PAIR-WISE SEQUENCE ALIGNMENT ALGORITHMS

Why compare sequences?

- Lots of sequences with unknown structure or function, but, few sequences with known structure or function

  1. If they align, they are similar, maybe due to common descent

  2. If they are similar, then they may have same structure or function

  3. If one of them has known structure or function, then alignment to the other yields insight about how structure or function works.

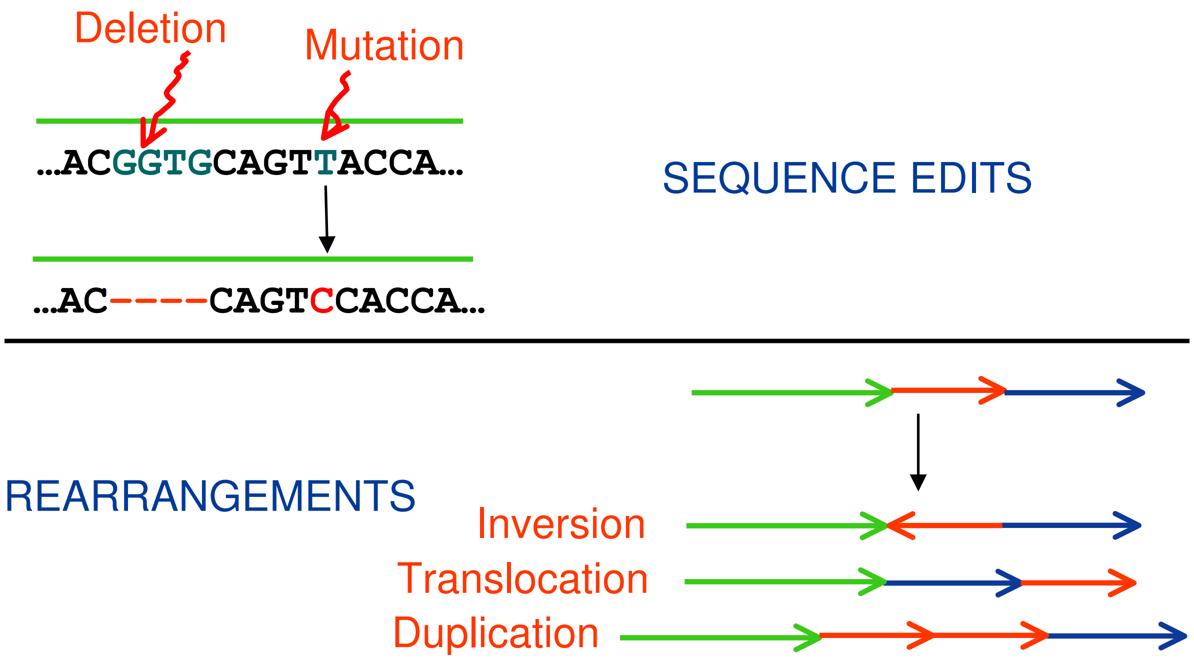
- Searching databases for related sequences

- Exploring evolutionary relationships among species

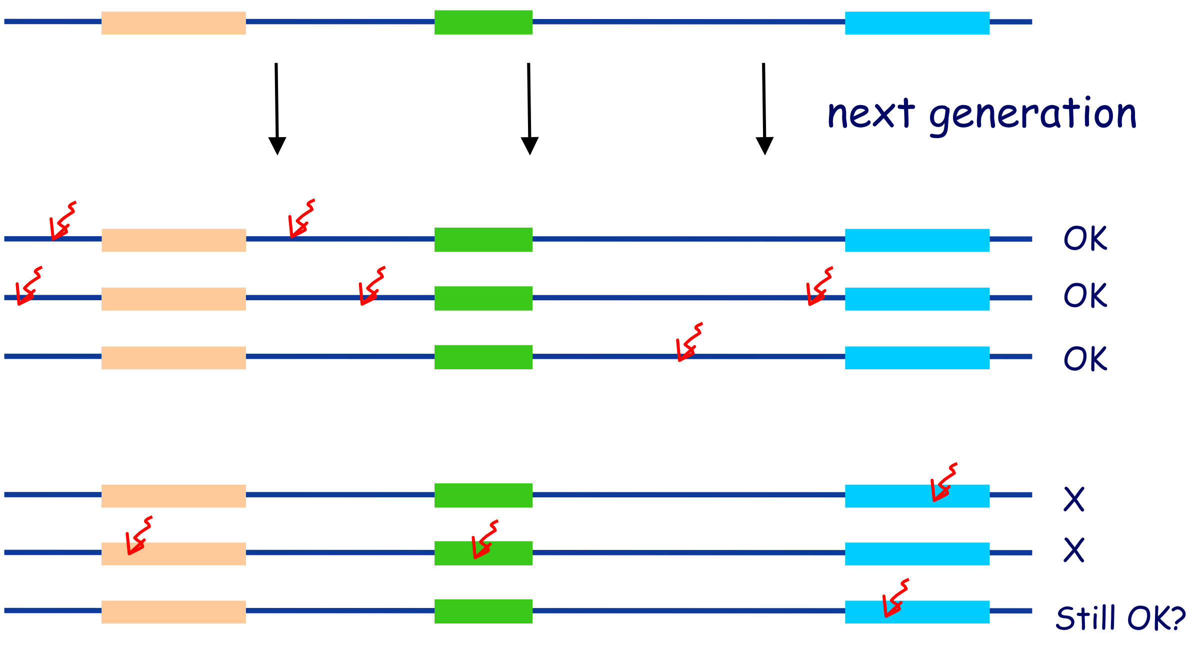
- Finding informative elements in sequences

