3 Pairwise alignment

We will discuss:

- The principle of *dynamic programming*
- Dot plots
- Scoring schemes
- Alignment algorithms based on dynamic programming
- Important variations of the basic alignment algorithm

3.1 References

- R. Durbin, S. Eddy, A. Krogh, G. Mitchison: *Biological sequence analysis*. Cambridge University Press, 1998. ISBN 0-521-62971-3 (Chapter 2)
- Neil C. Jones, Pavel A. Pevzner: *An Introduction to Bioinformatics Algorithms*. MIT Press, Cambridge, MA, 2004. ISBN 0-262-10106-8
- David B. Mount: *Bioinformatics*. Sequence and Genome Analysis. Cold Spring Harbor Laboratory Press, New York, 2001. ISBN 0-87969-608-7
- Kun-Mao Chao, Louxin Zhang, Sequence comparison, theory and methods, Springer, ISBN 987-1-84800-319-4

Paradigm: Dynamic Programming

Powerful algorithmic design paradigm

Principle
Compute solutions of "bigger" problems by combining solutions of smaller subproblems. Storing all solutions avoids recomputing the same quantity over and over again, and a potential exponential blow-up in the running time.

Well-suited if subproblems share subsubproblems.

Typically applied to *optimization problems*. Three components:

1. Recursively define the value of an optimal solution
2. Compute values of subproblems in bottom-up fashion (storing the solution values)
3. Construct solution (traceback)

Paradigm: Dynamic Programming (2)

Examples
Fibonacci numbers[1], knapsack, longest common subsequence, alignments in database search (BLAST, FASTA), multiple alignments (clustalW), gene finding (GENSCAN), RNA-folding (mfold), phylogenetic inference (PHYLIP), ...

3.2 Pairwise sequence alignment: motivation

Comparative Genomics

- Gene finding, gene function determination
  Compare sequence to genes of known function. Success stories:
  - Example 1: v-sis-oncogene[2] (discovered 1984). Comparison with all known genes led to striking similarity with regular growth gene. Conclusion: oncogene responsible for growth at the wrong time.
  - Example 2: discovery of the cystic fibrosis gene (1989). Location narrowed down to 10^6 nucleotides on chromosome 7. Compared to all genes → similarity with gene for ATP binding proteins. Shed light on the nature of cystic fibrosis.
  - ...

- High sequence similarity (e. g., human—mouse: 97%) between species allows to study other organisms in order to understand humans (e. g., Waardenburg's syndrome)
- derive information about common origin (evolutionary trees)
- ...

3.3 Sequence alignment (informally)

Two strings:	⟶	Alignment:
`IMISSMISSISSIPPI`		`I-MISSMISSISIPPI-`
		`  |||| ||   ||||`
`MYMISSISAHIPPIE`		`MYMISS-ISAH-IPPIE`

`I`: Isoleucine, `M`: Methionine, `S`: Serine, `P`: Proline, `Y`: Tyrosine, `A`: Alanine, `H`: Histidine, `E`: Glutamic Acid

Two sequences are (globally) aligned by writing them across a page in two rows. Identical (or similar) characters are placed in the same column and called *matches*. Non-identical characters are either placed in the same column as a *mismatch*, or they are opposite to a *gap* in the other sequence. Also, we disallow columns that consist of gaps only.

3.4 Sequence alignment (formally)

One way to formalize pairwise sequence alignment is as follows: We are given two sequences $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_n)$ over an alphabet Σ. Let $- \notin \Sigma$ be the *gap symbol*, also called *space*. Let $h\colon (\Sigma \cup \{-\})^* \to \Sigma^*$ be the mapping that removes all gap symbols from a sequence over the alphabet $\Sigma \cup \{-\}$.

[1] blackboard
[2] oncogenes are virus genes that cause cancer-like transformation of infected cells

For example, $h(\texttt{MYMISS-ISAH-IPPIE}) = \texttt{MYMISSISAHIPPIE}$.

Then a global alignment of x and y is a pair of sequences x', y' such that

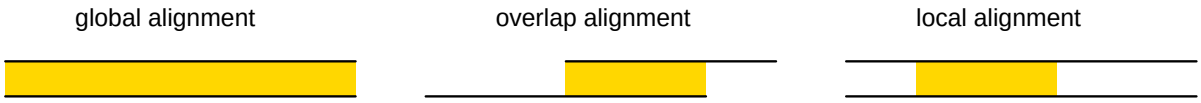
$$h(x') = x, \quad h(y') = y, \quad |x'| = |y'|, \quad \text{and} \quad (x'_i, y'_i) \neq (-, -) \text{ for all } i.$$

(Here $|\cdot|$ denotes length.)

