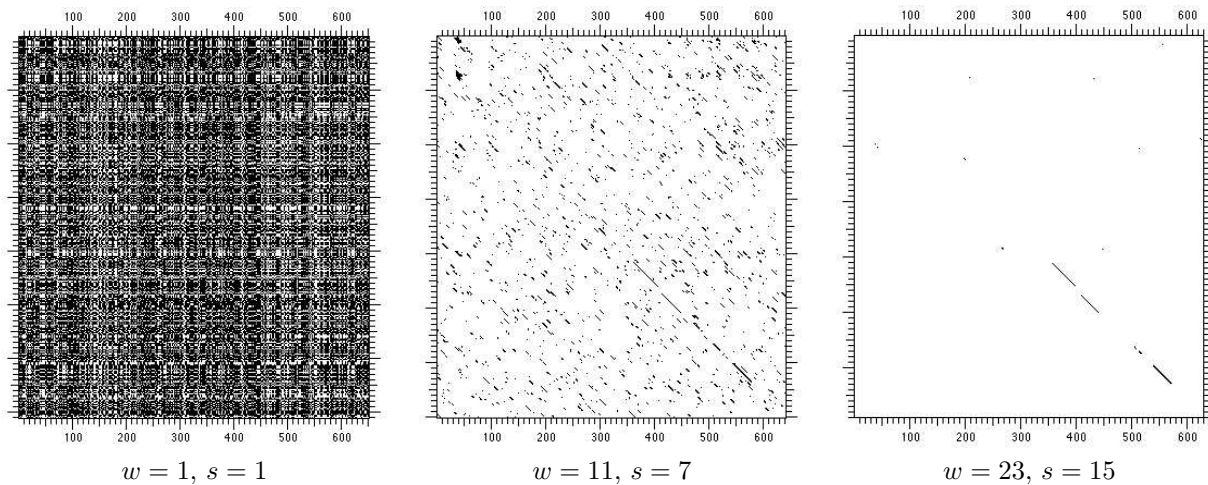
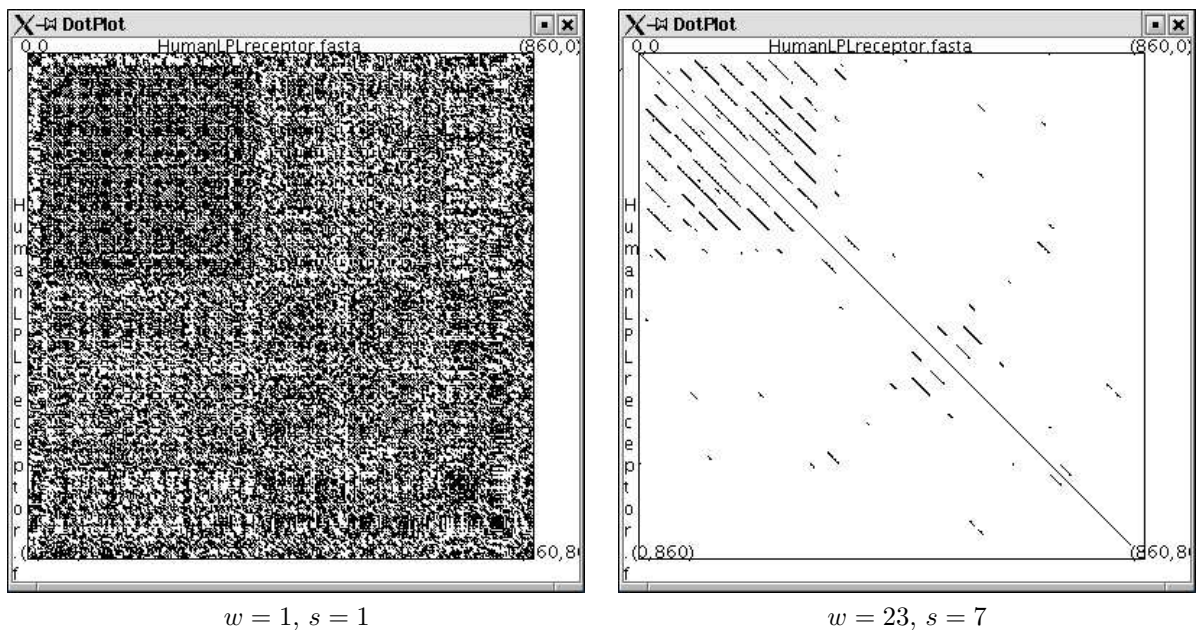


