Measures of sequence similarity

Hamming distance:

Number of positions with mismatching characters.

Defined for two strings of equal length.

```
agtc
cgta
```

Levenshtein distance:

Minimum number of edit operations (delete, insert, change a single character) needed to change one sequence into another.

```
agtcc
cgctca
```

Dotplots

A pictorial representation of the similarity between two sequences.

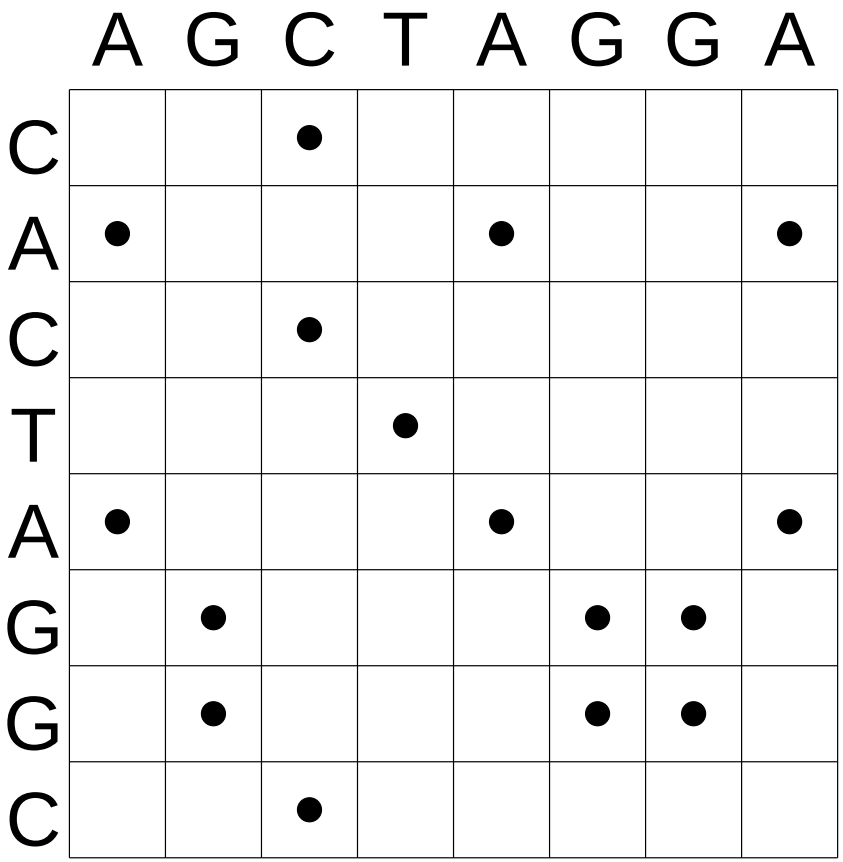
Compare a sequence with itself:

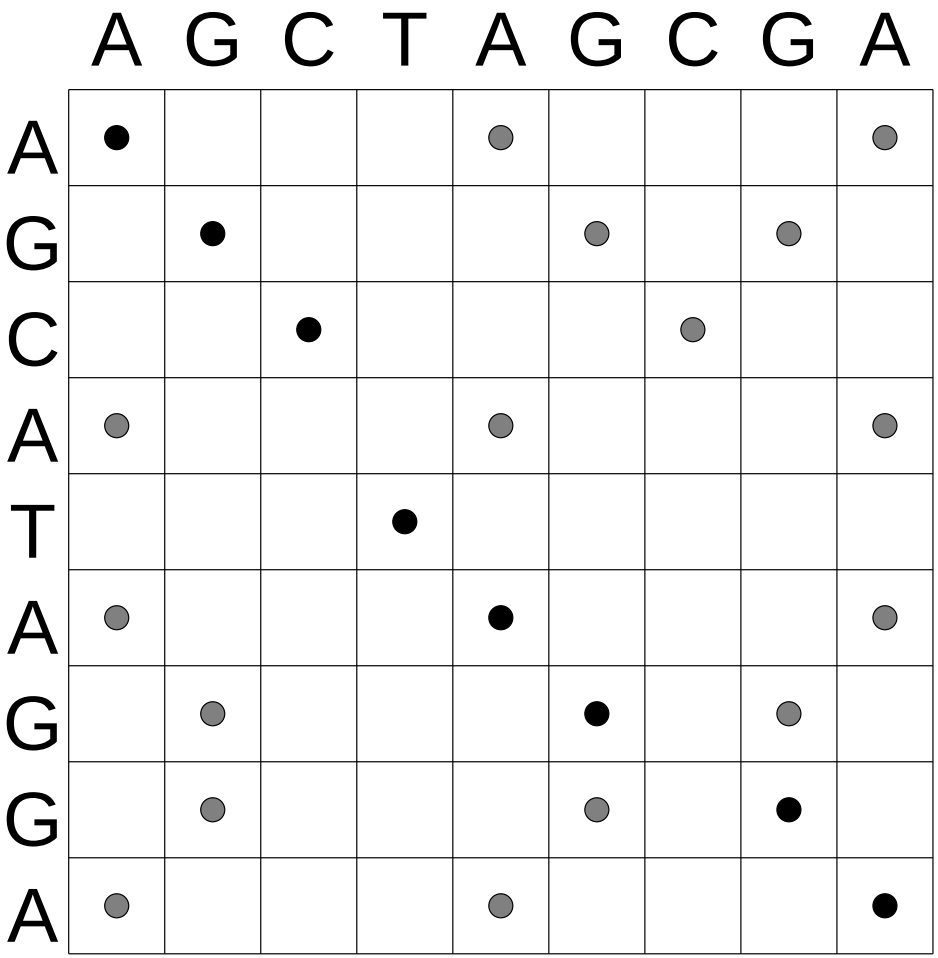
- Repeats
- Palindromic sequences

Compare two sequences:

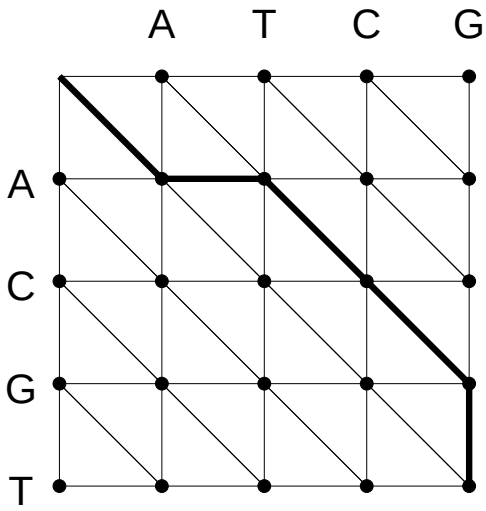
- Any path from upper left to lower right represents an alignment.
- Horizontal or vertical moves correspond to gaps in one of the sequences.
- Path with highest score corresponds to an optimal alignment.

Dotplots

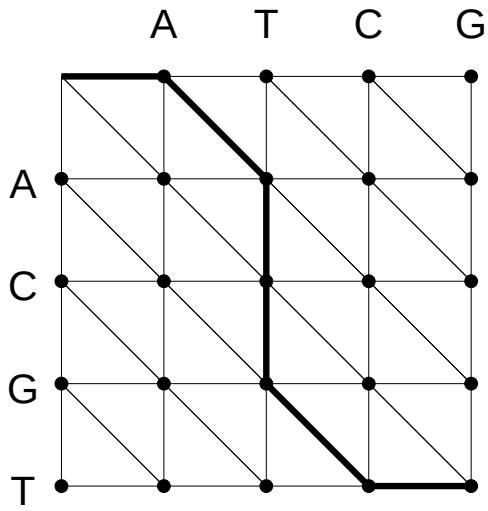




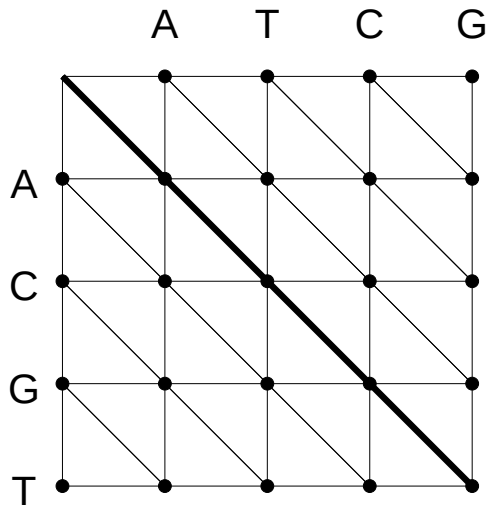
Each path represents an alignment




```
A-CGT
|  ||
ATCG-
```



