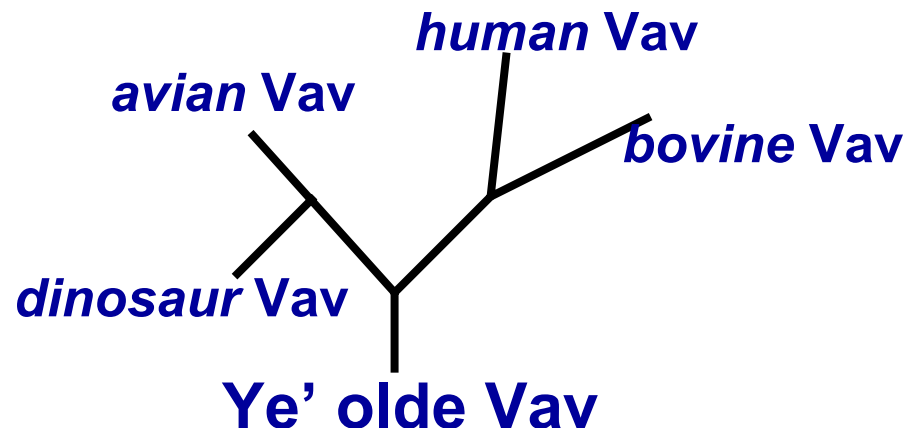


Vav avian



Vav elegans

-or-

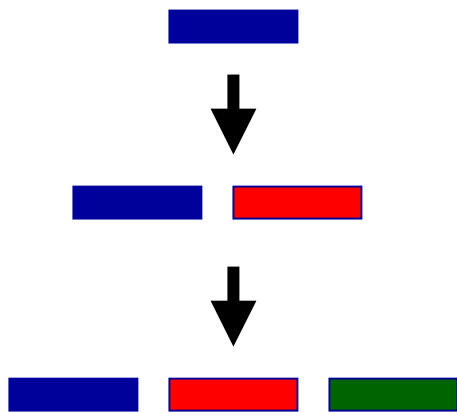


Definitions

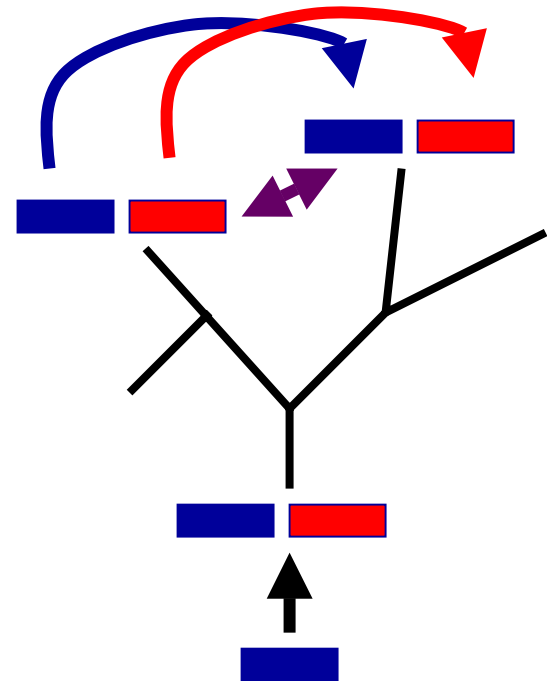
- Paralogue:

Para = (*greek*) along side of....implies some type of gene duplication event

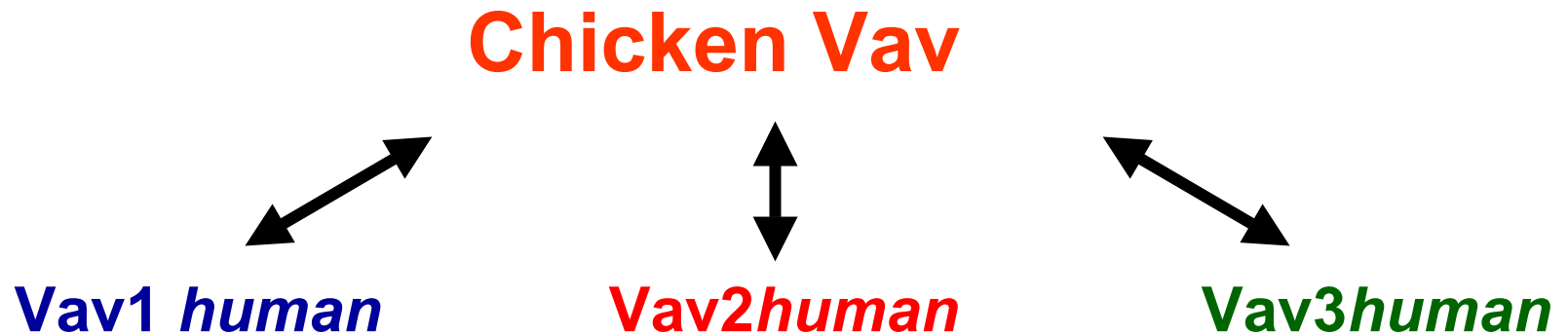
Vav1 human ↔ *Vav2human* ↔ *Vav3human*



-or-



Complex problem



Orthologue or Parologue?

Need to know evolutionary relationships

Can try and get these from alignments as well...

