2 Pairwise alignment

We will discuss:

1. Dot matrix method for comparing sequences
2. The probabilistic model of pairwise alignment
3. Global-, local-, repeat- and overlap alignment of two sequences using dynamic programming

Sources for this lecture:

- R. Durbin, S. Eddy, A. Krogh und G. Mitchison, Biological Sequence Analysis, Cambridge, 1998
- D.W. Mount. Bioinformatics: Sequences and Genome analysis, CSHL 2001.
- R. Giegerich. Sequence similarity and dynamic programming, 2002. `http://www.zbit.uni-tuebingen.de/biss2002/handouts/pdf/giegerich_part1.pdf`
- J. Setubal & J. Meidanis, Introduction to Computational Molecular Biology, 1997.

2.1 Importance of sequence alignment

The first fact of biological sequence analysis: In biomolecular sequences (DNA, RNA, or amino acid sequences), high sequence similarity usually implies significant functional or structural similarity.

"Duplication with modification": The vast majority of extant proteins are the result of a continuous series of genetic duplications and subsequent modifications. As a result, redundancy is a built-in characteristic of protein sequences, and we should not be surprised that so many new sequences resemble already known sequences. ... all of biology is based on enormous redundancy...

We didn't know it at the time, but we found everything in life is so similar, that the same genes work in flies are the ones that work in humans. (Eric Wieschaus, cowinner of the 1995 Nobel prize in medicine for work on the genetics of *Drosophia* development.)

Dan Gusfield, 1997, 212 ff

2.2 Sequence alignment

Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple-alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in both sequences.

Two sequences are aligned by writing them across a page in two rows. Identical or similar characters are placed in the same column, whereas non-identical characters are either placed in the same column as a mismatch or are opposite a gap in the other sequence.

```
Two strings:              →  Alignment:
I MISS MISSISSIPPI           I-MISSMISSISIPPI-
                             ||||||||||||||||
MY MISS IS A HIPPIE          MYMISS-ISAH-IPPIE
```

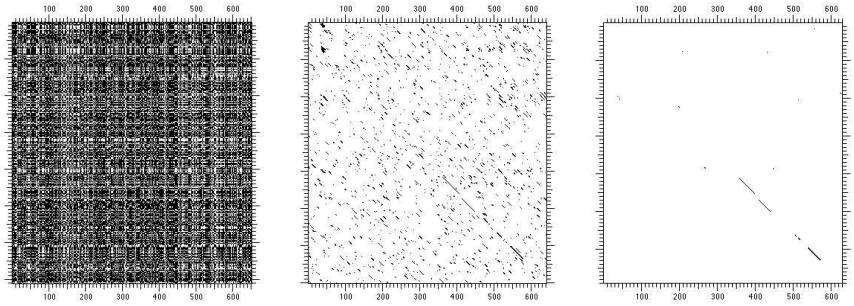
2.3 Dot matrix sequence comparison

A dot matrix analysis is primarily a method for comparing two sequences to look for a possible alignment of characters between the sequences.

A matrix relating two sequences is produced. A dot is placed at each cell for which the corresponding symbols match:

```
  IMISSMISSISSIPPI
M  .    .
Y
M  .    .
I .  .    .  .  .   .
S    ..   ..  ..
S    ..   ..  ..
I .  .    .  .  .   .
S    ..   ..  ..
A
H
I .  .    .  .  .   .
P                ..
P                ..
I .  .    .  .  .   .
E
```

Example: DNA sequences which encode the phage lambda and P22 repressor sequences:

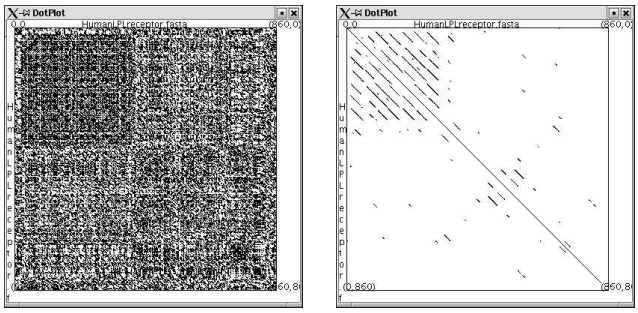


$w = 1, s = 1$ $w = 11,\ s = 7$ $w = 23,\ s = 15$

To obtain cleaner pictures, a *window size* w and a *stringency* s are used and a dot is only drawn at point (x, y) if in the next w positions at least s characters are equal.

2.4 Dot matrix repeat detection

Dot matrix analysis of human LDL receptor against itself (protein sequence):



$w = 1$, $s = 1$ $w = 23$, $s = 7$

This reveals many repeats in the first 300 positions.

2.5 Pairwise alignment

1. Alignment between very similar human alpha- and beta globins:

```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
           G+ +VK+HGKKV  A+++++AH+D++ +++++LS+LH  KL
HBB_HUMAN  GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL
```

2. Plausible alignment to leghaemoglobin from yellow lupin:

```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL
           ++ ++++H+ KV   + +A  ++          +L+ L+++H+ K
LGB2_LUPLU NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
```

3. A spurious high-scoring alignment of human alpha globin to a nematode glutathione *S*-transferase homologue:

```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSD----LHAHKL
           GS+ + G +   +D L  ++ H+ D+  A +AL D    ++AH+
F11G11.2   GSGYLVGDSLTFVDLLVAQHTADLL--AANAALLDEFPQFKAHQE
```

In (1), there are many positions at which the two corresponding residues are identical. Many others are functionally conserved, e.g. the D-E pairs, both negatively charged amino acids.

