2 Pairwise alignment

We will discuss:

1. Strings
2. Dot matrix method for comparing sequences
3. Edit distance and alignment
4. The number of possible alignments
5. Scoring matrices
6. Global and local alignment of two sequences using dynamic programming

2.1 Strings

Definition 2.1.1 (Symbols and alphabet) *An* alphabet Σ *is a finite, non-empty set of* letters *or* symbols*. A* string S *on* Σ *is obtained by writing a finite sequence* $S = s_1 s_2 \dots s_n$ *of symbols from* Σ*. The* length $|S|$ *of a string* S *is the number of symbols in* S*.*

The alphabet for DNA is $\Sigma = \{\texttt{A},\texttt{G},\texttt{C},\texttt{T}\}$ and, for example,

$$S = \texttt{ATGGAATGCTAATAG}$$

is a string on Σ of length 15.

Note: we will use the words string and sequence as synonyms. Usually we will use the word "sequence" in a biological context (such as DNA and protein sequences) and "string" in more abstract settings.

Definition 2.1.2 (Concatenation) *Let* S *and* T *be two strings on* Σ*. We use* ST *to denote the* concatenation *of* S *and* T*.*

For example, if $S = \texttt{UNEX}$ and $T = \texttt{PECTED}$ then $ST = \texttt{UNEXPECTED}$.

Definition 2.1.3 (Substring) *Let* S *and* T *be two strings on* Σ*. We call* S *a* substring *of* T *if there exist strings* U, V *on* Σ *such that* $T = USV$*.*

We will use $s_i \dots s_j$ *to denote the* substring *of* $S = s_1 \dots s_n$ *that starts at position* $i \geq 1$ *and ends at* $j \leq n$*.*

Example:

$$\begin{array}{rll} T = & \texttt{ATGGAATGCTAATAG} & \text{then} \\ S = & \quad\;\;\texttt{AATGCT} & \text{is a substring of } T. \end{array}$$

Definition 2.1.4 (Reverse complement) *Let* $S = s_1 \dots s_n$ *be a sequence of length* n *on the DNA alphabet* $\Sigma = \{\texttt{A},\texttt{G},\texttt{C},\texttt{T}\}$*. The sequence* $\bar{S} = \bar{s}_1 \dots \bar{s}_n$ *with*

$$\bar{s}_i = \begin{cases} \texttt{A} & \text{if } s_{n-i+1} = \texttt{T} \\ \texttt{G} & \text{if } s_{n-i+1} = \texttt{C} \\ \texttt{T} & \text{if } s_{n-i+1} = \texttt{A} \\ \texttt{C} & \text{if } s_{n-i+1} = \texttt{G} \end{cases}$$

is called the reverse complement *of* S*.*

Example: $S = \texttt{ACTGTGACCAA}$ and $\bar{S} = \texttt{TTGGTCACAGT}$.

2.2 Sequence file format

The most widely used format for DNA and protein sequences is *FASTA* (pronounced fast-Ay).

- Single FASTA format: Each record has a header line that starts with the symbol ">" and contains an identifier for the sequence. The following lines contain the actual sequence. We ignore any spaces in the sequences.

Example (a protein reference sequence):

```
>OQZ03718.1 putative hydrazine hydrolase A subunit [Candidatus Brocadia sp. UTAMX1]
MSKRIIGGVMVSALIAGALVCGDIFASGNQVLTGGSKQGKALWTDYSGMSKEIQGPVDVVLFTQSPRTAKGDPYQNYPHY
VSEGSRIVSYNLKTKEIKVLTNDFASAFDPCTYWDGKKFAFAGIHKKGGGCQIWEMNIDGSGVRQMTDYKGTCRSPIYYA
AGSIEEGKGRIIWRDRYFEGDWKERGTVDKTGFIIFAGSPDGVMDEFHNPYAYNLFRLDTQGGHVMERITGHVLSGIEFP
...
```

- Multiple-FASTA-Format: A file in multiple-FASTA format contains a series of FASTA records.