Alignments- Good Bad and Ugly

HgbA-human

GSAQVKGHGKKVADALTNVAHAVDDMPNALSALSDLHAHKL
:+ +::+:::++::+ :+++++::+:++ +++++:::++::+ :::
GNPKVKAHGKKVLGAFSDGLAHLNLIKGTFTLSELHCDKL

HgbB-human

Alignments- Good Bad and Ugly

SPURIOUS ALIGNMENT

HgbA-human

GSAQVKGHGKKVADALTNVAHAVDDMPNALSALSD----LHAHKL
::+ ++: + ++::: ++ :+ :+ : ++::+
GSGYLVGDSLTFVDLL--VAQHTADLLAANAALLDEFPOFKAHQE

Nematode glutathione S-transferase

HgbA-human

GSAQVKGHGKKVADALTNVAHV---D--DMPNALSALSDLHAHKL
++ ++++:+ ::+ ++ +:++ + +: :+ ++: +
NNPELQAHAGKVFKLVEAAIQLQVTGVVVTDATLKNLGSVHVSKE

Leghemoglobin, yellow lupin

Alignments

- **Types:**

- **Local**
- **Global**
- **Ungapped**
- **Gapped (2 types- linear, affine)**

- **Methods:**

- **Dot matrix**
- **Dynamic Programming**
- **Word, k-tup**

Example – self alignment

	G	F	D	S	F	K	R	L	E	F	S	E	V
G	1	0	0	0	0	0	0	0	0	0	0	0	0
F		1	0	0	1	0	0	0	0	1	0	0	0
D			1	0	0	0	0	0	0	0	0	0	0
S				1	0	0	0	0	0	0	1	0	0
F					1	0	0	0	0	1	0	0	0
K						1	0	0	0	0	0	0	0
R							1	0	0	0	0	0	0
L								1	0	0	0	0	0
E									1	0	0	1	0
F										1	0	0	0
S											1	0	0

The image shows a self-alignment matrix for the word "SEV". The matrix is a 13x13 grid with rows and columns labeled with the letters G, F, D, S, F, K, R, L, E, F, S, E, V. A red diagonal line runs from the top-left cell (G, G) to the bottom-right cell (S, S). Several cells containing the value '1' are circled in red: (F, F), (F, S), (S, S), (F, F), (E, E), and (S, S). The matrix is symmetric, with 1s on the diagonal and 0s elsewhere.

Example – self alignment....with sliding window

	G	F	D	S	F	K	R	L	E	F	S	E	V
G	1	0	0	0	0	0	0	0	0	0	0	0	0
F		1	0	0	1	0	0	0	0	1	0	0	0
D			1	0	0	0	0	0	0	0	0	0	0
S				1	0	0	0	0	0	0	1	0	0
F					1	0	0	0	0	1	0	0	0
K						1	0	0	0	0	0	0	0
R							1	0	0	0	0	0	0
L								1	0	0	0	0	0
E									1	0	0	1	0
F										1	0	0	0
S											1	0	0