To obtain cleaner pictures, a *window size* w and a *stringency* s are used: A dot is only drawn at point (x, y) if within the next w positions at least s characters are equal.



2.4 Repeat detection using dot plots

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



2.5 Significance of alignments

- Alignment between very similar human alpha- and beta hemoglobins:


```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAKL
              G+ +VK+HGKKV  A++++AH+D++ +++++LS+LH  KL
HBB_HUMAN  GNPVKVAHGKKVLGAFSDGLAHLDLKGTFATLSELHCDKL
```
- Plausible alignment to leghaemoglobin from yellow lupin:


```
HBA_HUMAN  GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAKL
              ++ ++++H+ KV  + +A  ++                +L+ L+++H+ K
LGB2_LUPLU NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
```
- A spurious high-scoring alignment of human alpha globin to a nematode glutathione *S*-transferase homologue:

```

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

```

The middle row indicates identical amino acids with letters, and similar ones with a +.

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.

The sequence data are from SWISS-PROT.

2.6 Homology and homoplasy

The goal is to use similarity to uncover *homology*, while avoiding *homoplasy*. From <http://www.webref.org/anthropology/>:

homology: a similarity due to inheritance from a common ancestor.

homoplasy: a similarity that is not homologous. Homoplasy can arise from parallelism, convergence, analogy, and chance.

parallelism: a condition in which homoplastic similarities are found in related species that did not exist in the common ancestor. However, the common ancestor provided initial commonalities that gave direction to the evolution of the similarities.

convergence: the evolution of nonhomologous similarities in different evolutionary lines; the result of similarities in selective pressures.

analogies: structural similarities among organisms that have evolved not because of common ancestry but because of adaptations to similar environments (common functions) e.g. panda's thumb (wrist bone) and human's thumb (phalange).

2.7 Scoring schemes

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

Under the assumption that mutations at different sites occur independently of each other, an additive scoring scheme is appropriate:

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

Formally, the score of an alignment (x', y') is $\sum_i \delta(x'_i, y'_i)$, where $\delta: (\Sigma \cup \{-\})^2 \rightarrow \mathbb{R}$ is a *score matrix*.

This is often reasonable for DNA and proteins, but not for structural RNA, where base pairing introduces very important long-range dependences.

We will not explain the probabilistic background of the additive scoring scheme in *this* lecture, but simply take it as granted.

2.8 Hamming and edit distance

In the simplest case, we do not allow gaps at all and charge all mismatches at unit cost. This is called the *Hamming distance*.

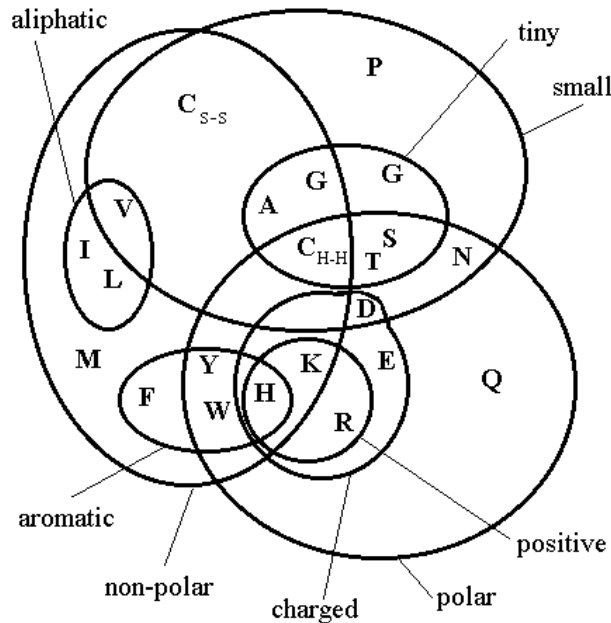
If we charge all insertions, deletions, and mismatches at unit cost, we obtain the *edit distance*.