Example (some sequencing reads):

```
>M00385:31:000000000-AVHNF:1:2107:22047:1190 1:N:0:NCAGTG
ACGGCTCGCAACGGCCAGACCNAGGCGACNGCCCNCCANGCCCAACTCGACGCCATCAACCGCCGCGGCAACGCCGAGTA
CAGCCGCCTCGTCGCCGAGGCCGACAGCGTGCACCAGAGCCAGACGGCCTTGGCCACCCAGTTCGACGCCCAGAT
>M00385:31:000000000-AVHNF:1:2107:22834:1191 1:N:0:NCAGTG
TTCCACCCGGGCATCGAAATCNGCCTCTANCGCGNACCNCAGCCGGGACTCCATCCAAAGTCGCTGGCCAGGCTCGCCGC
AGACCAGGAAGGAGAGCGACCCCGTGCGGACCACCAGCGTGAGGTCGCGGATCAGCCCGGAATCATCGAGGGCCA
>M00385:31:000000000-AVHNF:1:2107:20737:1193 1:N:0:NCAGTG
CTTTGGTTGTTAAAATGAGTANTATATTGNAAAANAGANTGACTGTTACTGCCTTAATATCTGCTGTCGCTAGCAAGATT
AACAGAATTAACAGGGAAATTCGGTTATTGCAGGTAAAGTGGCAAATGAGTCCTGTATATATTTGTGAAATGGCC
>M00385:31:000000000-AVHNF:1:2107:19619:1195 1:N:0:NCAGTG
GCAGTTCGGCGCTGGTGTTGANGCAGACGNCTTCNGTGNCGAACGCCGCTGGCGCCAGGCCGAGGGTGGGCAGCAGGCAG
GCGAGGATACGGGCAACGAGCGCGGATGCGCGATACGGTGATGACATGGCGATCGACTCCGGACGGGATCAGGGA
...
```

2.3 Dot matrix sequence comparison

A dot matrix analysis is a very simple method for comparing two sequences. An $(n \times m)$ matrix relating two sequences of length n and m repectively is produced by placing a dot at each cell for which the corresponding symbols match. Here is an example for the two sequences

IMISSMISSISSIPPI and MYMISSISAHIPPIE:

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

Definition 2.3.1 (Dot matrix) *Let $S = s_1 s_2 \ldots s_n$ and $T = t_1 \ldots t_m$ be two strings of length n and m respectively. A* (simple) dot matrix *is an $n \times m$ matrix M defined as:*

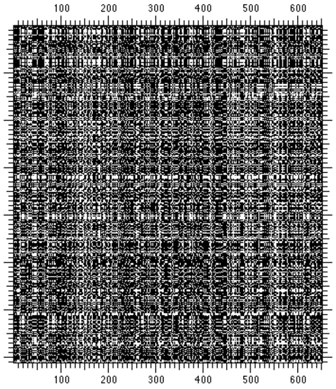
$$M[i,j] = \begin{cases} 1 & \text{for } s_i = t_j \\ 0 & \text{else,} \end{cases}$$

for all i, j with $1 \leq i \leq n,\ 1 \leq j \leq m$.

Note: The longest common substring within the two strings S and T is then the longest matrix subdiagonal containing only 1s. However, rather than *drawing* the digit 1 we draw a *dot*. A dot plot makes it easy to find:

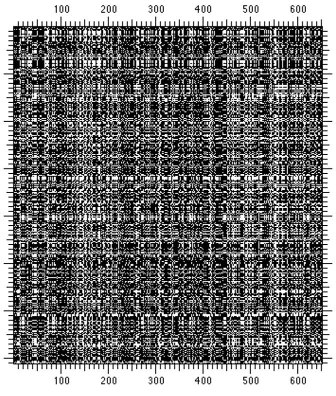
- Common substrings, they appear as contiguous dots along a diagonal
- Reversed common substrings
- Displaced common substrings
- Repeated common substrings

Example: DNA sequences which encode the Bacteriophage lambda and Bacteriophage P22 repressor proteins:

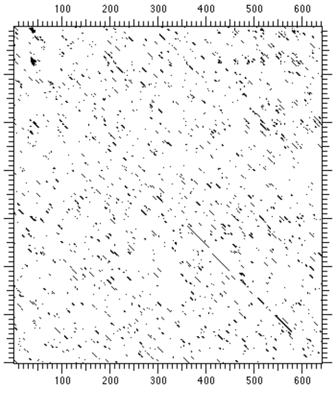


Dot plots of biological sequences will usually contain lots of dots, many of which are considered noise. To reduce the noise, a *window size w* and a *stringency s* are used and a dot is only drawn at point (i,j) if the next w positions have s or more equal characters.

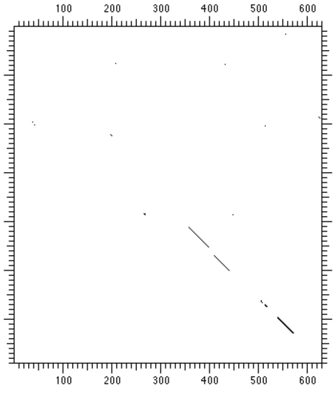
This identifies short stretches of matching sequence.