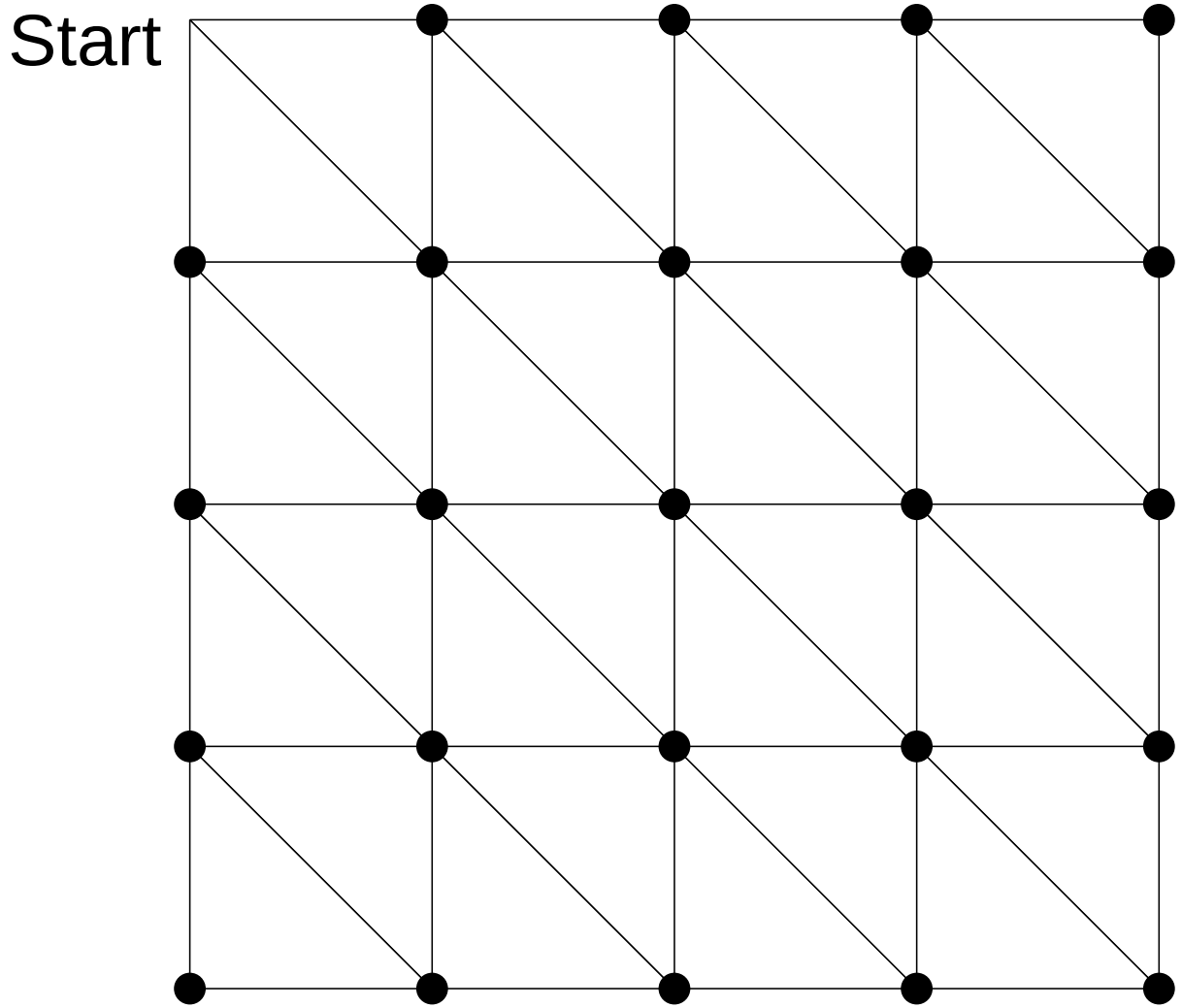

```
-ACGT-


AT--CG
```



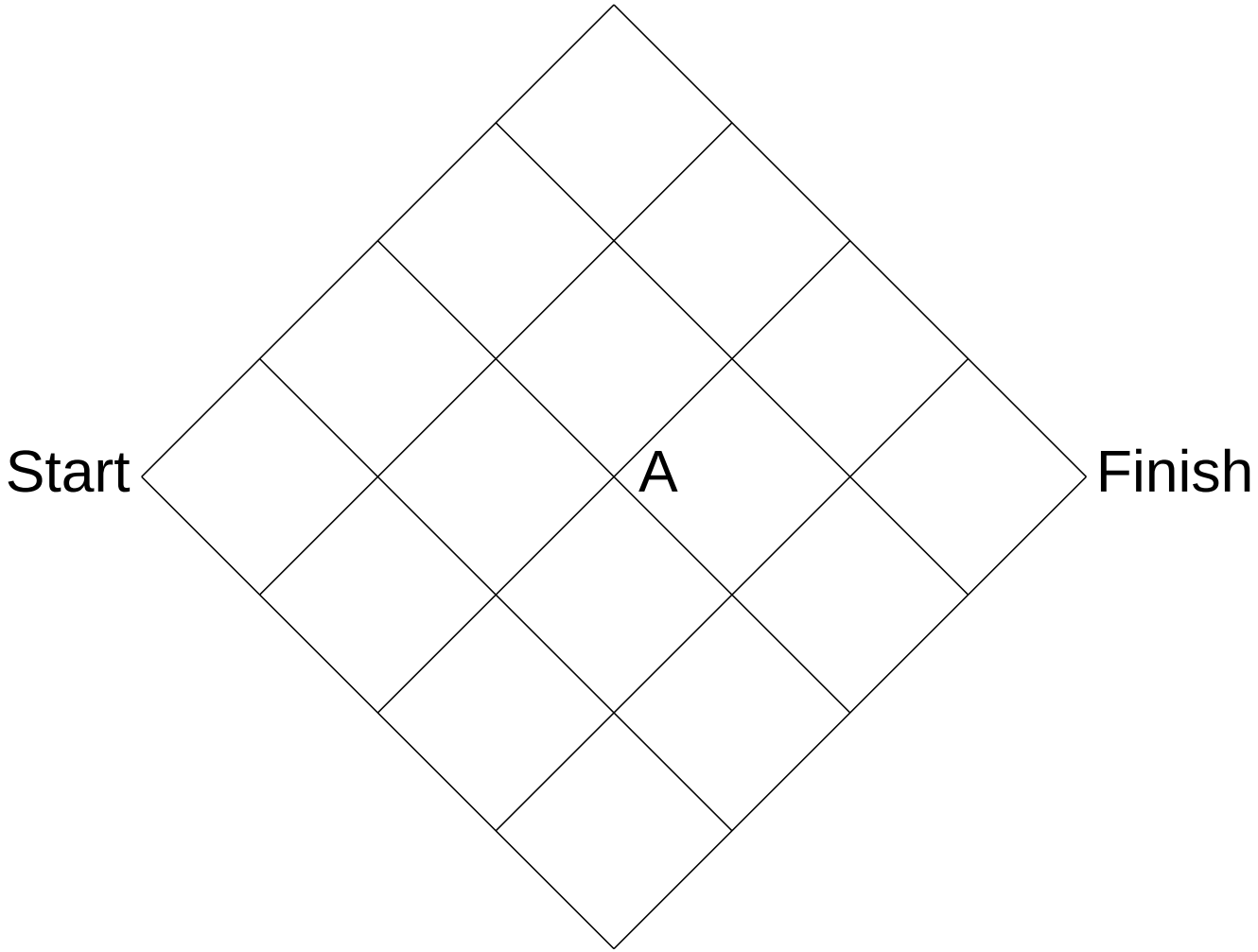

```
ACGT
|
ATCG
```

- Vertical steps add a gap to the horizontal sequence
- Horizontal steps add a gap to the vertical sequence

How many paths?



Do we have to enumerate all paths?