3.5 More terminology

- *Match* - same (or similar) letter in both rows
- *Mismatch* - different letters in both rows
- *Insertion* - the letter opposite to a space
- *Deletion* - the space opposite to a letter
- *Indel* - a column containing a space

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



In an overlap alignment, we do not charge the end gaps (hence it is also called end-gap free alignment).

A local alignment is the same as a global alignment of two substrings of the sequences.

3.6 Finding alignments

How many alignments are there? Answer: many (blackboard)

How can we find a good pairwise alignment?

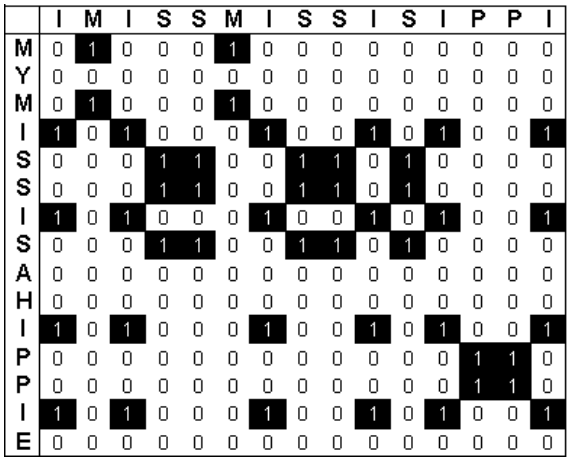
Why not simply *visualize* the data?

3.7 Dot plots

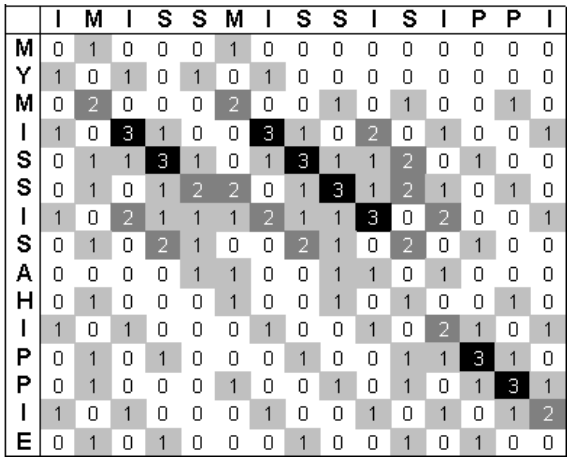
We can draw a matrix spanned up by two sequences and place a dot in each cell for which the correspondig symbols match. Stretches of matching symbols will show up as diagonal lines this way.

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

To obtain cleaner pictures, a *window size* w and a *stringency* s are used: A dot is only drawn at point (x, y) if within w positions around it there are at least s matches.

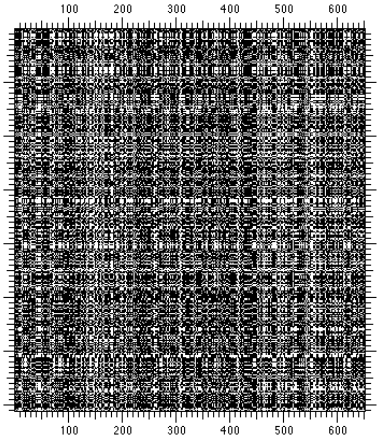


$w = 1, s = 1$

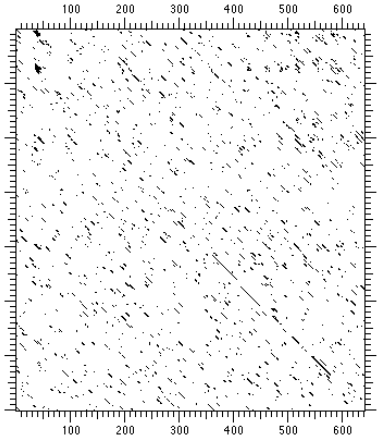


$w = 3, s = 3$

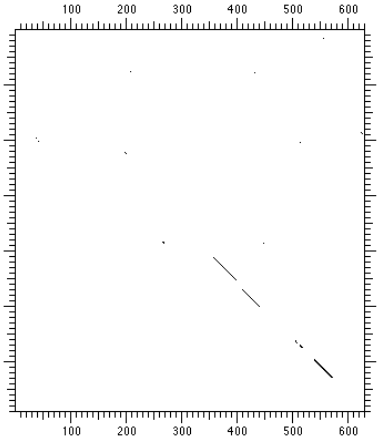
Here is a biological example: (I forgot which sequences…)