$w = 1, s = 1$

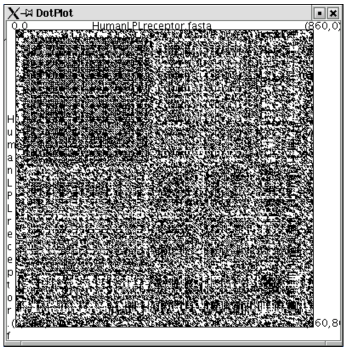


$w = 11,\ s = 7$

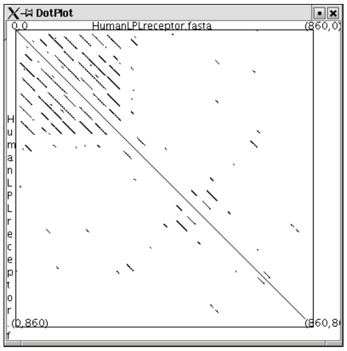


$w = 23,\ s = 15$

Dot matrix analysis of the human LDL receptor protein against itself:



$w = 1$, $s = 1$



$w = 23$, $s = 7$

The dot plot on the right reveals many repeats in the first 300 positions.

Limitations of dot plots:

- Only a visualization tool...
- No score to quantify identical or similar substrings
- Runtime is quadratic; more efficient algorithms to identify identical substrings exist (as we will see)

2.4 Comparing sequences

The comparison of biological sequences is one of the most important operations in computational biology.

Fundamental assumption: High sequence similarity implies similar structure and/or function

In an *alignment* of two sequences: place one sequence above the other one such that similar or identical characters are in the same column and non-identical/non-similar characters are either placed in the same column as a mismatch or opposite a gap in the other sequence.

Example:

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

Definition 2.4.1 (Pairwise alignment) *Given two sequences X and Y on an alphabet Σ. An* align- *ment A of X and Y is obtained by inserting dashes ('–') so that both resulting sequences X' and Y' are of equal length. They are then written one above the other such that each member of one sequence is in line, "aligned", with exactly one member of the other sequence.*

Usually we require that no two dashes are aligned with each other.

Example:

```
X=  Y  E  -  S  T  E  R  D  A  Y
Y=  -  E  A  S  T  E  R  S  -  -
```

is one alignment, another one is

```
X=  Y  E  S  T  E  R  D  A  Y
Y=  -  E  -  A  S  T  E  R  S
```

There are many more, which one is the "best" alignment?

In order to evaluate an alignment we need a *scoring scheme.*

There are two types, *distance scores* and *similarity scores.*

We will first look at the *edit distance* score which is used for strings and then will consider similarity scores that are used for molecular sequences.

2.4.1 Edit or Levenshtein distance

One way of defining the distance between two strings is based on the number of *edit operations* needed to transform one string into the other:

Definition 2.4.2 (Edit distance) *The* edit distance *(also known as* Levenshtein distance*)* d_{edit} *between two sequences* X *and* Y *is the minimum number of* edit operations *of type*

$$\left\{ \begin{array}{c} \textit{\underline{R}eplacement,} \\ \textit{\underline{I}nsertion, or} \\ \textit{\underline{D}eletion,} \end{array} \right\}$$

that are required to transform sequence X *into sequence* Y*:*

$$d_{edit}(X,Y) = \min\{R(X,Y) + I(X,Y) + D(X,Y)\}$$

Using `M` *for match, an* edit transcript *is a string on the alphabet* `I, D, R, M` *that describes a transformation of* X *to* Y*.*

Example: Given two strings $\begin{array}{rl} X = & \texttt{YESTERDAY} \\ Y = & \texttt{EASTERS} \end{array}$.

Here is a minimum *edit transcript* for the above example:

edit transcript=	D	M	I	M	M	M	M	R	D	D
X=	Y	E		S	T	E	R	D	A	Y
Y=		E	A	S	T	E	R	S		

The edit distance $d_{edit}(X,Y)$ of X, Y is 5.

Edit transcripts and alignments are mathematically *equivalent* ways of describing a relationship between two strings.

However, an edit transcript implies a set of putative *mutational events*, whereas an alignment presents a *static* picture of the relationship.

To obtain an algorithm for computing the edit distance, it is best to aim at computing an optimal alignment.

2.4.2 Dynamic programming calculation of edit distance

Given two sequences $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$. We want to compute the edit distance $d_{edit}(X,Y)$ between X and Y.

Let $D(i,j)$ denote the edit distance of the two prefixes $x_1 \dots x_i$ and $y_1 \dots y_j$.