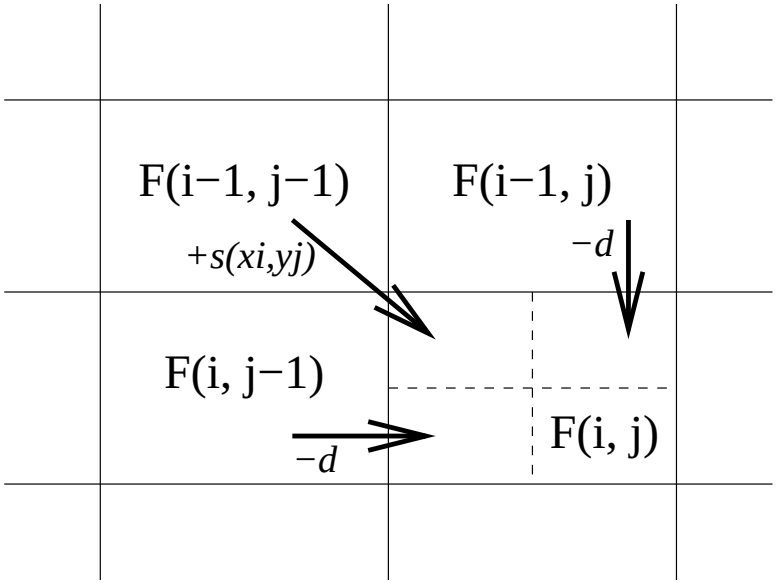
Pairwise global alignment (Needleman-Wunsch algorithm)

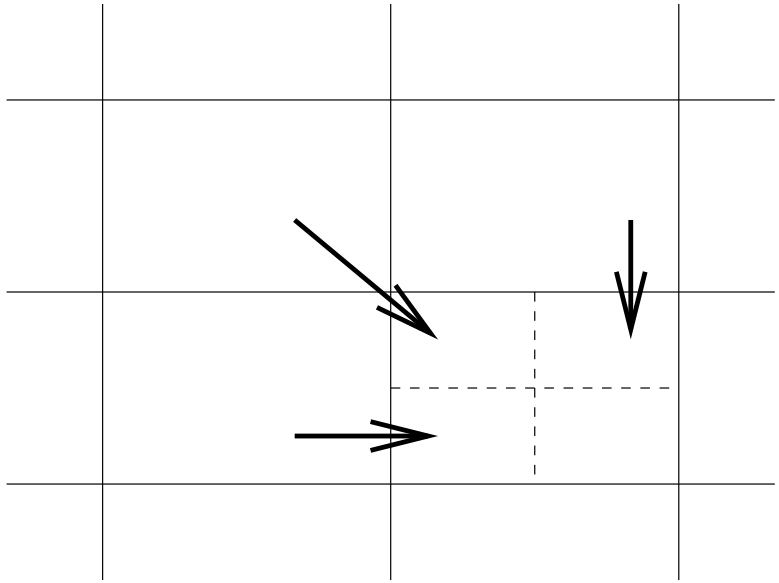
Rigorous algorithms use dynamic programming to find an optimal alignment.