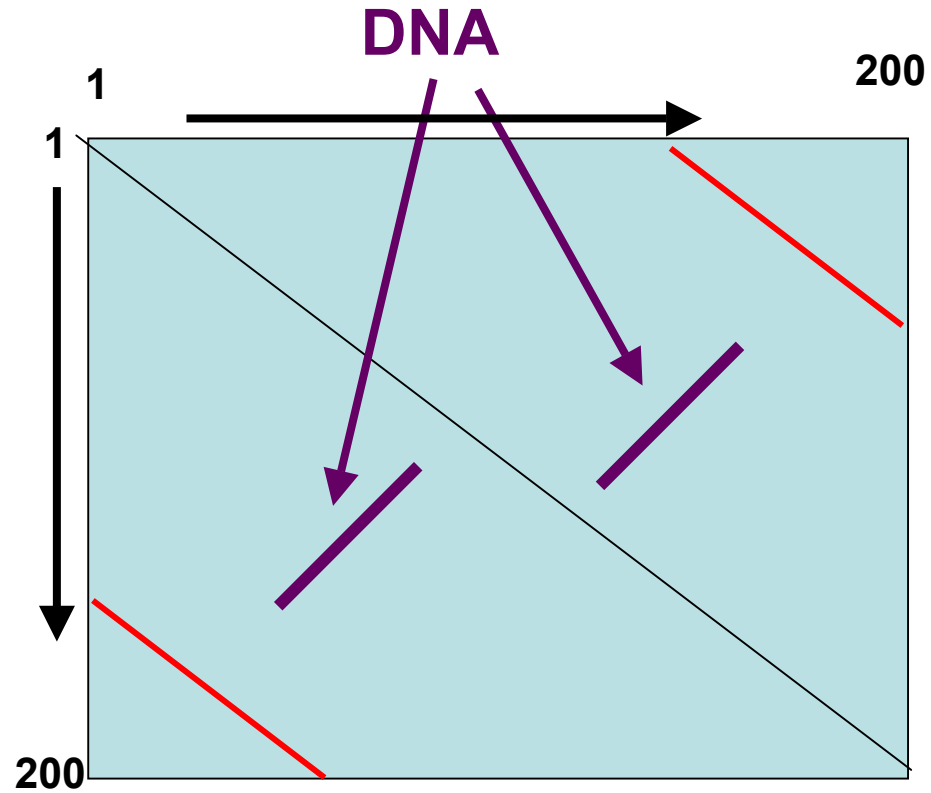
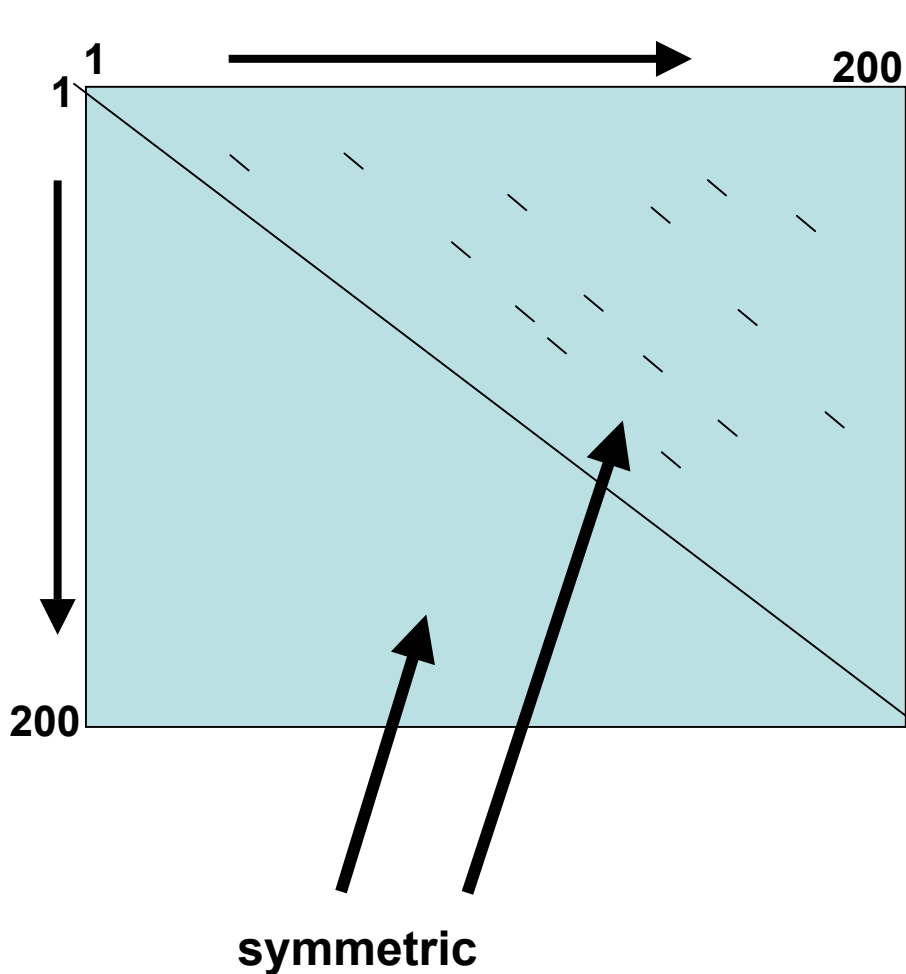
Example – self alignment

	G	F	D	S	F	K	R	L	E	F	S	E	V
G	1	0	0	0	0	0	0	0	0	0	0	0	0
F		1	0	0	1	0	0	0	0	0	0	0	0
D			1	0	0	0	0	0	0	0	0	0	0
S				1	0	0	0	0	0	0	0	0	0
F					1	0	0	0	0	0	0	0	0
K						1	0	0	0	0	0	0	0
R							1	0	0	0	0	0	0
L								1	0	0	0	0	0
E									1	0	0	0	0
F										1	0	0	0
S											1	0	0

The image shows a self-alignment matrix for the word "SEV". The matrix is a 13x13 grid where the top row and left column are labeled with the letters G, F, D, S, F, K, R, L, E, F, S, E, V. The diagonal elements are all 1, and all other elements are 0. A red diagonal line runs from the top-left corner (G, G) to the bottom-right corner (S, S). A purple zig-zag path starts at (E, E) and moves down-right to (F, F), then up-left to (S, S), then down-right to (E, E), then up-left to (F, F), then down-right to (S, S), then up-left to (E, E), then down-right to (F, F), then up-left to (S, S), then down-right to (E, E), then up-left to (F, F), then down-right to (S, S). Red circles highlight the zeros at (F, E), (S, F), (F, S), and (E, S).