We want to obtain $d_{edit}(X,Y) = D(n,m)$ by computing $D(i,j)$ for all pairs of prefixes $x_1 \dots x_i$ and $y_1 \dots y_j$.

We will use the standard *dynamic programming* approach.

The main idea is to recursively build an optimal solution for a larger instance of the problem from optimal solutions of smaller instances of the problem.

Dynamic programming has three essential components:

- the recurrence relation,
- the tabular computation, and
- the traceback.

The recursion is computed in tabular form:

$$
\begin{array}{r|ccccccccc|}
 & 0 & y_1 & y_2 & y_3 & \dots & y_{j-1} & & y_j & \dots \;\; y_m \\
\hline
0 & D(0,0) & & & & & & & | & \\
x_1 & & & & & & & & | & \\
x_2 & & & & & & & & | & \\
x_3 & & & & & & & & | & \\
 & & & & & & & & | & \\
\dots & & & & & & & & | & \\
x_{i-1} & & & & & & D(i-1,j-1) & & D(i-1,j) & \\
 & & & & & & & \searrow & \downarrow & \\
x_i & - & - & - & - & - & D(i,j-1) & \rightarrow & \boxed{D(i,j)} & \\
 & & & & & & & & & \\
\dots & & & & & & & & & \\
 & & & & & & & & & \\
x_n & & & & & & & & & \\
\hline
\end{array}
$$

2.4.3 The recurrence relation

The recurrence relation for $D(i,j)$ when both i and j are positive, is given by

$$D(i,j) = \min \left\{ \begin{array}{l} D(i,j-1)+1 \\ D(i-1,j)+1 \\ D(i-1,j-1)+\Delta(i,j) \end{array} \right\},$$

where $\Delta(i,j) = \begin{cases} 0 & \text{if } x_i = y_j, \\ 1 & \text{else.} \end{cases}$

The recurrence relation determines how we can obtain the value for $D(i,j)$ from values for smaller indices. When there are no smaller indices, then we must explicitly state *base conditions*:

- We set $D(i,0) = i$ for all $0 \leq i \leq n$. This corresponds to an alignment in which the first i characters of X are aligned to the left of the first character of Y.

$$\begin{array}{ccccc} x_1 & x_2 & \dots & x_i & x_{i+1} \\ - & - & \dots & - & y_1 \end{array}$$

- We set $D(0,j) = j$ for all $1 \leq j \leq m$. This corresponds to an alignment in which the first j characters of Y occur to the left of the first character of X.

$$\begin{array}{ccccc} - & - & \dots & - & x_1 \\ y_1 & y_2 & \dots & y_j & y_{j+1} \end{array}$$

2.4.4 Example

We can compute the edit score between sequences `ATTAC` and `GATTAG` by filling the following matrix:

D	0	G	A	T	T	A	G
0							
A							
T							
T							
A							
C							

Distance:

2.4.5 Proof of the recurrence relation

Theorem 2.4.3 (Correctness of edit distance recursion) *The above recurrence relation correctly computes the minimum edit distance.*

Proof

We will show:

$$d_{edit}(x_1 \dots x_i, y_1 \dots y_j) = D(i,j) = \min \left\{ \begin{array}{c} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1)+\Delta(i,j) \end{array} \right\}.$$

Induction start: $d_{edit}(\epsilon, \epsilon) = 0 = D(0,0)$, ok.

Induction step: Assume $d_{edit}(x_1 \dots x_{i'}, y_1 \dots y_{j'}) = D(i', j')$ for all $i' \leq i$ and $j' \leq j$ with either $i' \neq i$ or $j' \neq j$.

Given a minimum edit transcript $t_1 \dots t_r$ for $x_1 \dots x_i$ to $y_1 \dots y_j$. There are four possibilities for the last symbol t_r:

'I' Last operation was to insert y_j at the end of the first string. Then $t_1 \dots t_{r-1}$ must be a minimum edit transcript for $x_1 \dots x_i$ to $y_1 \dots y_{j-1}$.
(Otherwise, we could replace $t_1, \dots, t_{r-1}$ by a better transcript to obtain a smaller edit distance between $x_1 \dots x_i$ and $y_1 \dots y_j$.)
By induction, it contains $D(i, j-1)$ edit operations, thus $d_{edit}(x_1 \dots x_i, y_1 \dots y_j) = D(i, j-1)+1$.

'D' Last operation was to delete x_i. Then $t_1 \dots t_{r-1}$ must be a minimum edit transcript for $x_1 \dots x_{i-1}$ to $y_1 \dots y_j$ (as above).
By induction, it contains $D(i-1, j)$ edit operations, thus $d_{edit}(x_1 \dots x_i, y_1 \dots y_j) = D(i-1, j)+1$.