Both scoring schemes have applications in biological sequence analysis, especially for nucleic acids, but generally it is better to

1. distinguish the type of a mismatch, and

- take the length of consecutive gaps into account.

2.9 Classification of amino acids



2.10 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	1

2.11 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*

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

Why can we apply such a recursion?

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

$$\begin{array}{c|c|c}
 x_i \text{ aligns to } y_j: & x_i \text{ aligns to a gap:} & y_j \text{ aligns to a gap:} \\
 \hline
 \begin{array}{c} \text{I G A } x_i \\ \text{L G V } y_j \end{array} & \begin{array}{c} \text{A I G A } x_i \\ \text{G V } y_j \text{ - -} \end{array} & \begin{array}{c} \text{G A } x_i \text{ - -} \\ \text{S L G V } y_j \end{array}
 \end{array}$$

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

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

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 $F(i, 0) = id$ for $i = 0, 1, \dots, m$ and $F(0, j) = jd$ for $j = 0, 1, \dots, n$.

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

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

This is actually easy, as we only have to store one of the symbols $T(i, j) \in \{\leftarrow, \nearrow, \uparrow\}$ (or a subset thereof) when we assign $F(i, j)$.

2.14 Needleman-Wunsch algorithm

Input: two sequences x and y

Output: optimal alignment and score α

Initialization:

Set $F(0, 0) := 0$.

Set $F(i, 0) := -id$ and $T(i, 0) := (i-1, 0)$ for all $i = 1, 2, \dots, m$

Set $F(0, j) := -jd$ and $T(0, j) := (0, j-1)$ for all $j = 1, 2, \dots, n$

Recurrence:

for $i = 1, 2, \dots, m$ **do:**

for $j = 1, 2, \dots, n$ **do:**

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

 Set backtrack $T(i, j)$ to the maximizing pair (i', j')

The best score is $\alpha := F(m, n)$

Set $(i, j) := (m, n)$

Traceback:

repeat

if $T(i, j) = (i-1, j-1)$ **print** $\begin{pmatrix} x_{i-1} \\ y_{j-1} \end{pmatrix}$

else if $T(i, j) = (i-1, j)$ **print** $\begin{pmatrix} x_{i-1} \\ - \end{pmatrix}$ **else print** $\begin{pmatrix} - \\ y_{j-1} \end{pmatrix}$

 Set $(i, j) := T(i, j)$

until $(i, j) = (0, 0)$.

2.15 An example of global alignment

We will use two short amino acid sequences for illustration:

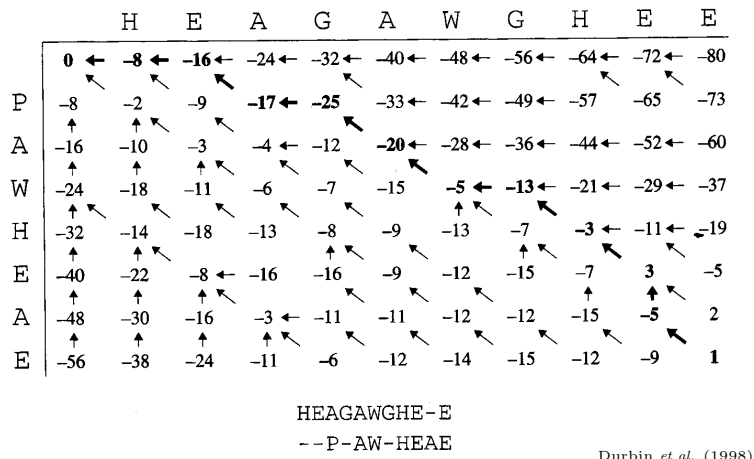
HEAGAWGHEE and PAWHEAE.