Simple alignments

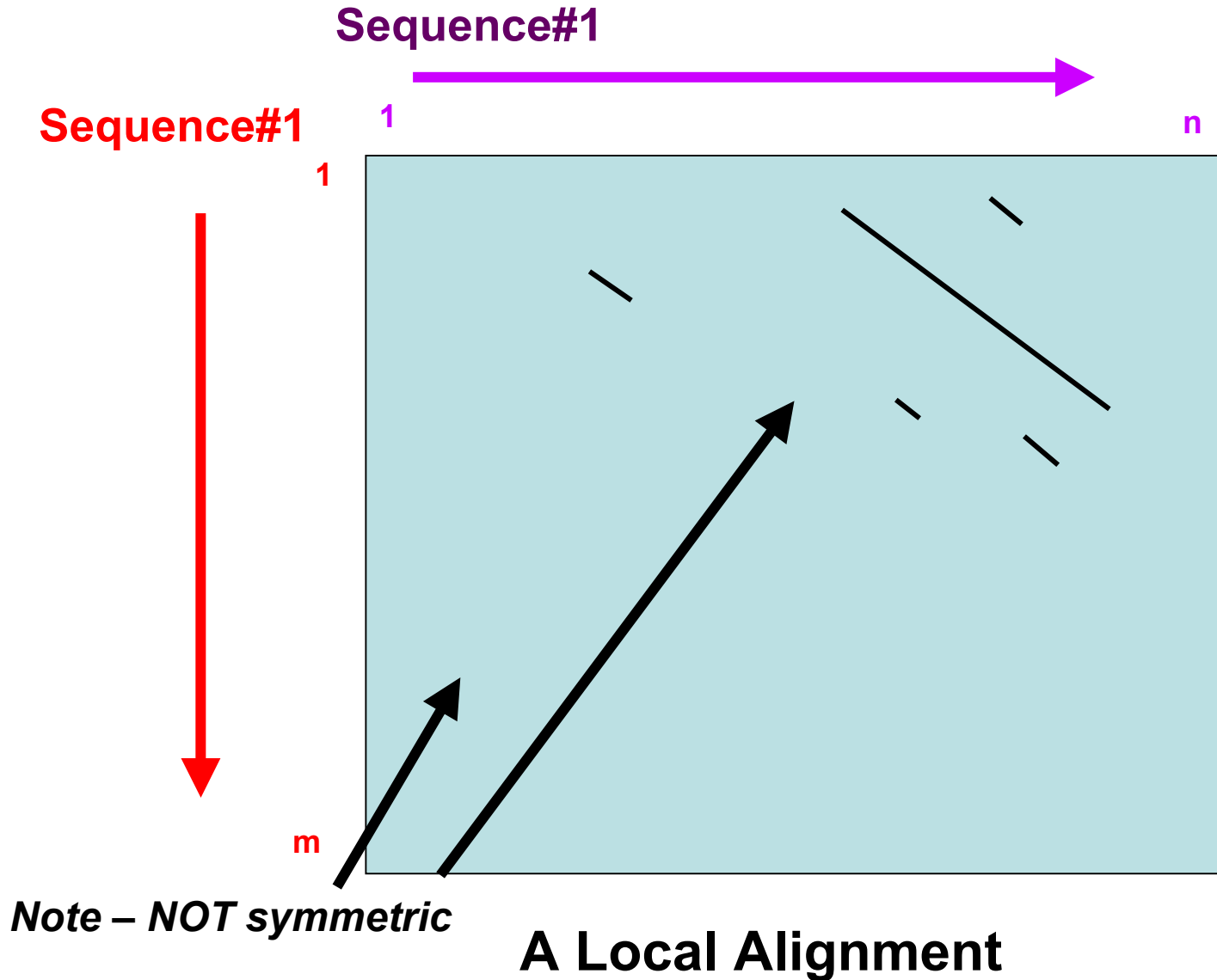
Self alignment – Dot Matrix



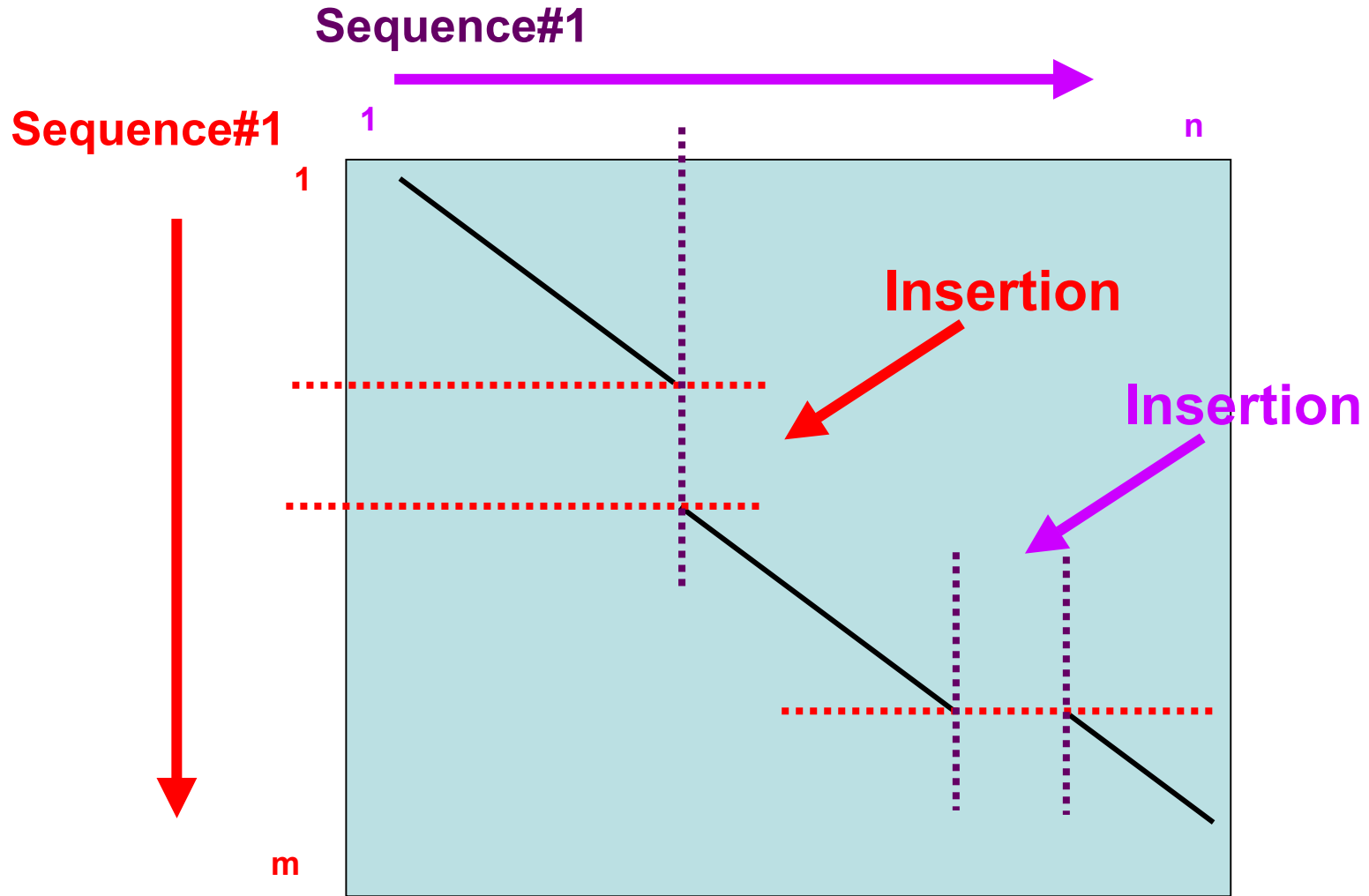
Now align two different sequences

- * **Consider other similarity matrices besides identity....**
 - ***Chemical similarity*** – binary decision
 - ***Amino acid conservation*** in aligned protein families – min. similarity score (+/- window)
 - ***Average*** of multiple scoring systems

Dot Matrix Alignments



Dot Matrix Alignments



A Global Alignment

General Rules for Dot Matrices

- Advantage – let's your eyes/brain do the work – VERY EFFICIENT!!!!
- DNA comparisons – long windows and high stringencies (11/7, 15/11)
- Proteins – short windows and stringencies (1/1) EXCEPT in looking for domains – then longer windows and smaller stringencies (15/5).