In (2), we also see a biologically meaningful alignment, as it is known that the two proteins are evolutionarily related, have the same 3D structure and both have the same function. However, there are many fewer identities and gaps have been introduced in the sequences.

In (3), we see an alignment with a similar number of identities or conservative changes. However, this is a spurious alignment between two proteins that have completely different structure and function.

The goal is to use similarity to uncover *homology*, while avoiding *homoplasy*.

2.6 The scoring model

When comparing two biological sequences, we want to determine whether and how they diverged from a common ancestor by a process of mutation and selection.

The basic mutational processes are *substitutions*, *insertions* and *deletions*. The latter two give rise to *gaps*.

The total score assigned to an alignment is the sum of terms for each aligned pair of residues, plus terms for each gap.

In a probabilistic interpretation, this will correspond to the the logarithm of the relative likelihood that the sequences are related, compared to being unrelated.

Using such an additive scoring scheme is based on the assumption that mutations at different sites occur independently of each other. This is often reasonable for DNA and proteins, but not for structural RNA, where base pairing introduces very important long-range dependences.

2.7 Substitution matrices

To be able to score an alignment, we need to determine score terms for each aligned residue pair.

Given two sequences

$$x = (x_1, x_2, \ldots, x_n) \text{ and } y = (y_1, y_2, \ldots, y_m).$$

The symbols come from some alphabet $\mathcal{A}$, e.g. the four bases $\{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$ for DNA or, in the case of amino acids, the 20 symbols $\{\mathtt{A}, \mathtt{R}, \mathtt{N}, \mathtt{D}, \mathtt{C}, \mathtt{Q}, \mathtt{E}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{L}, \mathtt{K}, \mathtt{M}, \mathtt{F}, \mathtt{P}, \mathtt{S}, \mathtt{T}, \mathtt{W}, \mathtt{Y}, \mathtt{V}\}$.

For now we will only consider non-gapped alignments such as:

```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
           G+ +VK+HGKKV  A+++++AH+D++ +++++LS+LH  KL
HBB_HUMAN  GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL
```

Given a pair of aligned sequences (without gaps), we want to assign a score to the alignment that gives a measure of the relative likelihood that the sequences are related (model M) as opposed to being unrelated (model R), as

$$\frac{P(x, y \mid M)}{P(x, y \mid R)}.$$

The unrelated or *random* model R assumes that the letter a occurs independently with some frequency q_a, and hence the probability of the two sequences is the product:

$$P(x, y \mid R) = \prod_i q_{x_i} \prod_j q_{y_j}.$$

In the *match* model M, aligned pairs of residues occur with a joint probability p_{ab}, which is the probability that a and b have each evolved from some unknown original residue c as their common ancestor.

Thus, the probability for the whole alignment is:

$$P(x, y \mid M) = \prod_i p_{x_i y_i}.$$

The ratio of theses two likelihoods is known as the *odds ratio*:

$$\frac{P(x, y \mid M)}{P(x, y \mid R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}.$$

To obtain an additive scoring scheme, we take the logarithm to get the *log-odds ratio:*

$$S = \sum_i s(x_i, y_i),$$

with

$$s(a, b) := \log \left(\frac{p_{ab}}{q_a q_b} \right).$$

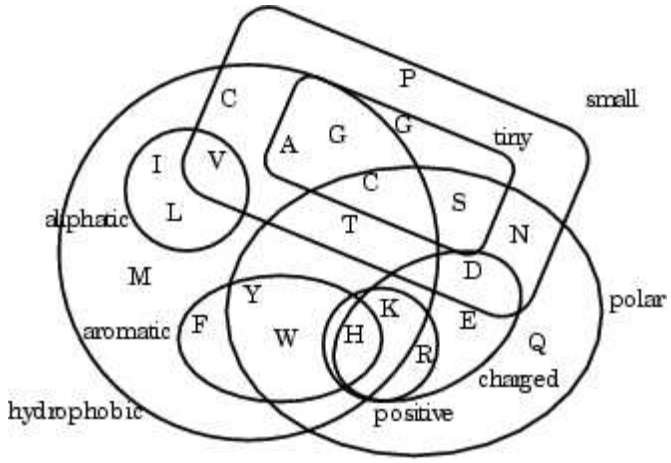
We thus obtain a matrix $s(a, b)$ that determines a score for each aligned residue pair, known as a *score* or *substitution* matrix.

For amino-acid alignments, commonly used matrices are the PAM and BLOSUM matrices, for example BLOSUM50 (to be discussed in detail later):

2.8 BLOSUM50

	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

2.9 Classification of amino acids



2.10 Gap penalties

Gaps are undesirable and thus penalized. The standard cost associated with a gap of length g is given either by a linear score

$$\gamma(g) = -gd$$

or an affine score

$$\gamma(g) = -d - (g-1)e,$$

where d is the *gap open* penalty and e is the *gap extension* penalty.