- match score
- mismatch score
- gap penalty

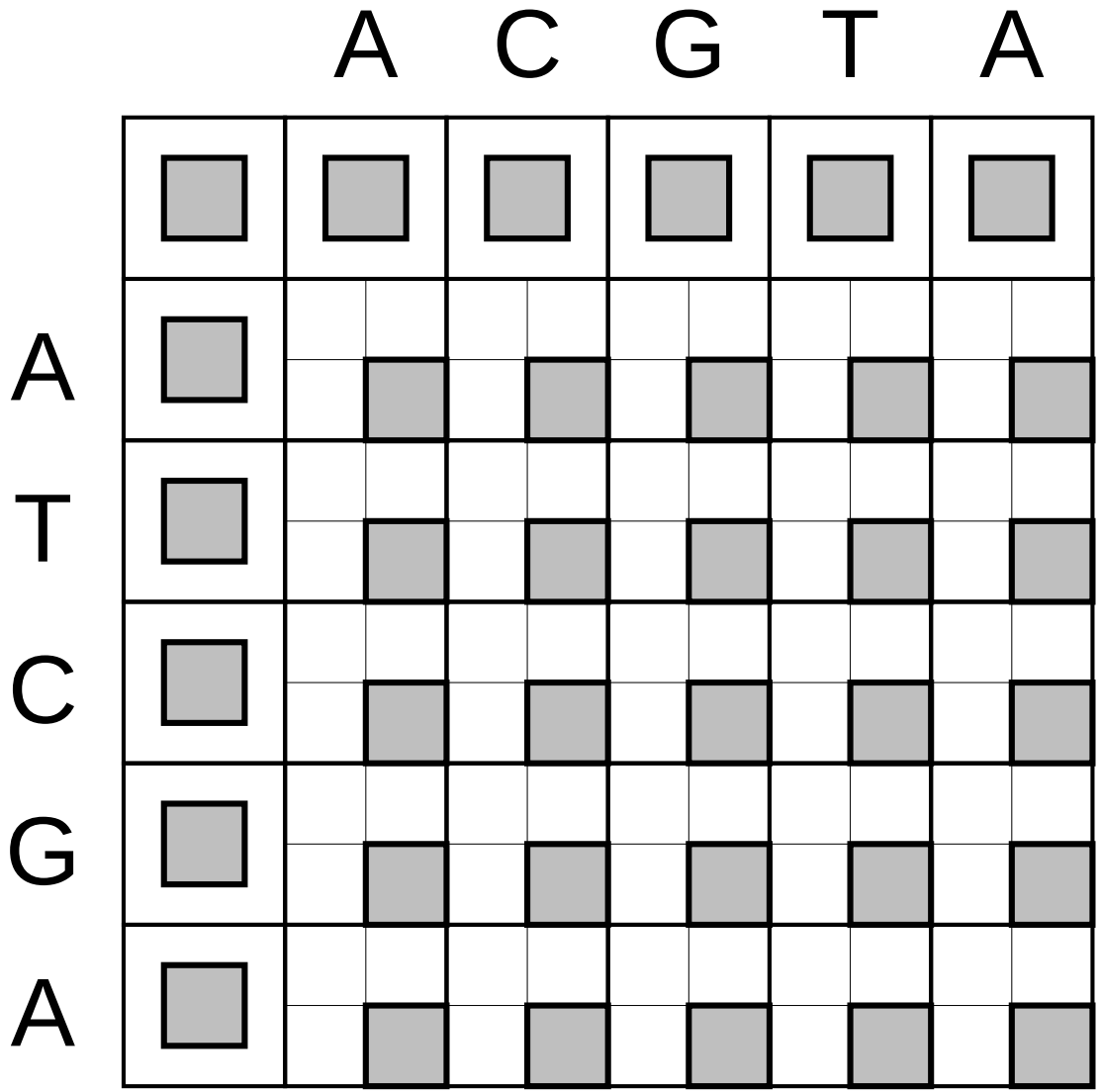
$$F(i,\, j) = \max \begin{cases} F(i-1,\, j-1) + s(x_{i,} y_j) \\ F(i-1,\, j) - d \\ F(i,\, j-1) - d \end{cases}$$

Dynamic programming





Score matrix



Percent identity

Having obtained an alignment, it is common to quantify the similarity between a pair of sequences by stating the percent identity.

```
-ACGATAG-CGAAACCAAAA
 ||| ||| |||    |
CAGC-TAGCCGATGTC----
```

Count the number of alignment positions with matching characters and divide by ... *what*?

- the length of the shortest sequence?
- the length of the alignment?
- the average length of the sequences?
- the number of non-gap positions?
- the number of equivalenced positions excluding overhangs?

Is the similarity significant, or could it be due to chance?

Even if two proteins are unrelated, we would expect some similarity simply by chance.

Is the alignment score significantly higher than random?

Align random permutations of the sequences, and find the mean and standard deviation of the resulting distribution.

The z-score reflects the significance of a global similarity score.

$$z\text{-}score = \frac{score - mean}{standard\ deviation}$$

Larger values imply greater significance.

Pairwise local alignment (Smith-Waterman algorithm)

Local similarities may be masked by long unrelated regions.

A minor modification to the global alignment algorithm.

- If the score for a subalignment becomes negative, set the score to zero.

$$F(i, j) = \max \begin{cases} 0 \\ F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

- Trace back from the position in the score matrix with the highest value.

- Stop at cell where score is zero.

More realistic similarity measures

Not all substitutions are equally likely.

- A transition between two purines (A, G) or between two pyrimidines (C, T/U) is more common than a purine-pyrimidine transversion.

- Replacement of one amino acid residue by another with similar size or physiochemical properties is more common than replacement by a dissimilar amino acid residue.

Insertion/deletion of N contiguous amino acid residues or nucleotides is more likely than N independent insertion/deletion events.

Thus, we should have different penalties for opening gap and for extending a gap.