'R','M' The last operation was to replace or match x_i with y_j. Then $t_1 \dots t_{r-1}$ must be a minimum edit transcript for $x_1 \dots x_{i-1}$ to $y_1 \dots y_{j-1}$ (as above).
By induction, it contains $D(i-1, j-1)$ edit operations. Thus, $d_{edit}(x_1 \dots x_i, y_1 \dots y_j) = D(i-1, j) + \Delta(i,j)$.

The recursion takes the minimum of all four scores, so we must show that all four can be achieved:

- There exists a transcript with score $D(i, j-1) + 1$:
  transform $x_1 \dots x_i$ to $y_1 \dots y_{j-1}$ using $D(i, j-1)$ edit operations, then use one more to insert y_j.
- There exists a transcript with score $D(i-1, j) + 1$:
  transform $x_1 \dots x_{i-1}$ to $y_1 \dots y_j$ using $D(i-1, j)$ edit operations, then use one more to delete x_i.
- There exists a transcript with score $D(i-1, j-1) + \Delta(x_i, y_j)$:
  transform $x_1 \dots x_{i-1}$ to $y_1 \dots y_{j-1}$ using $D(i-1, j-1)$ edit operations, then replace or match x_i with y_j, using one or zero more edit operations, respectively. □

2.4.6 Traceback

How to obtain an actual edit transcript that corresponds to the edit distance?

Starting at the last cell in the table, "trace-back" through the table via the predecessor cells that gave rise to the values of the cells.

D	0	G	A	T	T	A	G
0	0	← 1	2	3	4	5	6
A	1	1	↖ 1	2	3	4	5
T	2	2	2	↖ 1	2	3	4
T	3	3	3	2	↖ 1	2	3
A	4	4	3	3	2	↖ 1	2
C	5	5	4	4	3	2	↖ 2

Edit transcript:	D	M	M	M	M	R
Alignment:	G	A	T	T	A	G
	-	A	T	T	A	C

2.5 Sequence similarity

与ED不同，该得分越高，相似度越高。

We have seen how to express string relatedness using the edit distance. In biology, we are usually interested in *similarity* rather than distance.

Definition 2.5.1 (Similarity score) *Given two sequences $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$ on an alphabet Σ. A* similarity score *matrix $s : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \to \mathbb{R}$ assigns a similarity score to each pair of characters in $\Sigma \cup \{-\}$.*

Let A be an alignment of X and Y:

$X' = x'_1 \dots x'_l$ and $Y' = y'_1 \dots y'_l$

denote the two strings obtained after inserting gaps, both of length l.

The *score* $S(A)$ of A is defined as

$$S(A) = \sum_{i=1}^{l} s(x'_i, y'_i).$$

Example: for $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{L}, -\}$ we use the similarity score matrix:

s	A	B	L	$-$
A	3	1	-1	-2
B		2	0	-3
L			1	2
$-$				0

Example of an alignment and the calculation of its score:

X' =	B	L	A	-	B	L	A	
Y' =	A	L	A	B	B	L	-	
	1	+1	+3	-3	+2	+1	-2	= 3

2.5.1 Pairwise alignment: Example

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

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

Here there are many positions at which the two corresponding residues are identical. Many others are functionally conserved, e.g. the D-E pair, both negatively charged amino acids.

- Plausible alignment of human alpha globin to Leghemoglobin-2 - Lupinus luteus (European yellow lupin):

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

These two proteins are known to be evolutionarily related, have the same 3D structure and both have the same function, thus this is a biologically meaningful alignment.

Note, however, that there are only a few identities and many gaps.

- An alignment of human alpha globin to a nematode glutathione S-transferase homologue:

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

This alignment has a similar number of identities and conservative changes as in the previous example.

However, this is an alignment between two proteins that have completely different structure and function, and is thus considered a random chance alignment.