Usually, $e < d$, with the result that less isolated gaps are produced, as shown in the following comparison:

Linear gap penalty:	`GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL` `GSAQVKGHGKK--------VA--D----A-SALSDLHAHKL`
Affine gap penalty:	`GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL` `GSAQVKGHGKKVADA---------------SALSDLHAHKL`

2.11 Probabilistic model of gap penalties

Assume that the probability of a gap occurring at a particular site in a given sequence is

$$p(gap) = f(g) \prod_{x_i \text{ in gap}} q_{x_i},$$

where $f(g)$ is a function of the length g of the gap, and $\prod_{i \text{ in gap}} q_{x_i}$ is the combined probability of the inserted residues.

We can assume that the q_{x_i} are the probabilities given by the random model, because they correspond to unmatched residues. Hence, in the computation of the log-odds ratio for the gapped region, the q_{x_i} cancel and the remaining term is

$$\gamma(g) = \log\left(\frac{f(g)\prod_{x_i \text{ in gap}} q_{x_i}}{\prod q_{x_i}}\right) = \log(f(g)).$$

This shows that gap penalties correspond to the log probability of a gap of the given length.

2.12 The number of possible alignments

How many different gapped alignments are possible between two sequences $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$?

A sequences of pairs $(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)$ is called a *subsequence of indices* of x and y, if $1 \leq i_1 \leq i_2 \leq \ldots \leq i_r \leq n$ and if $1 \leq j_1 \leq j_2 \leq \ldots \leq j_r \leq m$.

We use such a subsequence to specify the set of all positions that are paired by a given alignment of x and y.

Example:

```
Alignment                Subsequence
a - c g t g t a - c      a c g t g t a c
↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕   ⇔  |   \   |   / /   |
a g c - t - g a c c      a g c t g a c c
```

Here the subsequence is:

$$(1,1),(2,3),(4,4),(6,5),(7,6),(8,8).$$

Lemma The number of possible alignments between x and y equals the number of possible subsequences of indices,

$$N(n,m) = \sum_{r=0}^{\min(n,m)} \binom{n}{r}\binom{m}{r}.$$

Proof: For each $r \in \{0,1,\ldots,\min(n,m)\}$: the number of ordered selections of $i_1, i_2, \ldots, i_r$ in $1,2,\ldots,n$ is $\binom{n}{r}$ and the number of ordered selections of $j_1, j_2, \ldots, j_r$ in $1,2,\ldots,m$ is $\binom{m}{r}$. All these possibilities can be combined. □

The number $N(n,n) = \sum_{r=0}^{n} \binom{n}{r}\binom{n}{r} = \binom{2n}{n}$ can be approximated by $\frac{2^{2^n}}{\sqrt{\pi n}}$, using Stirling's formula.

Hence, $N(1000,1000) \approx 10^{600}$.

2.13 Alignment algorithms

Given a scoring scheme, we need to have an algorithm that computes the highest-scoring alignment of two sequences.

We will discuss alignment algorithms based on *dynamic programming*. Dynamic programming algorithms play a central role in computational sequence analysis. They are guaranteed to find the optimal scoring alignment.

However, for large sequences they can be too slow and heuristics (such as BLAST, FASTA, MUMMER etc) are then used that usually perform very well, but will miss the best alignment for some sequence pairs.

Depending on the input data, there are a number of different variants of alignment that are considered, among them *global alignment*, *local alignment* and *overlap alignment*.

We will use two short amino acid sequences for illustration:

`HEAGAWGHEE` and `PAWHEAE`.

To score the alignment we will use the BLOSUM50 matrix and a gap cost of $d = -8$. (Later, we will also use affine gap costs.)

Here they are arranged to show a matrix of corresponding BLOSUM50 values:

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	-1	-4	-2	-2	-1	-1
A	-2	-1	**5**	0	**5**	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	**15**	-3	-3	-3	-3
H	**10**	0	-2	-2	-2	-3	-3	**10**	0	0
E	0	**6**	-1	-3	-1	-3	-3	0	**6**	**6**
A	-2	-1	**5**	0	**5**	-3	0	-2	-1	-1
E	0	**6**	-1	-3	-1	-3	-3	0	**6**	**6**

2.14 Global alignment: Needleman-Wunsch algorithm

Saul Needleman and Christian Wunsch (1970), improved by Peter Sellers (1974).

Consider the problem of obtaining the best *global* alignment of two sequences. The Needleman-Wunsch algorithm is a dynamic program that solves this problem.