- . . .

Evolution at the DNA Level



Evolutionary Rates



Alignment is the key to

- finding important regions
- determining function

  Sequence conservation implies functions
- uncovering the evolutionary forces

Distance and Similarity

All contemporary genetic material have one common ancestral DNA

Differences between families of contemporary species are due to *local mutations* during the course of evolution

Local mutations

- *Insertion* of one or more bases
- *Deletion* of one or more bases
- *Substitution* of one or more bases

Insertion and deletion are reverse of each other. They are called *indel*

Distance Given two sequences: the minimal sum of weights for a set of mutations transforming one into the other.

Similarity Given two sequences: the maximal sum of weights corresponding to resemblance of the two sequences

Simplest Model: Edit Distance

Definition The minimal number of mutations (insertions, deletions and substitutions) needed to transform one sequence into the other.

Approximates the number of DNA replications that occurred between two DNA sequences

Example Given `a c c t g a` and `a g c t a`, the minimal number of mutations required to transform one into the other is 2:

```
a c c t g a
a g c t g a
a g c t a
```

```
a c c t g a
a g c t - a
```

The alignment above shows the difference between the sequences: substitution and deletion

Remark The definition of edit distance implies that all operations are done on one sequence only and the representation shown above might make false impression that the order of the changes is known

Sequence Alignment

Definition Given two sequences $S = S_1S_2\ldots S_m$ and $T = T_1T_2\ldots T_n$, an *alignment* is an assignment of gaps to positions in S or T, so as to line up each letter in one sequence with either a letter or gap in the other sequence

Scoring function: to evaluate the *goodness* of an alignment

Alignment algorithm: to find the best alignment representing the *minimal distance* or the *maximum similarity* between S and T

Biological models consider the significance of each mutation and score the alignment accordingly. The alignment score can be used to estimate the *biological difference* of two sequences

Example The aligned sequences

```
SEQ 1 GTAGTACAGCT-CAGTTGGGATCACAGGCTTCT

SEQ 2 GTAGAACGGCTTCAGTTG---TCACAGCGTTC-
```

Distance 1: match 0, substitution 1, indel 2 ⇒dis=14
Distance 2: match 0, d(A,T)=d(G,C)=1, d(A,G)=1.5, indel 2, ⇒dis=14.5

Similarity: match 1, substitution 0, indel -1.5 ⇒sim=16.5

Dot Matrix Pair-Wise Sequence Comparison

General way to see similarities between two sequences

Used for

- finding repeats in sequences
- finding palindromes in sequences
- RNA structure predictions
- ...

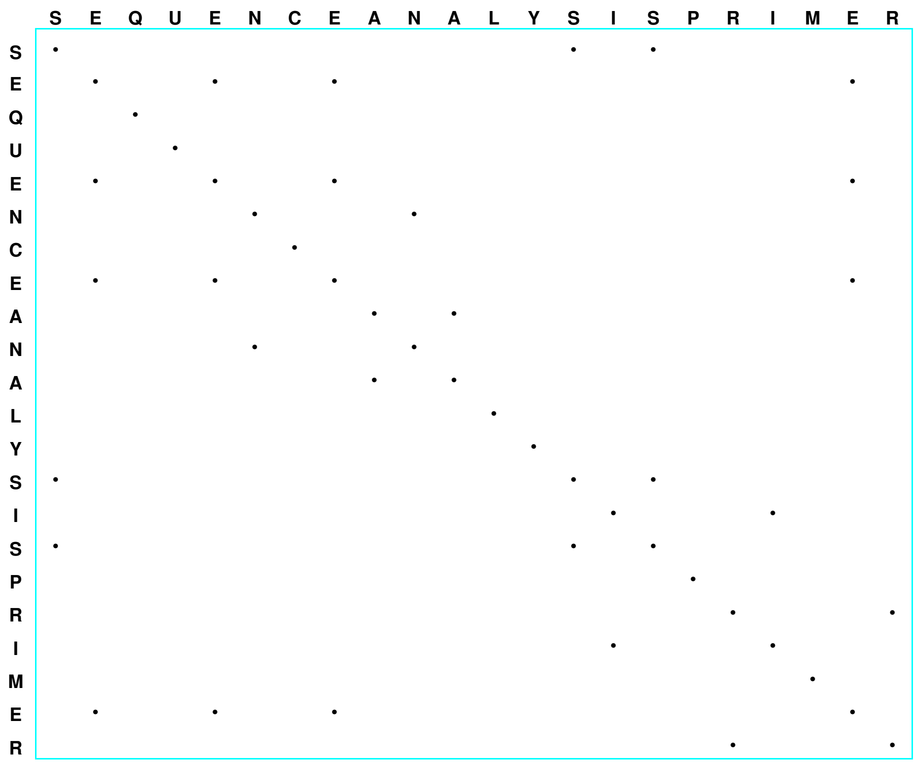
Visualizes the entire comparison at once