2.5.2 Substitution matrices

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

Definition 2.5.2 (Substitution matrix) *A substitution matrix S on an alphabet $\Sigma = \{a_1, \ldots, a_\kappa\}$ has $\kappa \times \kappa$ entries, where each entry (i, j) assigns a score for a substitution of the letter a_i by the letter a_j in an alignment.*

A substitution matrix can be generated as follows:

- Consider a database of high quality non-gapped alignments.
- Compute the frequency $f(a_i)$ of each symbol a_i and the frequency $f(a_i, a_j)$ of each substitution $a_i \to a_j$ in the database.
- Based on this counts, compare a *null/random model* against a *match model.*

Consider non-gapped alignments of the form:

$$
\begin{aligned}
X &= x_1 x_2 \ldots x_n \\
Y &= y_1 y_2 \ldots y_n
\end{aligned}
$$

The *null hypothesis* is: The two sequences are unrelated (not homologous). So, the alignment is then random with a probability described by a *random model* R: Each letter a occurs independently with some probability p_a, and the probability of the two sequences is the product:

$$P(X, Y \mid R) = \prod_i p_{x_i} \prod_j p_{y_j}.$$

$p_{x_i} \approx f(x_i)$

The alternative hypothesis is the *match* model M: The two sequences are related (homologous). Aligned pairs of residues occur with a joint probability p_{ab}, the probability that a and b have each evolved from some (unknown) common ancestor residue c. In this case, the probability for the whole alignment is:

Log Normalization $$P(X, Y \mid M) = \prod_i p_{x_i y_i}.$$ 为了避免数字过小

The ratio of the two gives a measure of the relative likelihood that the sequences are related (model M) as opposed to being unrelated (model R). This ratio is called the *odds ratio*:

$$\frac{P(X, Y \mid M)}{P(X, Y \mid R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i p_{x_i} \prod_i p_{y_i}} = \prod_i \frac{p_{x_i y_i}}{p_{x_i} p_{y_i}}$$

To obtain an additive scoring scheme, we take the logarithm (base 2 is usually chosen) to get the *log-odds ratio:*

$$\log(\frac{P(X, Y \mid M)}{P(X, Y \mid R)}) = \log(\prod_i \frac{p_{x_i y_i}}{p_{x_i} p_{y_i}}) = \boxed{\sum_i s(x_i, y_i),}$$

with 若 P 取 Match 若 n 取 Random

$$\boxed{s(a, b) = \log\left(\frac{p_{ab}}{p_a p_b}\right).}$$

In this approach, a matrix $S = s(a, b)$ determines the 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.

2.5.3 BLOSUM matrices

A popular family of substitution matrices are the *BLOSUM* (=BLOcks SUbstitution Matrix) matrices. These were empirically derived as described above from the BLOCKS database [1] (Henikoff and Henikoff, 1992)

Blocks are multiply aligned ungapped segments corresponding to the most highly conserved regions of proteins. Here is an example:

```
Block IPB000104A

ID   ANTIFREEZEI; BLOCK
AC   IPB000104A; distance from previous block=(1,51)
DE   Type I antifreeze protein signature
BL   PR00308;   width=15; seqs=10; 99.5%=863; strength=1222
ANP4_PSEAM|P02734    (  45) TAATAAAAAAAATAAT  45
ANPX_PSEAM|P07835    (  51) TAATAAAAAAAATAAT  45
ANPY_PSEAM|P23699    (  51) TAATAAAAAAAATAVT  59
Q99013|ANPB_PSEAM    (  46) TASDAAAAAALTAAN  41
ANP4_PSEAM|P02734    (  23) TASDAAAAAAATAAT  38
ANPA_PSEAM|P04002    (  46) TASDAAAAAALTAAN  41
ANP3_PSEAM|P02733    (   2) TASDAAAAAALTAAB  46
ANP_LIMFE|P09031     (  50) TASDAAAAAATAAA  45

Q547T1|Q547T1_PSEAM  (  40) TASDAAAAAAATAAT  38
Q7SIC4_PSEAM         (   2) VASDAKAAAELVAAN 100
```

[1] See http://blocks.fhcrc.org/, no longer updated

Different members of the BLOSUM family of matrices were created by considering alignments of sequences with different levels of identity.

For example, the most commonly used matrix is BLOSUM62. To create this matrix, sequences are clustered so that any two sequences of $\geq 62\%$ sequence identity are placed in the same cluster and all counts are based on comparisons between pairs of sequences contained in different clusters.

BLOSUM matrices are scaled so that their values are in half-bits, that is, the log-odds is reported as $s'(a,b) = 2 \times \log_2\left(\frac{p_{ab}}{p_a p_b}\right)$, rounded to the nearest integer value.