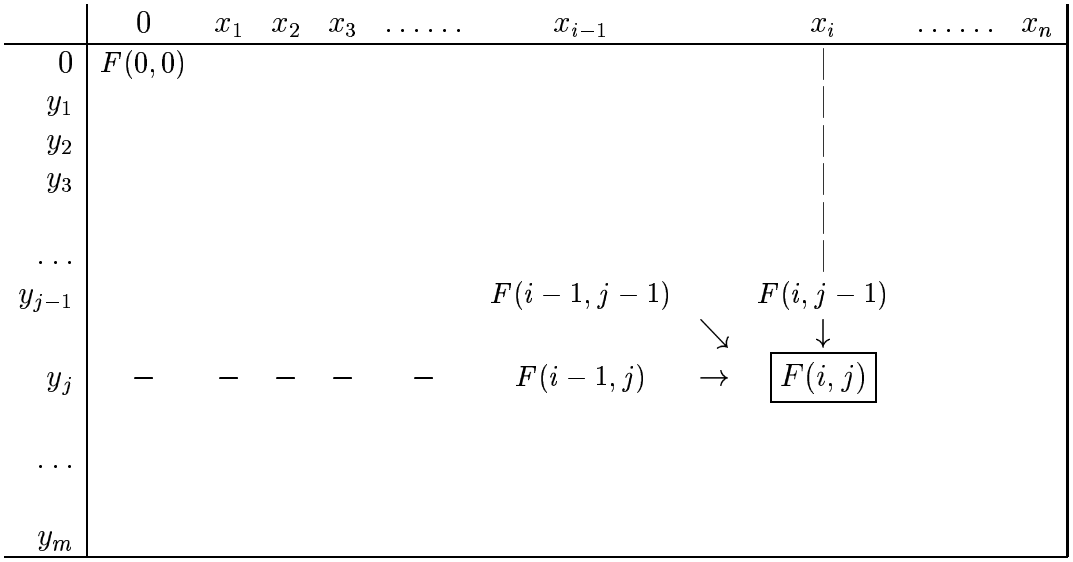
Idea: Build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences.

Given two sequences $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_m)$. We will compute a matrix

$$F : \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\} \to \mathbb{R}$$

in which $F(i, j)$ equals the best score of the alignment of the two prefixes $(x_1, x_2, \ldots, x_i)$ and $(y_1, y_2, \ldots, y_j)$.

This will be done recursively by setting $F(0, 0) = 0$ and then computing $F(i, j)$ from $F(i - 1, j - 1)$, $F(i - 1, j)$ and $F(i, j - 1)$:



2.15 The recursion

There are three ways in which an alignment can be extended up to (i, j):

x_i aligns to y_i:	x_i aligns to a gap:	y_i aligns to a gap:
I G A x_i	A I G A x_i	G A x_i - -
L G V y_j	G V y_j - -	S L G V y_j

We obtain $F(i, j)$ as the largest score arising from these three options:

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

This is applied repeatedly until the whole matrix $F(i, j)$ is filled with values.

To complete the description of the recursion, we need to set the values of $F(i, 0)$ and $F(0, j)$ for $i \neq 0$ and $j \neq 0$:

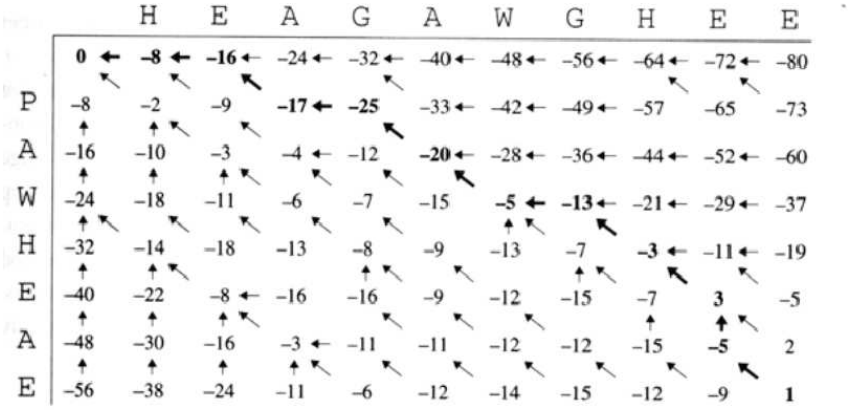
We set $F(i, 0) =$ _____ for $i = 0, 1, \ldots, n$ and
we set $F(0, j) =$ _____ for $j = 0, 1, \ldots, m$.

The final value $F(n, m)$ contains the score of the best global alignment between x and y.

To obtain an alignment corresponding to this score, we must find the path of choices that the recursion made to obtain the score. This is called a *traceback*.

2.16 Example of global alignment matrix

Alignment matrix made using $d = -8$ and BLOSUM50 scores:



HEAGAWGHE-E
--P-AW-HEAE

Durbin *et al.* (1998)

2.17 Needleman-Wunsch algorithm

Input: two sequences x and y
Output: optimal alignment and score α
Initialization: Set $F(i,0) := -id$ for all $i = 0, 1, 2, \ldots, n$
Set $F(0,j) := -jd$ for all $j = 0, 1, 2, \ldots, m$
For $i = 1, 2, \ldots, n$ **do**:
    For $j = 1, 2, \ldots, m$ **do**:
        Set $F(i,j) := \max \begin{cases} F(i-1, j-1) + s(x_i, y_i) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$
        Set backtrace $T(i,j)$ to the maximizing pair (i', j')
The best score is $\alpha := F(n,m)$
Set $(i,j) := (n,m)$

repeat
    if $T(i,j) = (i-1, j-1)$ **print** $\binom{x_{i-1}}{y_{j-1}}$
    else if $T(i,j) = (i-1, j)$ **print** $\binom{x_{i-1}}{-}$ **else** **print** $\binom{-}{y_{j-1}}$
    Set $(i,j) := T(i,j)$
until $(i,j) = (0,0)$.