Dot Matrix Method

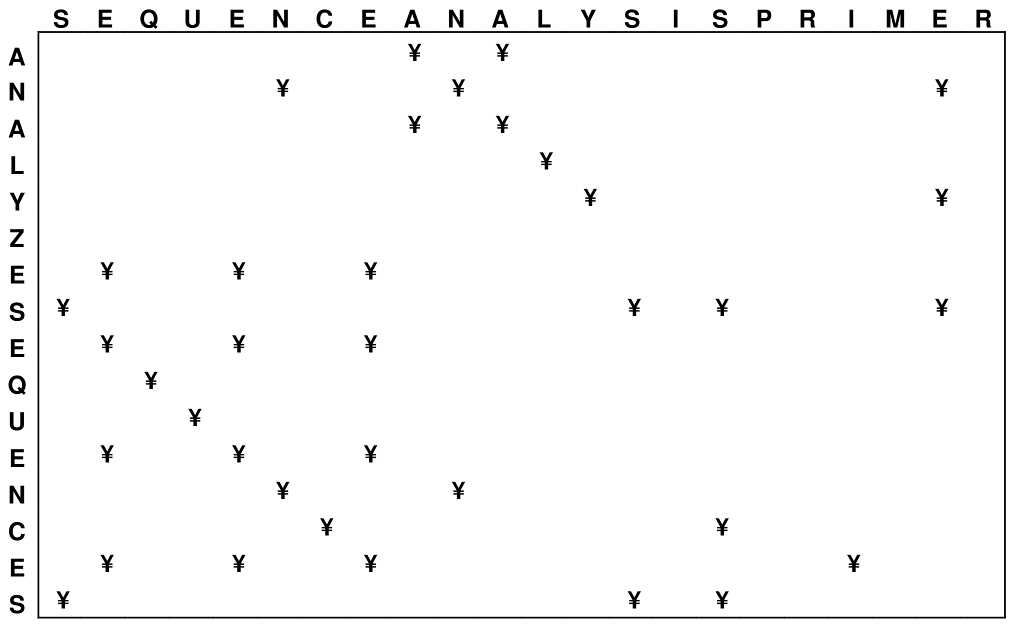
1. Sequences to be compared are written out along the x and y axis of a matrix
2. Define a scoring function, for example

   Identity scoring function: 1 (match) or 0 (no-match)
3. Put a *dot* wherever symbols match according to the scoring function
4. Diagonal lines indicates areas of similarities

Dot Matrix Pair-Wise Sequence Comparison



Since this is a comparison between two of the same sequences, an *intra-sequence comparison*, the most obvious feature is the main identity diagonal. Two short perfect palindromes can also be seen as crosses directly off the main diagonal; they are ANA and SIS



Here you can easily see the effect of a sequence insertion or deletion. It is impossible to tell whether the evolutionary event that caused the discrepancy between the two sequences was an insertion or a deletion and hence this phenomena is called an indel. A jump or shift in the register of the main diagonal on a dot plot clearly points out the existence of an indel

Scoring Function

To score an alignment we need to define the score of a given column of the alignment. There can be tree possibilities for a column

1. Match: Letters are the same
2. Mismatch: Letters are different
3. Gap: there is an indel

We need a score for each possibility:

- Similarity score function

  1. Match: $\sigma(a, a) = m$
  2. Mismatch: $\sigma(a, b) = s$, $a \neq b$
  3. Gap: $\sigma(a, -) = \sigma(-, a) = g$

  Similarity score of an alignment A of sequences S and T:

  $$\sigma(A) = \sum_{i=1}^{i=l} \sigma(S[i], T[i])$$

  that is

  $$\sigma(A) = (\#\text{matches}) \cdot m + (\#\text{mismatches}) \cdot s + (\#\text{gaps}) \cdot g$$