Here is the BLOSUM62 matrix:

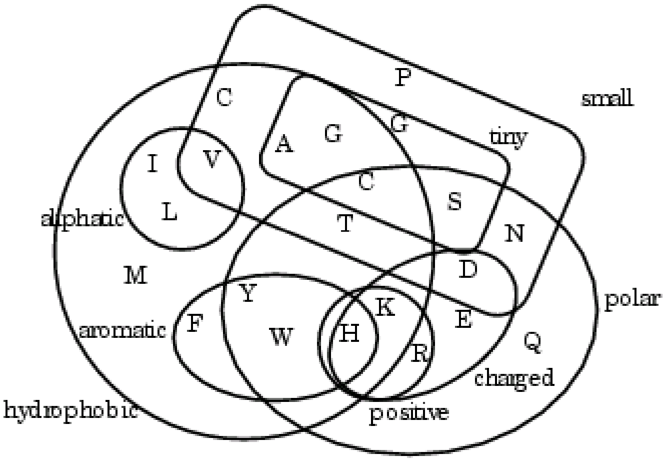
至少要 62% 相似

还有 BLOSUM 80

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

2.5.4 Classification of amino acids

Notice that the values in the BLOSUM62 matrix reflect some of the similarities between different amino-acids shown here:



based on chemical property but not only

举例：I U V

The 20 common amino acids:

Amino Acid	3-Letter	1-Letter	Side chain polarity	Side chain charge (pH 7)	Hydropathy index
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

2.5.5 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

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

倘若假设两基因同源可用

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
```

4次基因丢失

Affine gap penalty:

```
GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
GSAQVKGHGKKVADA---------------SALSDLHAHKL
```

1次 ~

所以下面 Preferred

2.6 The number of possible alignments

How many different gapped alignments are possible between two sequences $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_m$?

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

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
```

$$\binom{8}{6}$$

Here the subsequence is:

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

Lemma 2.6.1 (Number of alignments) *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 $r \in \{0, 1, \dots, \min(n,m)\}$: the number of ordered selections of $i_1, i_2, \dots, i_r$ in $1, 2, \dots, n$ is $\binom{n}{r}$ and the number of ordered selections of $j_1, j_2, \dots, j_r$ in $1, 2, \dots, m$ 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, $N(1000, 1000) \approx 10^{600}$.

2.7 Alignment algorithms

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

As in the case of edit distance-based alignments, we will discuss similarity-based alignment algorithms that employ *dynamic programming*. 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 used that usually perform very well in practice, (although they might sometimes miss the best possible alignment).

In particular we will study an algorithm for the computation of a *global alignment* and one for the computation of a *local alignment*.

2.7.1 Global alignment and Needleman-Wunsch

The Needleman-Wunsch algorithm[2] is a dynamic program that solves the problem of obtaining the best *global* alignment of two sequences.

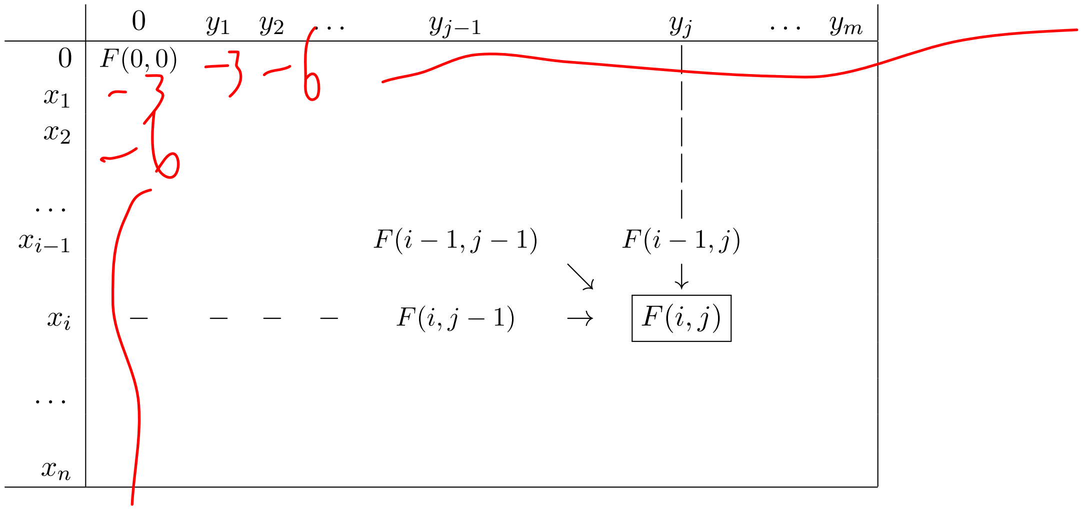
Idea: Build an optimal alignment from optimal alignments of prefixes.

Assume that we are given two sequences $x = x_1x_2\dots x_n$ and $y = y_1y_2\dots y_m$, a scoring matrix $s(\cdot,\cdot)$ and a gap penalty d. We will compute a matrix

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

in which $F(i,j)$ equals the best score of the alignment of the two prefixes $x_1x_2\dots x_i$ and $y_1y_2\dots y_j$.