$w = 1, s = 1$



$w = 11, s = 7$



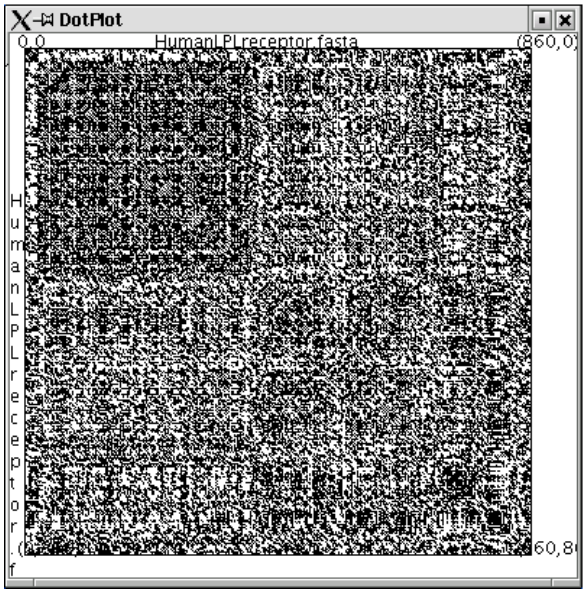
$w = 23, s = 15$

Tools on the web to play around with:

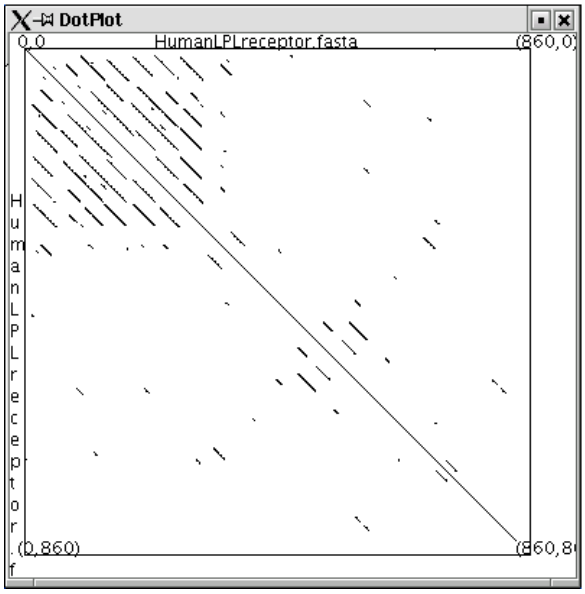
- `http://www.isrec.isb-sib.ch/java/dotlet/Dotlet.html`
- `http://arbl.cvmbs.colostate.edu/molkit/dnadot/`

3.8 Repeat detection using dot plots

Here is another one. These dot plots of the human LDL receptor (protein sequence) against itself reveal many *repeats* in the first 300 positions.



$w = 1, s = 1$



$w = 23, s = 7$

3.9 How to score it?

So we *see* there is an alignment, but do we really *have* it, written in two rows?

To come up with an algorithm, we need a formal concept when an alignment is "good", i.e., *significant*. Let us have a look at some good and bad examples before we formally introduce a *scoring scheme*.

3.10 Significance of alignments

In the alignments below, the middle row contains a letter for identical amino acids, and a + if the amino acids are similar.

1. An alignment between very similar human alpha- and beta hemoglobins:

```
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   GSGYLVGDSLTFVDLL--VAQHTADLLAANAALLDEFPQFKAHQE
```

In (1), there are many positions at which the two corresponding residues are identical. Many others are functionally conservative. E.g., the `D-E` pair towards the end: 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.

3.11 Scoring schemes

The basic mutational processes are *substitutions*, *insertions*, and *deletions*. Substitutions give rise to *mismatches*. Insertions and deletions give rise to *gaps*.

The *total score* assigned to an alignment is the

1. sum of terms for matches and mismatches, plus the
2. sum of terms for gaps (i.e., indels).

Formally we can treat the space character just like any other. Then we obtain the following additive scoring scheme:

The score of an alignment (x', y') is

$$\sum_i \delta(x'_i, y'_i),$$

where $\delta\colon (\Sigma \cup \{-\})^2 \to \mathbb{R}$ is a *score matrix*.

3.12 Hamming distance and Levenshtein distance

So how should we choose the **score matrix** δ?

In the simplest case, we do not allow any gaps at all and charge all mismatches at unit cost. This is called the *Hamming distance.* The alignment is forced to be on one diagonal of the dot plot.

If we charge all insertions, deletions, and mismatches at unit cost, we obtain the *Levenshtein distance.* This measure is also called the *edit distance*, because it counts the number of elementary edit operations needed to transform one sequence into the other.

Both scoring schemes have applications in biological sequence analysis, especially for nucleic acids. Often, however, it is better to

1. distinguish the type of a mismatch, and
2. take the length of consecutive gaps into account.