2.18 Complexity

Complexity of the Needleman-Wunsch algorithm:

We need to store $(n+1) \times (m+1)$ numbers. Each number takes a constant number of calculations to compute: three sums and a max.

Hence, the algorithm requires $O(nm)$ time and memory.

For biological sequence analysis, we prefer algorithms that have time and space requirements that are linear in the length of the sequences. Quadratic time algorithms are a little slow, but feasible. $O(n^3)$ algorithms are only feasible for very short sequences.

Something to think about: if we are only interested in the best score, but not the actual alignment, then it is easy to reduce the space requirement to linear.

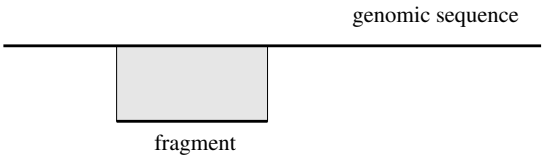
2.19 Local alignment: Smith-Waterman algorithm

Temple Smith and Mike Waterman (1981).

Global alignment is applicable when we have two similar sequences that we want to align from end-to-end, e.g. two homologous genes from related species.

Often, however, we have two sequences x and y and we would like to find the best match

between *subsequences* of both. For example, we may want to find the position of a fragment of DNA in a genomic sequence:



The best scoring alignment of two subsequences of x and y is called the best *local alignment.*

The Smith-Waterman local alignment algorithm is obtained by making two simple modifications to the global alignment algorithm.

(1) In the main recursion, we set the value of $F(i,j)$ to zero, if all attainable values at position (i,j) are negative:

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

The value $F(i,j) = 0$ indicates that we should start a new alignment at (i,j). This is because, if the best alignment up to (i,j) has a negative score, then it is better to start a new one, rather than to extend the old one.

Note that, in particular, we have $F(i,0) = 0$ and $F(0,j) = 0$ for all $i = 0, 1, 2, \ldots, n$ and $j = 0, 1, 2, \ldots, m$.

(2) Instead of starting the traceback at (n,m), we start it at the cell with the highest score, $\arg\max F(i,j)$. The traceback ends upon arrival at a cell with score 0, with corresponds to the start of the alignment.

For this algorithm to work, we require that the expected score for a random match is negative, i.e. that

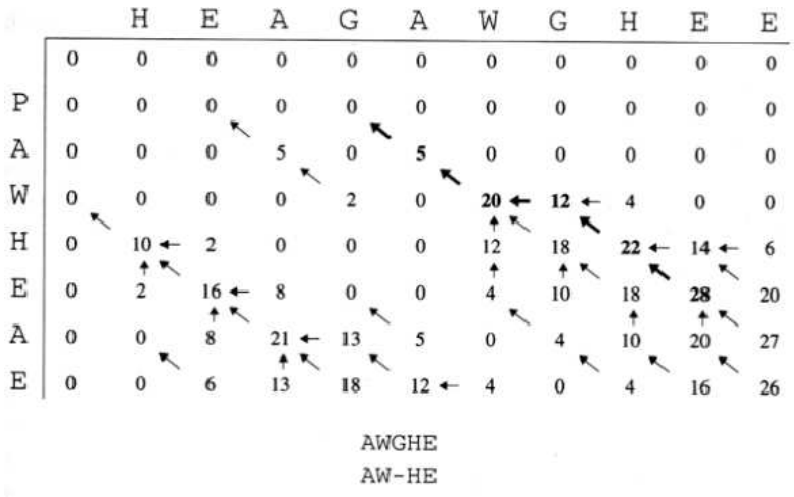
$$\sum_{a,b} q_a q_b s(a,b) < 0,$$

where q_a and q_b are the probabilities for the seeing the symbol a or b at any given position, respectively.

Otherwise, matrix entries will tend to be positive, producing long matches between random sequences.

2.20 Example of a local alignment matrix

Alignment matrix made using $d = -8$ and BLOSUM50 scores:



Durbin *et al.* (1998)