- *Note – we can use these types of self-aligned matrices for more than just sequence comparisons...i.e. distance between side C α atoms in 3d-structures etc....maybe later in the course!*

Computational Efficiency

Measure efficiency in cpu run time and memory

$O()$ = “big-oh” notation

Both scale as size of the problem, measured in number of units, n , in the problem, i.e. run time is $f(n)$.

Analyse the asymptotic worst-case running time....

Sometimes just do the experiment and measure it....

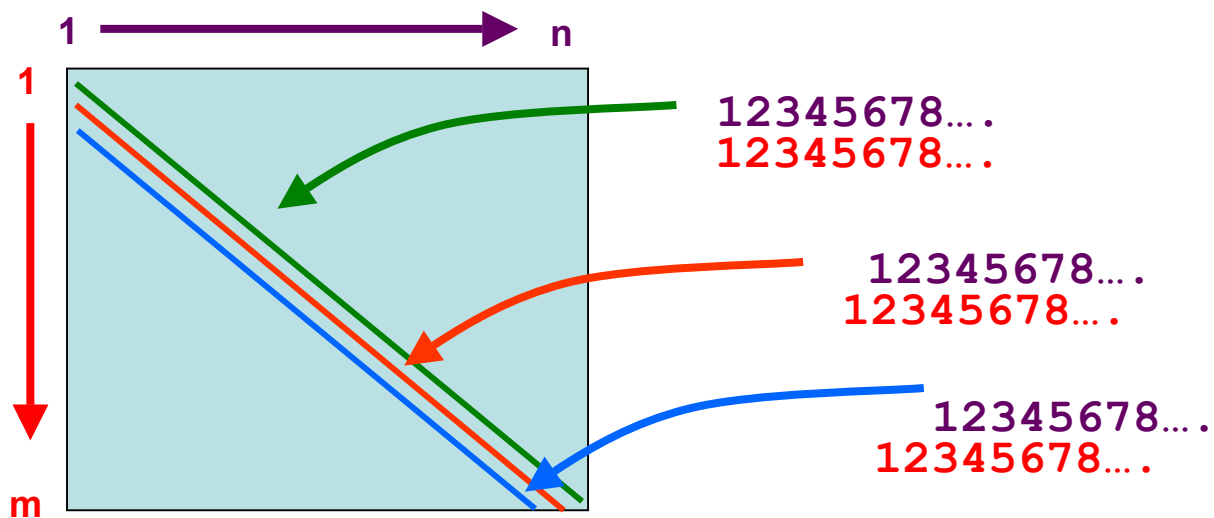
If problem scales as square of the number of units in the problem

$O(n^2)$ (=order n-squared)

Examples

$O(n^k)$ is “polynomial time” as long as
 $k \leq 3$ tractable

Consider our un-gapped dot matrix
Global alignment:



.....essentially an $O(mn)$ problem

O.K. Examples

$O(n)$ better than $O(n \log(n))$, better than $O(n^2)$, better than $O(n^3)$

Terrible Examples

$O(k^n)$ = exponential time....horrible!!!!