3.13 Classification of amino acids

Amino acids can be grouped according to chemical properties.

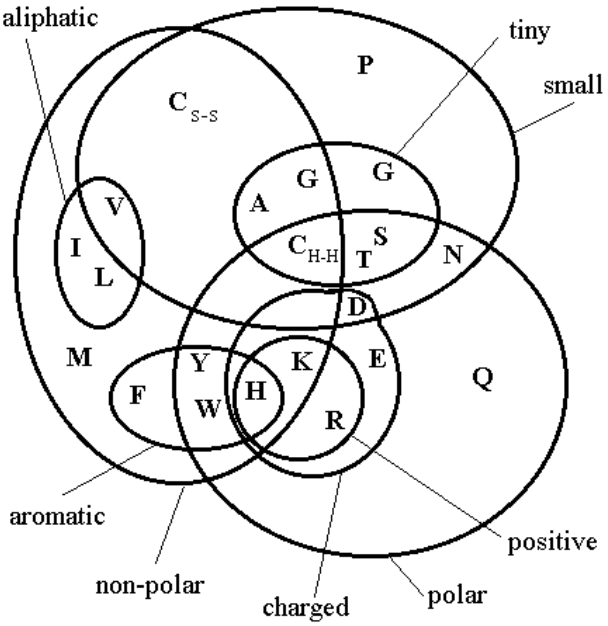


Table of standard amino acid abbreviations and side chain properties

Main article: List of standard amino acids

Amino Acid	3-Letter[80]	1-Letter[80]	Side chain polarity[80]	Side chain charge (pH 7)[80]	Hydropathy index[81]
Alanine	Ala	A	nonpolar	neutral	1.8
Arginine	Arg	R	polar	positive	-4.5
Asparagine	Asn	N	polar	neutral	-3.5
Aspartic acid	Asp	D	polar	negative	-3.5
Cysteine	Cys	C	nonpolar	neutral	2.5
Glutamic acid	Glu	E	polar	negative	-3.5
Glutamine	Gln	Q	polar	neutral	-3.5
Glycine	Gly	G	nonpolar	neutral	-0.4
Histidine	His	H	polar	positive	-3.2
Isoleucine	Ile	I	nonpolar	neutral	4.5
Leucine	Leu	L	nonpolar	neutral	3.8
Lysine	Lys	K	polar	positive	-3.9
Methionine	Met	M	nonpolar	neutral	1.9
Phenylalanine	Phe	F	nonpolar	neutral	2.8
Proline	Pro	P	nonpolar	neutral	-1.6
Serine	Ser	S	polar	neutral	-0.8
Threonine	Thr	T	polar	neutral	-0.7
Tryptophan	Trp	W	nonpolar	neutral	-0.9
Tyrosine	Tyr	Y	polar	neutral	-1.3
Valine	Val	V	nonpolar	neutral	4.2