2.21 Smith-Waterman algorithm

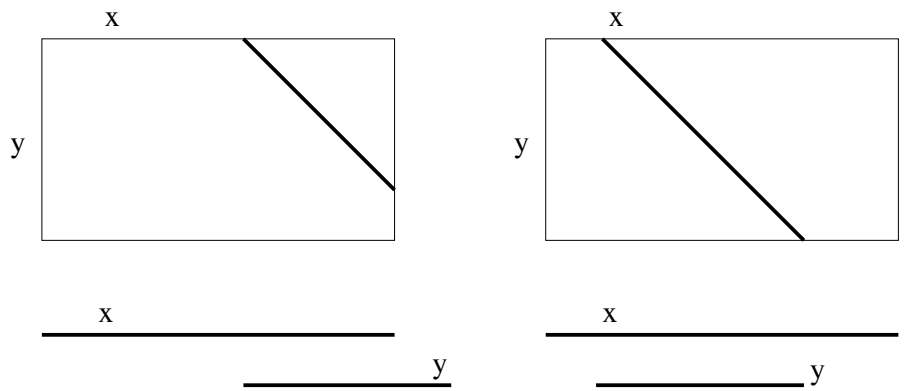
Input: two sequences x and y
Output: optimal local alignment and score α
Initialization: Set $F(i,0) := 0$ for all $i = 0, 1, 2, \ldots, n$
Set $F(0,j) := 0$ for all $j = 0, 1, 2, \ldots, m$
For $i = 1, 2, \ldots, n$ **do**:
  For $j = 1, 2, \ldots, m$ **do**:

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

  Set backtrace $T(i,j)$ to the maximizing pair (i', j')
Set $(i,j) := \arg\max\{F(i,j) \mid i = 1, 2, \ldots, n, j = 1, 2, \ldots, m\}$
The best score is $\alpha := F(i,j)$
repeat
  if $T(i,j) = (i-1, j-1)$ **print** $\binom{x_{i-1}}{y_{j-1}}$
  else if $T(i,j) = (i-1, j)$ **print** $\binom{x_{i-1}}{-}$ **else print** $\binom{-}{y_{j-1}}$
  Set $(i,j) := T(i,j)$
until $F(i,j) = 0$.

2.22 Algorithm for finding overlap alignments

If we are given different fragments of genomic DNA that we would like to piece together, then we need an alignment method that does not penalize overhanging ends:



For an overlap alignment, matches should be allowed to start anywhere on the top or left boundary of the matrix, and should be allowed to end anywhere on the bottom or right boundary.

To allow the former, simply set $F(i,0) = 0$ and $F(0,j) = 0$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$. The recurrence relations are those used for global alignment.

To allow the latter, start the traceback at the best scoring cell contained in the bottom row or rightmost column, i.e. start at

$$\arg\max\{F(i,j) \mid i = n \text{ or } j = m\}.$$

2.23 Example of an overlap alignment

Given two sequences $x =$ `acatatt` and $y =$ `ttttac`. We use 1, -1 and 2 for the match, mismatch and gap scores, respectively:

	0	a	c	a	t	a	t	t
0								
t								
t								
t								
t								
a								
c								

2.24 Dynamic programming with more complex models

So far, we have considered the case of a linear gap score. This type of scoring scheme is not ideal for biological sequences, as it penalizes additional gap steps as much as the initial one. However, gaps are often longer than one residue and one would like a scheme that makes it expensive to open a gap, but once open, makes it less expensive to extend a gap.

The simplest solution is to let the gap score be a function γ of the length of the gap, like this:

$$F(i,j) := \max \begin{cases} F(i-1,j-1) + s(x_i, y_j), & \\ F(k,j) + \gamma(i-k), & k = 0, \ldots, i-1, \\ F(i,k) + \gamma(j-k), & k = 0, \ldots, j-1. \end{cases}$$

Problem: requires $O(n^3)$ time, because for each cell we need to inspect $i+j+1$ predecessors.

2.25 Affine gap scores

The standard alternative to using the above recursion is to use an *affine gap score*

$$\gamma(g) = -d - (g-1)e,$$

with d the *gap-open score* and e the *gap-extension* score.

We will discuss how to modify the Needleman-Wunsch algorithm for global alignment so as to incorporate affine gap costs. Approach is due to Osamu Gotoh (1982).

Instead of using one matrix $F(i,j)$ to represent the best score attainable up to x_i and y_j, we will now use three matrices M, I_x and I_y:

Case 1 x_i aligns to y_j:	Case 2 x_i aligns to a gap:	Case 3 y_j aligns to a gap:
I G A x_i L G V y_j	A I G A x_i G V y_j - -	G A x_i - - S L G V y_j

1. $M(i,j)$ is the best score up to (i,j), given that x_i is aligned to y_j,
2. $I_x(i,j)$ is the best score up to (i,j), given that x_i is aligned to a gap, and
3. $I_y(i,j)$ is the best score up to (i,j), given that y_j is aligned to a gap.

2.26 Recursion for global alignment with affine gap costs

Initialization:

$$M(0,0) = 0,\ I_x(0,0) = I_y(0,0) = -\infty$$
$$M(i,0) = I_x(i,0) = -d-(i-1)e,\ I_y(i,0) = -\infty,\ \text{for } i = 1,\ldots,n,\ \text{and}$$
$$M(0,j) = I_y(0,j) = -d-(j-1)e,\ I_x(0,j) = -\infty,\ \text{for } j = 1,\ldots,m.$$

Recursion:

$$M(i,j) = \max\begin{cases} M(i-1,j-1)+s(x_i,y_j), \\ I_x(i-1,j-1)+s(x_i,y_j), \\ I_y(i-1,j-1)+s(x_i,y_j); \end{cases}$$

$$I_x(i,j) = \max\begin{cases} M(i-1,j)-d, \\ I_x(i-1,j)-e; \end{cases}$$

$$I_y(i,j) = \max\begin{cases} M(i,j-1)-d, \\ I_y(i,j-1)-e. \end{cases}$$

2.27 Example of a global alignment with affine gap costs

Given two sequences $x =$ `ttagat` and $y =$ `ttgt`. We use 1, -1, 2 and 1 for the match-, mismatch-, gap-open and gap-extension scores, respectively:

		0	t	t	a	g	t
0	M						
	I_x						
	I_y						
t	M						
	I_x						
	I_y						
t	M						
	I_x						
	I_y						
g	M						
	I_x						
	I_y						
t	M						
	I_x						
	I_y						

2.28 Alignment in linear space

Can we compute a best alignment between two sequences
$x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, m)$ using only linear space?