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 $F(i,j-1)$:



2.7.2 The recursion

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

x_i aligns to y_j:	x_i aligns to a gap:	y_j aligns to a gap:
`A G A` x_i `A G G` y_j	`A A G A` x_i `A G G` y_j `-`	`T G A` x_i `-` `T G G C` y_j

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

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

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

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

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 using *traceback*.

2.7.3 Example of a global alignment matrix

Needleman-Wunsch matrix for the sequences ATTAC and GATTAG, scoring values $s(a, a) = 1$, $s(a, b) = -1$ and a linear gap cost of $d = 2$:

F	0	G	A	T	T	A	G
0	0	-2	-4	-6	-8	-10	-12
A	-2	-1	-1	-3	-5	-7	-9
T	-4	-3	-2	0	-2	-4	-6
T	-6	-5	-4	-1	1	-1	-3
A	-8	-7	-4	-3	-1	2	0
C	-10	-9	-6	-5	-3	0	1

$(m+1)+(n+1)$

$+ 4 \cdot m \cdot n$

Score: ; Alignment:

2.7.4 Needleman-Wunsch algorithm

```
Input: two sequences $X$ and $Y$
Output: optimal alignment and score $\alpha$
Initialization: Set $F(i,0) = -i \cdot d$ for all $i = 0, 1, 2, \ldots, n$
Set $F(0,j) = -j \cdot d$ for all $j = 0, 1, 2, \ldots, m$
For $i = 1, 2, \ldots, n$ do:
    For $j = 1, 2, \ldots, m$ do:
        Set $F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$
        Set traceback $T(i,j)$ to the maximizing pair $(i', j')$
The best score is $\alpha = F(n, m)$

Set $(i,j) = (n,m)$
repeat (Comment: prints out alignment in reverse order)
    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 $(i,j) = (0,0)$.
```

2.7.5 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 additions and a max.

Hence, the algorithm requires $\boxed{O(nm)}$ time and memory. O(n) 是一个函数集合

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.

存在 $C \in R$，使 $C \cdot nm >$ 需要计算的次数

2.7.6 Local alignment and Smith-Waterman

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 *substrings* of both. For example, we may want to find a motif shared by two different genes:

```
ACCATGTGCA|CTGCGCTAATCCAGGCA|CATTACCGAATCCGGGATTCACACCACA
     CTCTC|CTGCGCTAATCCAGGCA|ACCGTTC
```

Here the score of an alignment between two substrings would be larger than the score of an alignment between the full lengths strings.

The Smith-Waterman[3] 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 implies for the base conditions: set $F(i,0) =$ ____ and $F(0,j) =$ ____ for all $i = 0,1,2,\dots,n$ and $j = 0,1,2,\dots,m$.

(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

$$\sum_{a,b\in\Sigma} p_a \cdot p_b \cdot s(a,b) < 0,$$

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

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

Smith-Waterman matrix of the sequences GATTAG and ATTAC with $s(a,a) = 1$, $s(a,b) = -1$ and $s(a,-) = s(-,a) = -2$:

F	0	G	A	T	T	A	G
0	0	0	0	0	0	0	0
A	0	0	1	0	0	1	0
T	0	0	0	2	1	0	0
T	0	0	0	1	3	1	0
A	0	0	1	0	0	4	2
C	0	0	0	0	0	2	3

遇到0就

break

Score: ____; Alignment =

[3] Smith, T. and M. Waterman, Identification of common molecular subsequences. J. Mol. Biol. 147:195-197, 1981

2.7.7 Smith-Waterman algorithm

```
Input: two sequences $X$ and $Y$
Output: optimal local alignment and score $\alpha$
Initialization: Set $F(i,0) = 0$ for all $i = 0, 1, 2, \dots, n$
Set $F(0,j) = 0$ for all $j = 1, 2, \dots, m$
For $i = 1, 2, \dots, n$ do:
    For $j = 1, 2, \dots, m$ 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 $(i', j')$
Set $(i,j) = \arg\max\{F(i,j) \mid i = 1, 2, \dots, n, j = 1, 2, \dots, m\}$
The best score is $\alpha = F(i,j)$
repeat (Comment: prints out alignment in reverse order)
    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 $F(i,j) = 0$.
```

2.8 Summary

We have discussed:

- the dot matrix for visual comparison of two sequences,
- the edit distance and the dynamic programming algorithm for its computation,
- global alignments and the Needleman-Wunsch algorithm, and
- local alignments and the Smith-Waterman algorithm.