Possible substitution matrices for DNA

	A	C	G	T
A	2	-1	-1	-1
C	-1	2	-1	-1
G	-1	-1	2	-1
T	-1	-1	-1	2

	A	C	G	T
A	2	-2	-1	-2
C	-2	2	-2	-1
G	-1	-2	2	-2
T	-2	-1	-2	2

Relative likelihood and alignment score

Match model (M):

Sequences assumed to be dependent. Residues x_i and y_i at position i in the alignment occur together with probability $p_{x_i y_i}$.

Random model (R):

Sequences assumed to be independent. Residues x_i and y_i at position i in the alignment occur together with probability $q_{x_i} q_{y_i}$.

We can score an alignment using the log of the relative likelihood:

$$S = \log\left(\frac{Pr(x, y|M)}{Pr(x, y|R)}\right) = \log \frac{p_{x_1 y_1} p_{x_2 y_2} \cdots p_{x_n y_n}}{q_{x_1} q_{y_1} q_{x_2} q_{y_2} \cdots q_{x_n} q_{y_n}}$$

$$= \sum_{i=1}^{n} \log\left(\frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}\right) = \sum_{i=1}^{n} s(x_i, y_i)$$

Percent accepted mutations

Expresses scores as log-odds values.

Score of mutation a-b is

$$\log \frac{\textit{observed a-b mutation rate}}{\textit{mutation rate expected from amino acid frequencies}}$$

Frequencies of substitutions of each pair of amino acid residues, extracted from alignments of closely related proteins.

PAM1 reflects the amount of evolutionary change that yields an average of one mutation per 100 amino acids.

Can assume that no position has changed more than once.

Correct for different amino acid abundances.

PAM substitution matrices

Extrapolate to a family of PAMk matrices by multiplying the PAM1 matrix by itself k times.

Different PAM matrices are more suitable when comparing sequences that have diverged by different amounts.

The PAM250 matrix is commonly used.

250 mutations per 100 amino acids.

Sequences still 20% identical:

- some positions change many times, while others don't change at all.
- some positions change one or more times, then revert back to the original amino acid residue.

PAM250

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

BLOSUM substitution matrices

Henikoff, S. and Henikoff, J.G. (1992) “Amino acid substitution matrices from protein blocks”, *Proc. Natl. Acad. Sci. USA*, 89:10915-10919

Based on large collection of multiple alignments of similar ungapped segments.

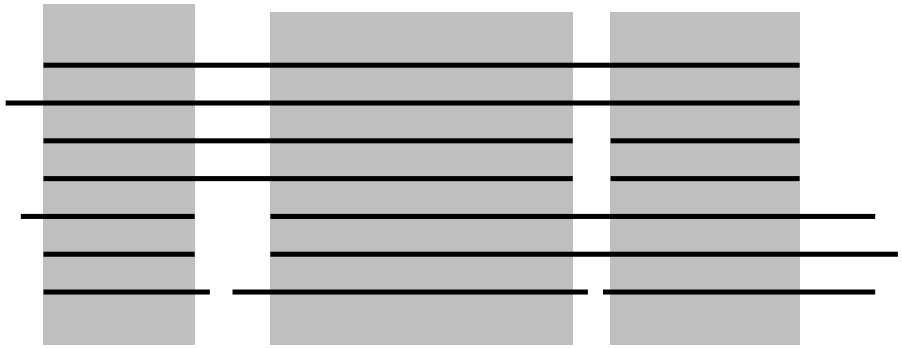
$$score_{ab} = \log \frac{observed\ relative\ frequency\ of\ aligned\ pairs\ ab}{expected\ probability\ of\ pair\ ab}$$

Pairs are only counted between segments that are more than x% identical.

Different values of x give different BLOSUM matrices.

The BLOSUM62 matrix is commonly used.

Deriving a frequency table from blocks



For each column of the block, count the number of matches and mismatches between pairs of sequences.

To illustrate the calculation, suppose we have a single block with one column, containing 9 A residues and 1 S residue.

In this case, there are 36 AA pairs and 9 AS or SA pairs.
That is, $f_{AA} = 36$ and $f_{AS} = 9$

Each column of each block in the blocks database will contribute to the observed frequency counts.

p_i is based on the proportion of residue type i in the whole blocks database.