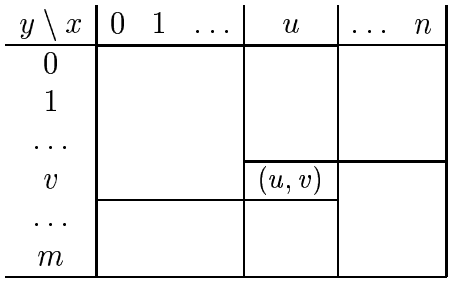
The best score of an alignment is easy to compute in linear space, as $F(i, j)$ is computed locally from the values in the previous and current column only.

However, to obtain an actual alignment in linear space, we need to replace the traceback matrix.

We will discuss this the case of global alignments.

Idea: Divide-and-conquer! Consider the middle column $u = \lfloor \frac{n}{2} \rfloor$ of the F matrix. If we knew at which cell (u, v) the best scoring alignment passes through the u^{th} column, then we could split the dynamic-programming problem into two parts:

- align from $(0, 0)$ to (u, v), and then
- align from (u, v) to (n, m).



Concatenate the two solutions to obtain the final result.

How to determine v, the row in which a best path crosses the u^{th} column?

For $i > u$, define $c(i, j)$ such that $(u, c(i, j))$ is on an optimal path from $(1, 1)$ to (i, j).

As we compute $F(i, j)$ column for column, we update $c(i, j)$:

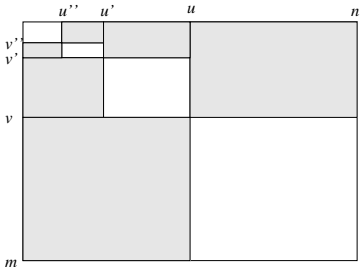
Let (i', j') be the cell from which $F(i, j)$ is obtained. Set

$$c(i, j) := \begin{cases} j', & \text{if } i' = u, \\ c(i', j'), & \text{else} \end{cases} \qquad (\text{ for } i > u).$$

.

Note that $c(i, j)$ is computed locally from the values in the previous and current column only. The final value is $v = c(n, m)$.

Once we have determined (u, v), we recurse, as indicated here:



If we implement $c(i,j)$ and $F(i,j)$ using only two vectors of length m for each, then the algorithm only requires linear space. We obtain the actual alignment as a sequence of pairs $(1,v_1),(2,v_2),\ldots(n,v_n)$.

What is the time complexity? We first look at $1 \times nm$ cells, then at $\frac{1}{2}nm$ cells, then at $\frac{1}{4}nm$ cells etc. As $\sum_{i=0}^{n} \frac{1}{2^i} < 2$, this algorithm is only twice as slow as the quadratic-space one!

2.29 Simplifying the affine-gap algorithm

In practice, the affine-gap formulation of the dynamic programs for sequence alignment usually use only two matrices, M and I, corresponding to an alignment of two symbols and an insertion in one of the two sequences, respectively. For global alignment, the recursions are:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i, y_j), \\ I(i-1,j-1) + s(x_i, y_j); \end{cases}$$

$$I(i,j) = \max \begin{cases} M(i-1,j) - d, \\ I(i-1,j) - e, \\ M(i,j-1) - d, \\ I(i,j-1) - e. \end{cases}$$

This produces the same result as the original algorithm, if the lowest mismatch score is $> -2e$. But, even if this does not hold, then the difference in score and alignment will be insignificant (in a poorly matched gapped region).

Assume that the original algorithm (using M, I_x and I_y) produces the following optimal alignment:

```
x x x x - - - a x x x x x x x
y y y y y y b - - - - y y y y
```

If $s(a,b) < -2e$, then the modified algorithm (using M and I) will produce the following higher scoring alignment, adding a gap before `a` and one after `b`:

```
x x x x - - - - a x x x x x x x
y y y y y y b - - - - - y y y y
```

However, situations in which $\binom{-}{b}$ is directly followed by $\binom{a}{-}$ (or vice-versa) are uninteresting and so the original algorithm rules them out.

2.30 Where do we stand?

So far, we have discussed:

- the dot matrix for visual comparision of two sequences,
- global alignments and the Needleman-Wunsch algorithm,
- local alignments and the Smith-Waterman algorithm, and
- overlap alignments and a corresponding dynamic program.

A naive implementation of any of the dynamic programming algorithms uses $O(nm)$ time and $O(nm)$ space. We saw that:

- the space complexity can be reduced to $O(n)$.