3.14 The BLOSUM50 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
```

3.15 Gap penalties

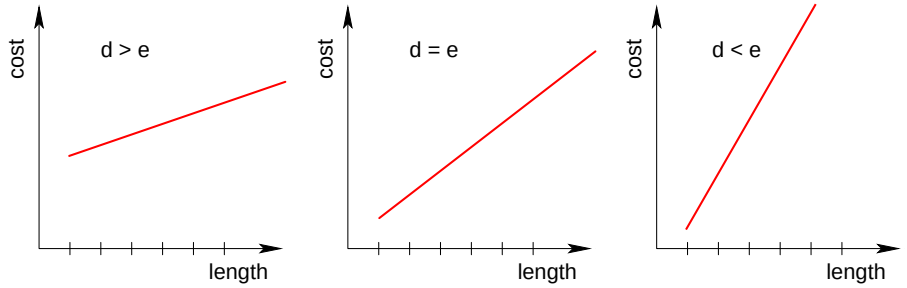
Gaps in an alignment are undesirable and thus penalized. In its simplest form, the cost associated with a gap of length $g \geq 1$ is given by a *linear* score,

$$\gamma(g) = -g \cdot d\,.$$

An *affine* score, however,

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

often produces better results. Here d is the *gap open* penalty and e is the *gap extension* penalty.



Usually one sets $e < d$, i.e., there is a large penalty for opening a gap, but a smaller penalty for extending it. Then affine gap costs favor alignments with fewer but larger gaps. For example:

Using linear gap penalties:	`GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL` `GSAQVKGHGKK--------VA--D----A-SALSDLHAHKL`
Using affine gap penalties:	`GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL` `GSAQVKGHGKKVADA---------------SALSDLHAHKL`

The case $d < e$ is sometimes used when comparing output of DNA sequencing machines. There it happens frequently that single bases are left out near the end of a read.

3.16 Remarks on scoring schemes

1. The scoring schemes we have seen so far are based solely on primary structure. This is a reasonable approximation especially for DNA. For RNA, one observes that bases which are coupled by secondary structure are highly correlated. But even for proteins the primary structure can tell us a lot.
2. We will not explain the probabilistic background of the additive scoring scheme (in this lecture), but simply take it as granted. Thus we will not explain, e.g., how the BLOSUM50 matrix was derived from experimental data, assuming a certain model of evolution, etc.
3. The alignment algorithms we describe next can be generalized to affine gap cost without an increase in run time. We will start, however, with linear gap costs.

3.17 Alignment algorithms

Given a scoring scheme and two input sequences, we need an algorithm to compute the highest-scoring alignment of the 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 highest-scoring alignment.

Note: For large sequences exact "DP algorithms" can be too slow and *heuristics* (such as BLAST, FASTA, MUMMER, QUASAR,...) are then used which perform very well in most cases, but will miss the best alignment for some sequence pairs.

3.18 Global alignment: Needleman-Wunsch algorithm

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

Consider the problem of finding the optimal *global alignment* of two sequences x and y. The Needleman-Wunsch algorithm is a "dynamic program" that solves this problem. What does this mean?

The **idea** is to build up a table of optimal alignments for all pairs of prefixes of x and y using (already known) optimal alignments of shorter prefixes of x and y.

We are given two sequences $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_n)$. We will compute a matrix, usually called *dynamic programming table*,

$$F : \{0, 1, 2, \ldots, m\} \times \{0, 1, 2, \ldots, n\} \to \mathbb{R}$$

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

Outline of the algorithm: We fill the table F recursively, bottom-up[3]. We start with initial cases like $F(0, 0) = 0$ and then compute each $F(i, j)$ from $F(i-1, j-1)$, $F(i-1, j)$ and $F(i, j-1)$:

		x_1	x_2	...	x_{i-1}		x_i	...	x_m
	$F(0,0)$	$F(1,0)$	$F(2,0)$				$\vdots$		
y_1	$F(0,1)$						$\vdots$		
y_2	$F(0,2)$						$\vdots$		
$\vdots$							$\vdots$		
y_{j-1}					$F(i-1,j-1)$		$F(i,j-1)$		
						$\nwarrow$	$\uparrow$		
y_j	...	...	...	...	$F(i-1,j)$	$\leftarrow$	$F(i,j)$		
$\vdots$									
y_n									

This is applied until the whole matrix F is filled with values. Then $F(m, n)$ is the score of the best global alignment.

Why *can* we apply such a **recursion**?

There are three ways how the last column of an alignment of $(x_1, x_2, \ldots, x_i)$ and $(y_1, y_2, \ldots, y_j)$ can look like:

x_i aligns to y_j:

I	G	A	x_i
L	G	V	y_j

x_i aligns to a gap:

A	I	G	A	x_i
G	V	y_j	-	-

y_j aligns to a gap:

G	A	x_i	-	-
S	L	G	V	y_j

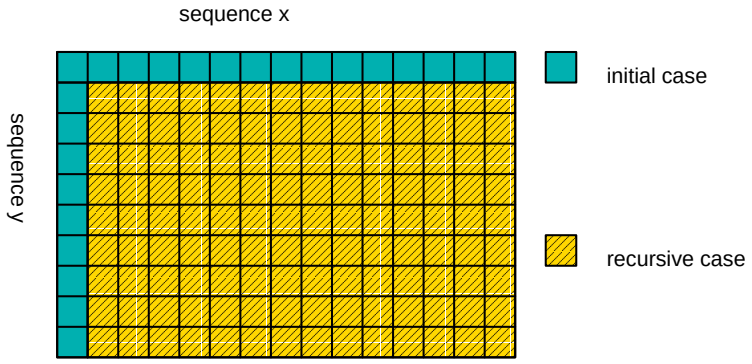
[3] This corresponds to step 2 on the DP paradigm slides

We obtain $F(i, j)$ as the largest score that arises from one of these cases[4]:

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

To complete the description of the recursion, we need to set the **initial values** on the upper and the left boundary, $F(i, 0)$ and $F(0, j)$:

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



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

To obtain an *alignment* corresponding to this score, we still need to find the path of choices that has led the recursion to the final score. This is called a *traceback*[5]

However this is actually easy, as we only have to store one of the symbols

$$T(i, j) \in \{\leftarrow, \nwarrow, \uparrow\}$$

(or a subset thereof) whenever we assign a value to a "DP entry" $F(i, j)$.

3.19 Needleman-Wunsch algorithm

Input: two sequences x and y
Output: optimal alignment and score α

Initialization:
Set $F(0, 0) := 0$
Set $F(i, 0) := -i \cdot d$ and $T(i, 0) := (i - 1, 0)$ **for all** $i = 1, 2, \ldots, m$
Set $F(0, j) := -j \cdot d$ and $T(0, j) := (0, j - 1)$ **for all** $j = 1, 2, \ldots, n$
Recurrence:
for $i = 1, 2, \ldots, m$ **do**:
  for $j = 1, 2, \ldots, n$ **do**:
    Set $F(i,j) := \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$

[4] This corresponds to step 1 on the DP paradigm slides
[5] This corresponds to step 3 on the DP paradigm slides.

Set backtrace $T(i, j)$ to the maximizing pair (i', j') (encoded as $\in \{\leftarrow, \nwarrow, \uparrow\}$)
The best score is $\alpha := F(m, n)$
Traceback:
Set $(i, j) := (m, n)$
repeat
  if $T(i, j) = (i-1, j-1)$ **print** $\binom{x_i}{y_j}$
  else if $T(i, j) = (i-1, j)$ **print** $\binom{x_i}{-}$ **else print** $\binom{-}{y_j}$
  Set $(i, j) := T(i, j)$
until $(i, j) = (0, 0)$

3.20 An example of global 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 $d = 8$.

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**

DP matrix:

	H	E	A	G	A	W	G	H	E	E
	0 ←	**–8** ←	**–16** ←	–24 ←	–32 ←	–40 ←	–48 ←	–56 ←	–64 ←	–72 ←	–80
P	–8 ↑	–2 ↑↖	–9 ↑↖	**–17** ←	**–25** ↖	–33 ←	–42 ←	–49 ←	–57	–65	–73
A	–16 ↑	–10 ↑	–3 ↑↖	–4 ←	–12 ↖	**–20** ←	–28 ←	–36 ←	–44 ←	–52 ←	–60
W	–24 ↑↖	–18 ↑↖	–11 ↑↖	–6 ↖	–7 ↖	–15 ↖	**–5** ←	**–13** ←	–21 ←	–29 ←	–37
H	–32 ↑	–14 ↑↖	–18 ↑↖	–13 ↑↖	–8 ↑↖	–9 ↖	–13 ↖	–7 ↑↖	**–3** ←	–11 ←	–19
E	–40 ↑	–22 ↑	–8 ←	–16 ↑↖	–16 ↑↖	–9 ↖	–12 ↖	–15 ↖	–7 ↑	**3** ↑↖	–5
A	–48 ↑	–30 ↑	–16 ↑	–3 ←	–11 ↑↖	–11 ↖	–12 ↖	–12 ↖	–15 ↖	**–5** ↖	2
E	–56 ↑	–38 ↑	–24 ↑	–11 ↑	–6 ↑↖	–12 ↖	–14 ↖	–15 ↖	–12 ↖	–9 ↖	**1**