To score the alignment we will use the BLOSUM50 matrix and a gap cost of $d = 8$.

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

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



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

<http://lectures.molgen.mpg.de/PracticalSection/AlnApplet/index.html>

2.16 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: 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.

2.17 Computing the score in linear space

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, since we only need values from two columns of the matrix F at a time.

We can even come along with only one column and a few extra variables.

2.18 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

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

with

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

However this increases the running time from $O(mn)$ to $O(mn \max\{m, n\})$.

Note: For *affine gap* costs there is a clever way to do this in $O(mn)$. The idea is to use two more arrays and have a look at the second last column of an optimal alignment.

2.19 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 an infix x' of x and an infix y' of y .

The only reason why we would not like to include some prefixes (x_1, \dots, x_i) and (y_1, \dots, y_j) in a local alignment is that the best global alignment of (x_1, \dots, x_i) and (y_1, \dots, 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

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

with

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

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

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

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

This way, we do not worsen a local alignment by pulling it forth to the end.

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

2.20 Smith-Waterman algorithm

Input: two sequences x and y

Output: optimal alignment and score α

Initialization:

Set $F(0, 0) := 0$

Set $F(i, 0) := 0$ and $T(i, 0) := (i-1, 0)$ for all $i = 1, 2, \dots, m$

Set $F(0, j) := 0$ and $T(0, j) := (0, j-1)$ for all $j = 1, 2, \dots, n$

Recurrence:

for $i = 1, 2, \dots, m$ **do:**

for $j = 1, 2, \dots, n$ **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') ,
 or let it **undefined** in the first case

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

The best score is $\alpha := F(k, \ell)$

Set $(i, j) := (k, \ell)$

Traceback:

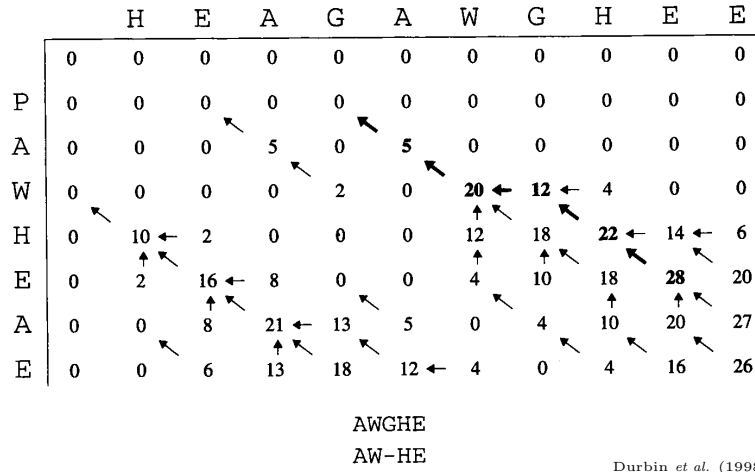
repeat

```

if  $T(i, j) = (i - 1, j - 1)$  print  $\begin{pmatrix} x_{i-1} \\ y_{j-1} \end{pmatrix}$ 
else if  $T(i, j) = (i - 1, j)$  print  $\begin{pmatrix} x_{i-1} \\ - \end{pmatrix}$  else print  $\begin{pmatrix} - \\ y_{j-1} \end{pmatrix}$ 
Set  $(i, j) := T(i, j)$ 
until  $F(i, j) = 0$ .
    
```

2.21 An example of local alignment

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



2.22 Alignment in linear space

We have already seen how to compute the *score* of an optimal alignment in linear space. This applies to global, end-gap free, and local alignment.