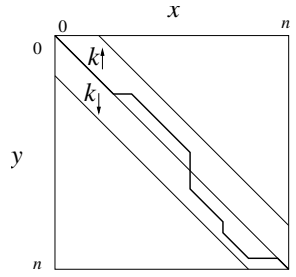
Now we ask:

- can we reduce the time complexity, too?

2.31 Banded global alignment

For simplicity, we consider DNA sequences, assume $n = m$ and use a linear gap score d.

Idea: Instead of computing the whole matrix F, use only a *band* of cells along the main diagonal:



Let $2k$ denote the height of the band. Obviously, the time complexity of the banded algorithm will be $O(kn)$.

Questions: Will this algorithm produce an optimal global alignment? What should k be set to?

2.32 The KBand algorithm

Input: two sequences x and y of equal length n, integer k
Output: best score α of global alignment at most k diagonals away from main diagonal
Initialization: Set $F(i,0) := -id$ for all $i = 0, 1, 2, \ldots, k$.
Set $F(0,j) := -jd$ for all $j = 0, 1, 2, \ldots, k$.

for $i = 1$ **to** n **do**
  for $h = -k$ **to** k **do**
    $j := i + h$
    if $1 \leq j \leq n$ **then**
      $F(i,j) := F(i-1, j-1) + s(x_i, y_j)$
      if insideBand$(i-1, j, k)$ **then**
        $F(i,j) := \max\{F(i,j), F(i-1,j) - d\}$
      if insideBand$(i, j-1, k)$ **then**
        $F(i,j) := \max\{F(i,j), F(i,j-1) - d\}$
return $F(n,n)$

To test whether (i, j) is inside the band, we use:

$$\text{insideBand}(i, j, k) := (-k \leq i - j \leq k).$$

2.33 Searching for high-identity alignments

We can use the KBand algorithm as a fast method for finding high-identity alignments:

If we know that the two input sequences are highly similar and we have a bound b on the number of gaps that will occur in the best alignment, then the KBand algorithm with $k = b$ will compute an optimal alignment.

For example, in forensics, one must sometimes determine whether a sample of human mtDNA obtained from a victim matches a sample obtained from a relative (or from a hair brush etc). If two such sequences differ by more than a couple of base-pairs or gaps, then they are not considered a match.

2.34 Optimal alignments using KBand

Given two sequences x and y of the same length n. Let M be the match score and d the gap penalty.

Question: Let α_k be the best score obtained using the KBand algorithm for a given k. When is α_k equal to the optimal global alignment score α?

Lemma If $\alpha_k \geq M(n - k - 1) - 2(k + 1)d$, then $\alpha_k = \alpha$.

Proof If there exists an optimal alignment with score α that does not leave the band, then clearly $\alpha_k = \alpha$. Else, all optimal alignments leave the band somewhere. This requires insertion of at least $k + 1$ gaps in each sequence, and allows only at most $n - k - 1$ matches, giving the desired bound. □

2.35 Optimal alignment using repeated KBand

The following algorithm computes an optimal alignment by repeated application of the KBand algorithm, with larger and larger k:

Input: two sequences x and y of the same length
Output: an optimal global alignment of x and y

Initialize $k := 1$

repeat
compute α_k using KBand
if $\alpha_k \geq M(n-k-1) - 2(k+1)d$ **then**
return α_k
$k := 2k$
end

As usual, we omit details of the traceback.

2.36 Analysis of time complexity

The algorithm terminates when:

$$\begin{aligned} \alpha_k \geq M(n-k-1) - 2(k+1)d \quad &\Leftrightarrow \\ \alpha_k - Mn + M + 2d \geq -(M+2d)k \quad &\Leftrightarrow \\ -\alpha_k + Mn - (M+2d) \leq (M+2d)k \quad &\Leftrightarrow \\ \frac{Mn - \alpha_k}{M+2d} - 1 \leq k \end{aligned}$$

At this point, the total complexity is:

$$n + 2n + 4n + \ldots + kn \leq 2kn.$$

So far, this doesn't look better than nn. To bound the total complexity, we need a bound on k.

When the algorithm stops for k, we must have:

$$\frac{k}{2} < \frac{Mn - \alpha_{\frac{k}{2}}}{M+2d} - 1.$$

There are two cases: If $\alpha_{\frac{k}{2}} = \alpha_k = \alpha$, then

$$k < 2(\frac{Mn - \alpha}{M+2d} - 1).$$

Otherwise, $\alpha_{\frac{k}{2}} < \alpha_k = \alpha$. Then any optimal alignment must have more than $\frac{k}{2}$ spaces, and thus

$$\alpha \leq M(n - \frac{k}{2} - 1) + 2(\frac{k}{2} + 1)d \quad \Rightarrow \quad k \leq 2(\frac{Mn - \alpha}{M+2d} - 1).$$

As $M+2d$ is a constant, it follows that k is bounded by $O(\Delta)$, with $\Delta = Mn - \alpha$, and thus the total bound is $O(\Delta n)$. □

In consequence, the more similar the sequences, the faster the KBand algorithm will run!
We can easily generalize to $n \neq m$. Space saving, how much do we need?