**NP problems- no known polynomial time
Solutions = non-deterministic polynomial
Problems.**

Recursion and Dynamic Programming

Aligning two protein sequences without gaps – roughly an $O(mn)$ problem.
With gaps – becomes computationally astronomical, and cannot be done by direct comparison methods. ($= 2^{2L}/\sqrt{2\pi L}$; L =sequence length)

Alternative is to compare all possible pairs of characters (matches and mismatches, and also take gaps into account as well, while keeping the number of comparisons manageable. The approach is called dynamic programming. Mathematically proven to produce optimal alignment

Need a substitution or similarity matrix and some way to account for gaps.

Example of how to score an alignment: Write down two sequences:

sequence#1	V	D	S	-	C	Y
sequence#2	V	E	S	L	C	Y
Score from sub. Matrix	4	2	4	-11	9	7

Score = Σ (AA pair scores) – gap penalty = 15

**Scoring system should: favor matching identical or related amino acids
Penalize for poor matches and for gaps.**

**To get a good scoring system need to know: how often a particular amino acid
Pair is found in related proteins compared with its occurrence by chance. This
Is the information contained in the substitution matrix
.....and when a gap would be a better choice**

Deriving realistic substitution matrices:

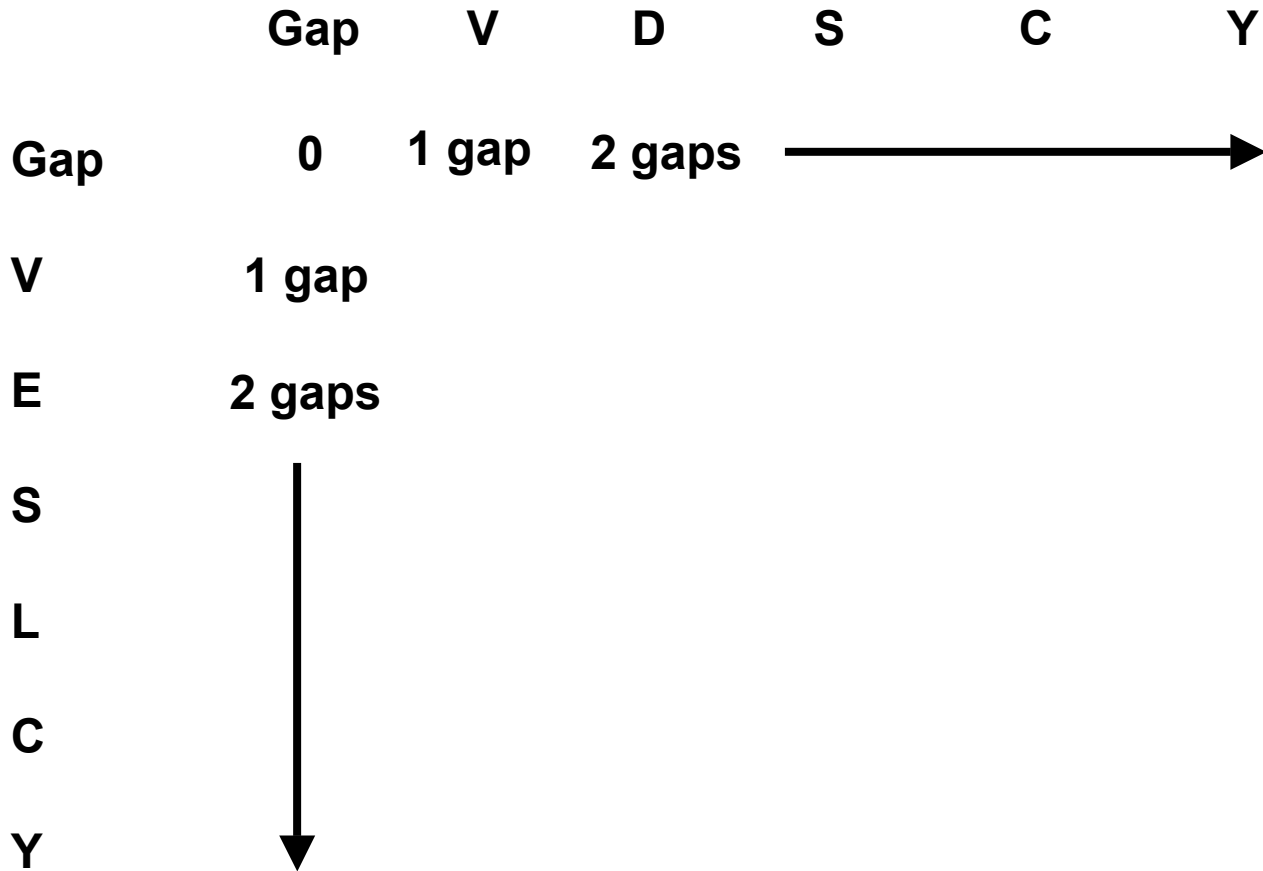
**First need to know frequency of one amino acid substituting for another
In related proteins [=P(ab)] c/w the chance that substituting one for the other
occurred by chance, based on the relative frequencies of each amino acid
in proteins, q(a) and q(b). Call this the “odds ratio”: $P(ab)/q(a)q(b)$**

**If we do this for all positions in an alignment, then the total probability will
be the product of the odds ratios at each position....but multiplication is
computationally expensive....so....take the **log (odds ratio)** and add them instead.**

**Matrices like PAM and BLOSUM matrices are derived from these log odds ratios
And contain positive and negative numbers reflecting likelihood of amino
Acid substitutions in related proteins.**

To do Dynamic Programming:

First write one sequence across the top, and one down along the side



Note – linear gap penalty: $\gamma(n)=nA$, where A =gap penalty

To do Dynamic Programming:

First write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5
j =		Gap	V	D	S	C	Y
0	Gap	0	-8	-16	-24	-32	-40
1	V	-8	S_{ij}				
2	E	-16					
3	S	-24					
4	L	-32					
5	C	-40					
6	Y	-48					

So scoring S_{ij} requires that we know $S(i-1, j-1)$ and $S(i, j-1)$ and $S(i-1, j)$...
Therefore recursive. We use the solutions of smaller problems to solve larger ones.
AND we store how we got to the S_{ij} score, i.e. the intermediate solutions in a tabular matrix. Computer scientists call this dynamic programming, where “programming” means the matrix, not some kind of computer code.