```
HEAGAWGHE-E
--P-AW-HEAE
```

Durbin *et al.* (1998)

3.21 Needleman-Wunsch Java applet

There is a nice Java applet illustrating the NW algorithm on the web:

`http://lectures.molgen.mpg.de/PracticalSection/AliApplet/index.html`

3.22 Complexity of the Needleman-Wunsch algorithm

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

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

3.23 Arbitrary gap costs

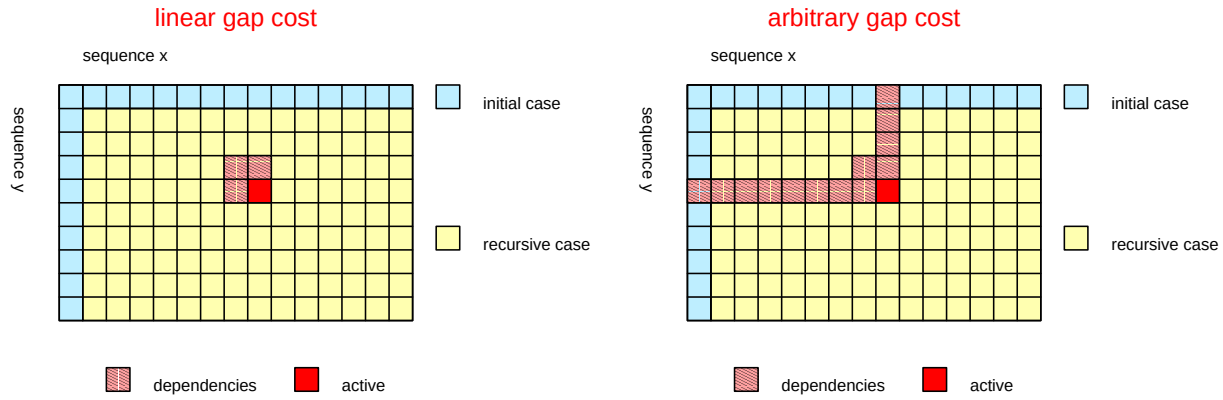
A way to deal with arbitrary gap costs is as follows. Assume a gap of length g has cost $\gamma(g)$. Then we can replace

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

with

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

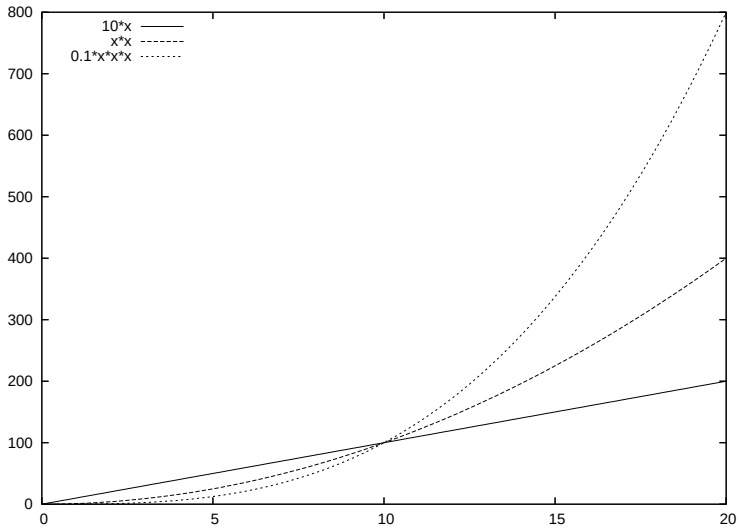
However this increases the running time from $O(mn)$ to $O(mn \max\{m,n\})$, since we have much more **dependencies in the DP recurrence**:



Remark: For *affine gap* costs there is are clever ways to do this in $O(mn)$.

3.24 Growth rates

For biological sequence analysis, we prefer algorithms that have time and space requirements that are linear in the length of the sequences. Quadratic time ($O(n^2)$) algorithms are a little slow, but feasible. Cubic time ($O(n^3)$) algorithms are feasible only for very short sequences.



3.25 Local alignment: Motivation

- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.
- Example:
  - *Homeobox genes*[6] 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

3.26 Local alignment: Motivation

- DNA toy example:
  - *Global Alignment*:

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

  - *Local Alignment*—better suited to find conserved segment:

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

3.27 Local alignment: Smith-Waterman algorithm

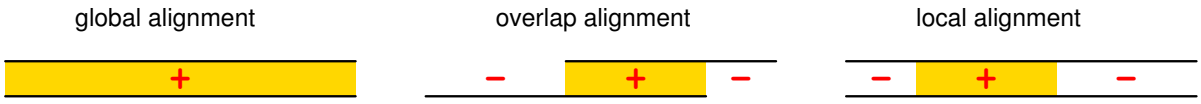
(Temple Smith and Mike Waterman, 1981)

A *local alignment* of two sequences x and y is a global alignment of a substring x' of x and a substring y' of y.

Why should we not like to include some prefixes $(x_1, \ldots, x_i)$ and $(y_1, \ldots, y_j)$ in a local alignment?

[6] regulate embryonic development

The only reason is that the best global alignment of $(x_1, \ldots, x_i)$ and $(y_1, \ldots, y_j)$ has a negative score.



Therefore we modify the recurrence formula for $F(i, j)$ such that we can *start a local alignment at any place in the DP matrix*. This means, we replace

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

with

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

Then any (i, j) can play the role that was reserved for $(0, 0)$ in the Needleman-Wunsch algorithm.

Likewise, we *start the* backtrace *at a position* (k, ℓ) *that maximizes* $F(k, \ell)$, not necessarily at (m, n). This is written as

$$(k, \ell) := \arg\max\{F(k, \ell) \mid k = 0, \ldots, m,\ \ell = 0, \ldots, n\}$$

This way, the local alignment can stop before it reaches the end of the sequences.

The backtrace stops when we reach a position (i, j) such that $F(i, j) = 0$.

3.28 Smith-Waterman algorithm

Input: two sequences x and y
Output: optimal local alignment and score α

Initialization:
Set $F(0, 0) := 0$
Set $F(i, 0) := 0$ and $T(i, 0) := (i - 1, 0)$ **for all** $i = 1, 2, \ldots, m$
Set $F(0, j) := 0$ and $T(0, j) := (0, j - 1)$ **for all** $j = 1, 2, \ldots, n$
Recurrence:
for $i = 1, 2, \ldots, m$ **do**:
  for $j = 1, 2, \ldots, n$ **do**:
    Set $F(i,j) := \max \begin{cases} 0 \\ F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$
    Set backtrace $T(i, j)$ to the maximizing pair (i', j') (encoded as $\in \{\leftarrow, \nwarrow, \uparrow\}$)
    or let it be `undefined` in the first case
Set $(k, \ell) := \arg\max\{F(k, \ell) \mid k = 0, \ldots, m,\ \ell = 0, \ldots, n\}$
The best score is $\alpha := F(k, \ell)$
Traceback:
Set $(i, j) := (k, \ell)$
repeat
  if $T(i, j) = (i - 1, j - 1)$ **print** $\binom{x_i}{y_j}$

else if $T(i, j) = (i - 1, j)$ **print** $\binom{x_i}{-}$ **else print** $\binom{-}{y_j}$
Set $(i, j) := T(i, j)$
until $F(i, j) = 0$

3.29 An example of local alignment

A local alignment matrix using $d = 8$ and BLOSUM50 scores:

		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	**5**	0	0	0	0	0
W	0	0	0	0	2	0	**20**	**12**	4	0	0
H	0	10	2	0	0	0	12	18	**22**	14	6
E	0	2	16	8	0	0	4	10	18	**28**	20
A	0	0	8	21	13	5	0	4	10	20	27
E	0	0	6	13	18	12	4	0	4	16	26

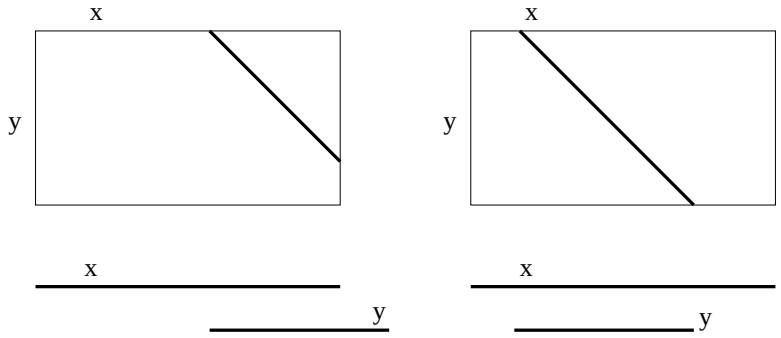
AWGHE
AW-HE

Durbin *et al.* (1998)

3.30 Algorithm for finding overlap alignments

We will now look at important *variations* of the pairwise alignment problem.

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

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

3.31 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								

3.32 Affine gap scores

We mentioend already that more complex gap score models might result in a cubic run time. However, for certain functions this can be avoided.

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.

3.33 Recursion for global alignment with affine gap costs

Initialization:

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

Recursion:

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

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

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

3.34 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						

3.35 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 for 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).

$y \setminus x$	0	1	…	u	…	n
0						
1						
…						
v				(u, v)		
…						
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).

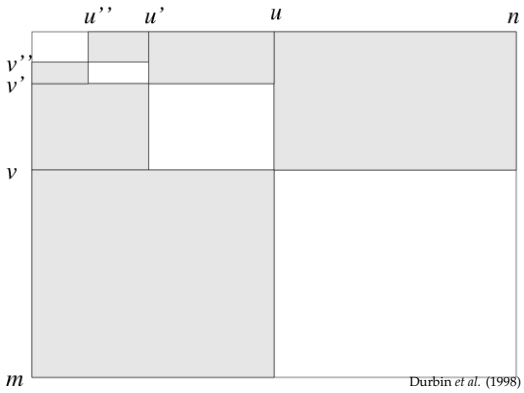
As we compute $F(i, j)$ column for column, we update $c(i, j)$:

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:



Durbin *et al.* (1998)

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 using the values of the c vector (exercise).

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!

3.36 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 $O(n)$.

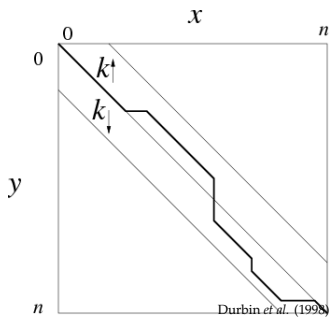
Now we ask:

- can we reduce the time complexity, too?

3.37 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?

3.38 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:

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

3.39 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.

3.40 Optimal alignments using KBand

Given two sequences x and y of the same length n. For simplicity, let M be a uniform 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. □

3.41 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 $k := 1$ (or some small number)
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.

3.42 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:

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

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!