Computing a logarithm of odds (Lod) matrix

Observed probability for each pair i,j is:

$$q_{ij} = \frac{f_{ij}}{\sum_{k=1}^{20} \sum_{l=1}^{k} f_{kl}}$$

Expected probability for each i,j pair is:

$$e_{ij} = \begin{cases} p_i p_j & \text{if } i = j \\ 2 p_i p_j & \text{if } i \neq j \end{cases}$$

Logarithm of odds is:

$$s_{ij} = \log_2(q_{ij}/e_{ij})$$

s_{ij} is multiplied by 2, then rounded to the nearest integer to give the BLOSUM score.

BLOSUM62

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4																			
R	-1	5																		
N	-2	0	6																	
D	-2	-2	1	6																
C	0	-3	-3	-3	9															
Q	-1	1	0	0	-3	5														
E	-1	0	0	2	-4	2	5													
G	0	-2	0	-1	-3	-2	-2	6												
H	-2	0	1	-1	-3	0	0	-2	8											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Which substitution matrix should I use?

Use a matrix that corresponds to the evolutionary distance between the proteins being compared (usually not known!).

Low PAM matrices are good for finding short, strong similarities.

High PAM matrices are good for finding long, weak similarities.

BLOSUM matrices have been found to perform better for detecting weak homologies than the extrapolated PAM matrices.

global_alignment.pl

```
#!/usr/bin/perl -w

$seq1 = "ATCGAT";
$seq2 = "ATACGT";

$MATCH    =  2;
$MISMATCH = -2;
$GAP      = -2;


# Initialise the score matrix and the trace matrix

$score_matrix[0][0] = 0;
$trace_matrix[0][0] = "STOP";

for ( $row = 1; $row <= length($seq1); $row++ ) {
    $score_matrix[$row][0] = $score_matrix[$row-1][0] + $GAP;
    $trace_matrix[$row][0] = "UP";
}

for ( $column = 1; $column <= length($seq2); $column++ ) {
    $score_matrix[0][$column] = $score_matrix[0][$column-1] + $GAP;
    $trace_matrix[0][$column] = "LEFT";
}


# Fill the score matrix and the trace matrix

for ( $row = 1; $row <= length($seq1); $row++ ) {
    for ( $column = 1; $column <= length($seq2); $column++ ) {
        if ( substr($seq1, $row-1, 1) eq substr($seq2, $column-1, 1) ) {
            $diagonal_score = $score_matrix[$row-1][$column-1] + $MATCH;
        } else {
            $diagonal_score = $score_matrix[$row-1][$column-1] + $MISMATCH;
        }
        $left_score = $score_matrix[$row][$column-1] + $GAP;
        $up_score   = $score_matrix[$row-1][$column] + $GAP;

        if ( ( $diagonal_score >= $left_score ) &&
             ( $diagonal_score >= $up_score ) ) {
            $score_matrix[$row][$column] = $diagonal_score;
            $trace_matrix[$row][$column] = "DIAGONAL";
        } elsif ( $left_score >= $up_score ) {
            $score_matrix[$row][$column] = $left_score;
            $trace_matrix[$row][$column] = "LEFT";
        } else {
            $score_matrix[$row][$column] = $up_score;
            $trace_matrix[$row][$column] = "UP";
        }
    }
}


# Print the score matrix

for ( $row = 0; $row <= length($seq1); $row++ ) {
    for ( $column = 0; $column <= length($seq2); $column++ ) {
        print $score_matrix[$row][$column] . " ";
    }
    print "\n";
}


# Trace back from the bottom-right cell

$aligned1 = "";
$aligned2 = "";

$row = length($seq1);
$column = length($seq2);

while ( $trace_matrix[$row][$column] ne "STOP" ) {
    if ( $trace_matrix[$row][$column] eq "DIAGONAL" ) {
        $aligned1 = substr($seq1, $row-1, 1) . $aligned1;
        $aligned2 = substr($seq2, $column-1, 1) . $aligned2;
        $row--;
        $column--;
    } elsif ( $trace_matrix[$row][$column] eq "LEFT" ) {
        $aligned1 = "-" . $aligned1;
        $aligned2 = substr($seq2, $column-1, 1) . $aligned2;
        $column--;
    } elsif ( $trace_matrix[$row][$column] eq "UP" ) {
        $aligned1 = substr($seq1, $row-1, 1) . $aligned1;
        $aligned2 = "-" . $aligned2;
        $row--;
    }
}

print "$aligned1\n";
print "$aligned2\n";
```