To do Dynamic Programming:

First write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5
j =		Gap	V	D	S	C	Y
0	Gap	0	-8	-16	-24	-32	-40
1	V	-8	S_{ij}				
2	E	-16					
3	S	-24					
4	L	-32					
5	C	-40					
6	Y	-48					

Global alignments: Needleman-Wunsch-Sellers
 $O(n^2)$ using linear gap penalty

$$\mathbf{S}_{ij} = \max \text{ of: } \left\{ \begin{array}{l} \mathbf{S}_{i-1, j-1} + \sigma(\mathbf{x}_i, \mathbf{y}_j) \text{ (diagonal)} \\ \mathbf{S}_{i-1, j} - \mathbf{A} \text{ (from left to right)} \\ \mathbf{S}_{i, j-1} - \mathbf{A} \text{ (from top to bottom)} \end{array} \right.$$

To do Dynamic Programming:

First write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5
j =		Gap	V	D	S	C	Y
0	Gap	0	-8	-16	-24	-32	-40
1	V	-8	S_{ij}				
2	E	-16					
3	S	-24					
4	L	-32					
5	C	-40					
6	Y	-48					

Global alignments: Needleman-Wunsch-Sellers
 $O(n^2)$ using linear gap penalty

$$S_{ij} = \max \text{ of: } \left\{ \begin{array}{l} S_{i-1, j-1} + \sigma(x_i, y_j) \text{ (diagonal)} \\ S_{i-1, j} - A \text{ (from left to right)} \\ S_{i, j-1} - A \text{ (from top to bottom)} \end{array} \right.$$

To do Dynamic Programming:

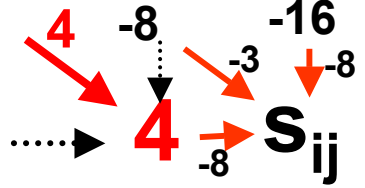
First write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5
j =	Gap		V	D	S	C	Y
0	Gap	0	-8	-16	-24	-32	-40
1	V	-8	4				
2	E	-16					
3	S	-24					
4	L	-32					
5	C	-40					
6	Y	-48					

To do Dynamic Programming:

First write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5
j =		Gap	V	D	S	C	Y
0	Gap	0	-8	-16	-24	-32	-40
1	V	-8	4	-8	-16	-24	-32
2	E	-16	-8	-16	-24	-32	-40
3	S	-24	-16	-24	-32	-40	-48
4	L	-32	-24	-32	-40	-48	-56
5	C	-40	-32	-40	-48	-56	-64
6	Y	-48	-40	-48	-56	-64	-72



Global alignments: Needleman-Wunsch-Sellers
 $O(n^2)$ using linear gap penalty

$$S_{ij} = \max \text{ of: } \left\{ \begin{array}{l} S_{i-1, j-1} + \sigma(x_i, y_j) \text{ (diagonal)} \\ S_{i-1, j} - A \text{ (from left to right)} \\ S_{i, j-1} - A \text{ (from top to bottom)} \end{array} \right.$$