But how can we obtain the actual *alignment*, not just its score?

For simplicity we restrict ourselves to global alignments in what follows. The following overvation helps. Assume this is an optimal global alignment:

```

GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSSDLHAHKL
++ ++++H+ KV + +A ++ +L+ L+++H+ K
NNPELQAHAAGKVFVKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
    
```

Then we can split this alignment at any position and will obtain two optimal global alignments for both sides:

```

GSAQVKGHGKKVADAL          TNAVAHV---D--DMPNALSALSSDLHAHKL
++ ++++H+ KV              + +A ++ +L+ L+++H+ K
NNPELQAHAAGKVFVKLV      YEAAIQLQVTGVVVTDATLKNLGSVHVSKG
    
```

Recursively, we want to break the alignment problem into smaller pieces, until they become trivial. But since we do not have a global alignment yet, we do not know where to split the sequences.

Can we find the optimal split position in linear space?

This is the idea of *divide and conquer*: “Spend some effort to divide the problem into more manageable parts, then combine the final solution out of the solutions for the subproblems.”

Let $m := |x|$, $n := |y|$ and $u := \lfloor m/2 \rfloor$.

Idea: We will split the DP matrix at column u . We can find in linear space the row v where the global alignment backtrace crosses the u -th column of the DP matrix.

Then we know that the alignment passes through the cell (u, v) of the DP matrix and can split the problem of

finding the backtrace into a upper left and a lower right part:

	x_1	\cdots	x_{u-1}	x_u	x_{u+1}	\cdots	x_m
y_1	$F(0,0)$						
\vdots							
y_{v-1}							
y_v				$F(u,v)$			
y_{v+1}					$F(m,n)$		
\vdots							
y_n							

Then we concatenate the global alignments for both parts (columns 1 to u , and columns $u + 1$ to m).

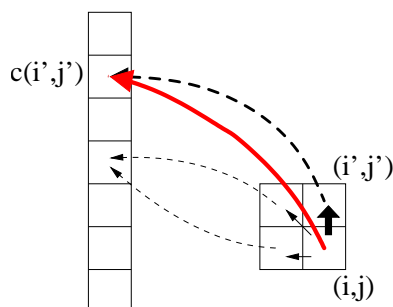
In order to find v when the calculation of $F(i, j)$ has reached $F(m, n)$, we cannot use a backtracking array like the T in the NW algorithm, as this would require $\Omega(mn)$ space. Instead we store pointers into the u -th column:

For $i > u$ and any j , let $c(i, j)$ be a row where a backtracking path from $(0, 0)$ to (i, j) goes through the u -th column.

2.23 Finding the row where to cut

The values $c(i, j)$, $i > u$, can be computed as follows: Let (i', j') be the cell from which $F(i, j)$ is obtained. Then

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



Note that $c(i, j)$ is determined by the values in the previous and current column, so this can be done in $O(n)$ space. The final value is $v = c(m, n)$.

Now we know (u, v) and can solve both subalignments by recursive calls of the same algorithm.

2.24 Time complexity of linear space alignment

Since we do not construct the backtrace at once, we need to recompute certain $F(i, j)$ values in recursive calls. However the total area of F which is recomputed in the next stage of subdivision is bounded by

$$(m + 1)(n + 1)/2 + n,$$

i.e., it is roughly halved; and this goes on for the subproblems.

Since no column is selected twice, the “ n ” terms add up to $O(mn)$ over the whole run of the algorithm.

The sum of the “ $(m + 1)(n + 1)/2$ ” terms over the whole run of the algorithm is bounded by $(m + 1)(n + 1) \sum_{i=0}^{\infty} 2^{-i} = 2(m + 1)(n + 1) = O(mn)$.

Therefore the total running time is $O(mn)$.