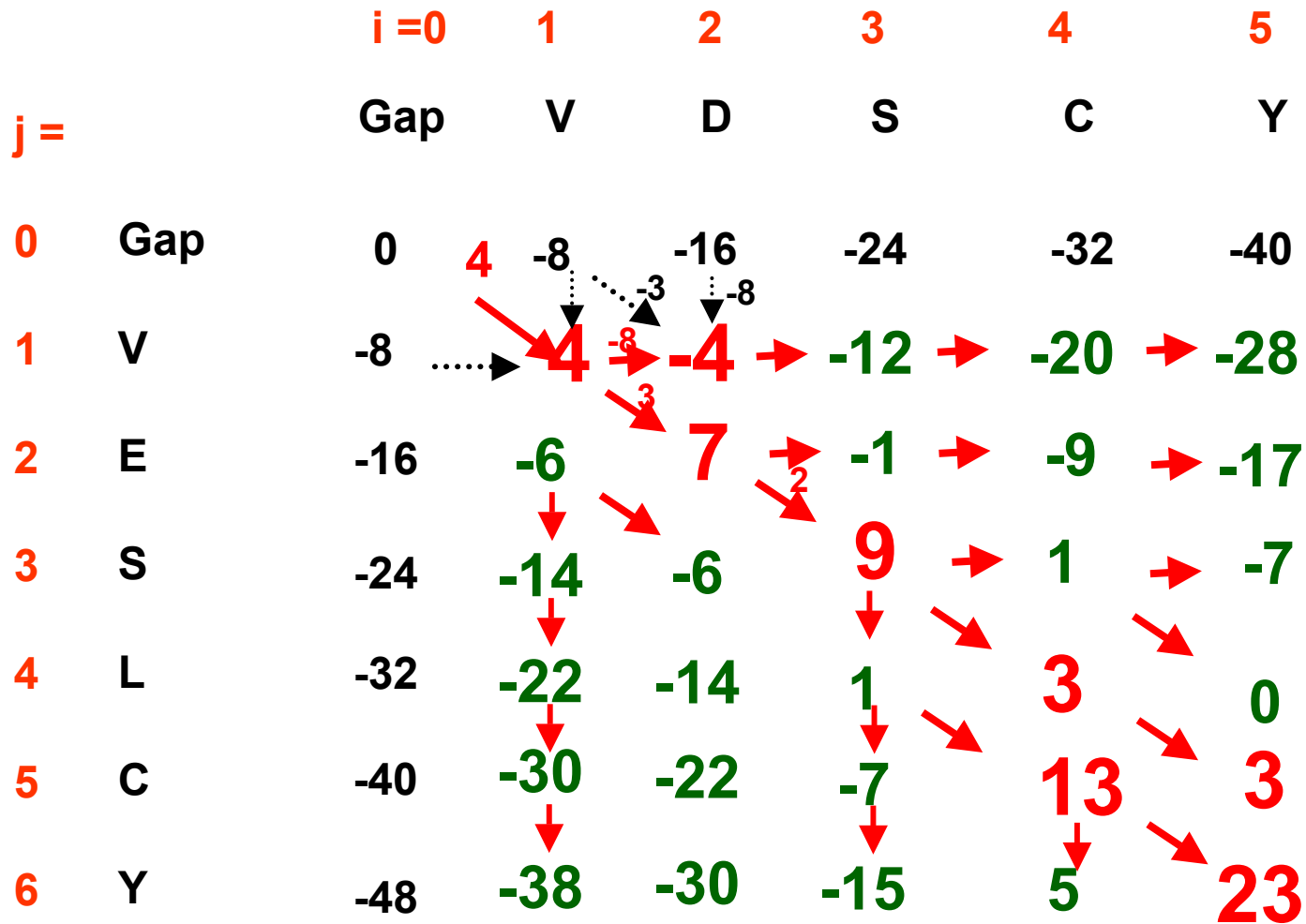
To do Dynamic Programming:

First write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5	
j =		Gap	V	D	S	C	Y	
0	Gap	0	4	-8	-16	-24	-32	-40
1	V	-8	4	-4				
2	E							
3	S							
4	L							
5	C							
6	Y							

To do Dynamic Programming:

First write one sequence across the top, and one down along the side



The Traceback:

After the alignment square is finished, start at the lower right and work backwards following the arrows to see how you got there...



The Traceback gives the alignment:

V D S - C Y
V E S L C Y



Local Alignment

- Temple Smith and Michael Waterman, 1981 – modified Needleman-Wunsch-Sellers

Local alignment is the best scoring alignment of a substring in sequence x to a substring in sequence y.

KEY ELEMENT IS NOT TO FORCE THE ALIGNMENT TO GO TO THE ENDS OF THE SEQUENCES.

For sequence x, residues 1, 2, 3.....N, can pick up to $\sim N^2$ substrings, i.e. start point $a= 1,2....N$ and end point $b= 1, 2....n$. Same for sequence y, $\sim M^2$ substrings. For any two substrings, we have the old $O(mn)$ alignment problem, so the total number of possible alignments is $\sim N^2M^2(NM)=O(M^3N^3)$ - UGLY!!!! Solveable in polynomial time, but a big polynomial!!!

Local Alignment

- Again, dynamic programming comes to the rescue!

Same basic scheme for dynamic programming as before except....

Similarity matrix MUST include negative values for mismatches

--AND--

******When the value calculated for a position in the scoring matrix is Negative, the value is set to zero.**

THIS TERMINATES THE ALIGNMENT

Smith-Waterman:

Write one sequence across the top, and one down along the side

		i = 0	1	2	3	4	5
j =	Gap	0	0	0	0	0	0
0	Gap	0	0	0	0	0	0
1	V	0	S_{ij}				
2	E	0					
3	S	0					
4	L	0					
5	C	0					
6	Y	0					

Local alignments: Smith-Waterman

$$S_{ij} = \max \text{ of: } \left\{ \begin{array}{l} S_{i-1, j-1} + \sigma(x_i, y_j) \text{ (diagonal)} \\ S_{i-1, j} - A \text{ (from left to right)} \\ S_{i, j-1} - A \text{ (from top to bottom)} \\ 0 \end{array} \right.$$

**Programs for Global and Local alignments:
Biology workbench**

<http://workbench.sdsc.edu/>

Bill Pearson's Web Page

<http://fasta.bioch.virginia.edu/>

NCBI, Expassy

Amino Acid Substitution Matrices

Margaret Dayhoff, 1978, PAM Matrices

Evolutionary model based on a small data set.

Assumes symmetry: $A \rightarrow B = B \rightarrow A$

Assumes amino acid substitutions observed over short periods of time can be extrapolated to long periods of time

71 groups of protein sequences, 85% similar

1572 amino acid changes.

Functional proteins \leftrightarrow "Accepted" mutations by natural selection

PAM1 matrix means probability of each amino acid changing into another is $\sim 1\%$ and probability of not changing is $\sim 99\%$