  Best alignments have highest scores

- Distance score function: defined such that best alignments have lowest scores

Finding the Best Alignment Exhaustively

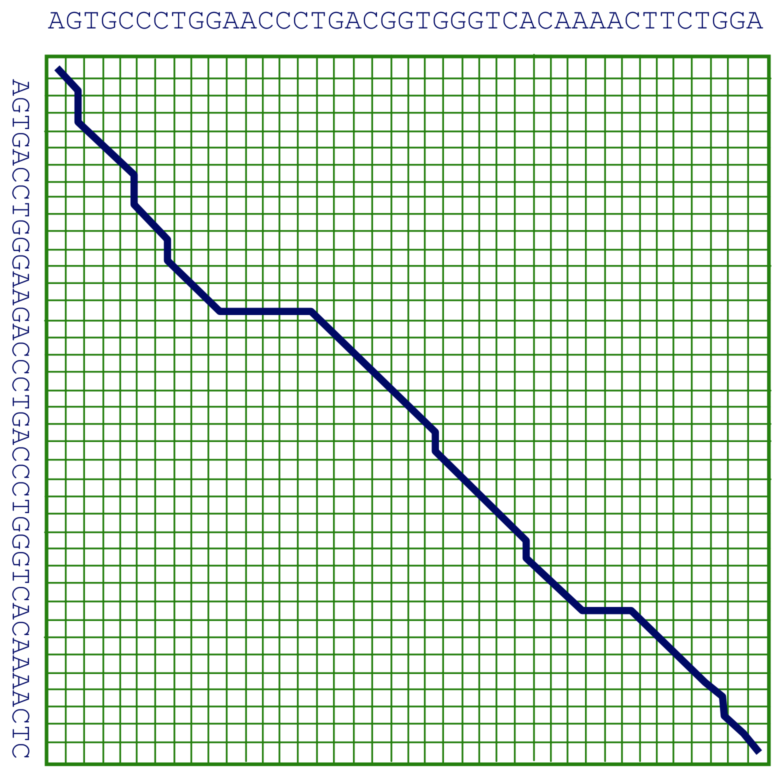
Generate all possible alignments

- Generate sequences with gaps in every position
- . . . and of every possible length
- . . . and pick one best alignment

This is too slow

Complexity is exponential: $O(2^{m+n})$

Better method is *Dynamic Programming*: $O(mn)$


AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

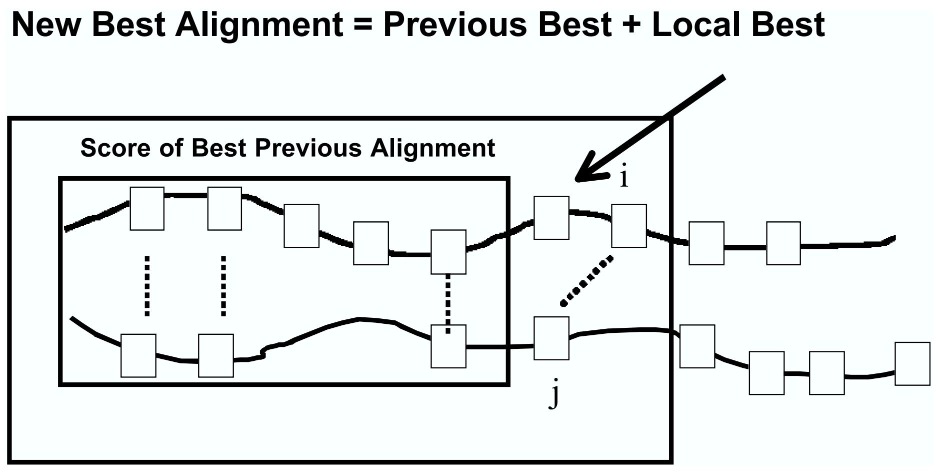
Too many possible alignments:

$O(2^{M+N})$

Dynamic Programming Method

Alignment is additive: score of alignment is sum of the scores of sub-alignments

- An optimal alignment is composed of optimal sub-alignments
- Score of the best alignment that ends at positions (i, j) in two sequences = score of the best alignment previous to those two positions + score of aligning those two positions


New Best Alignment = Previous Best + Local Best
Score of Best Previous Alignment
i
j

Dynamic Programming Solve a "large and hard" problem using solutions of its "easier and smaller subproblems". Given S and T, build up an optimal alignment by determining optimal alignments between *prefixes* of the two sequences. We start with the shorter prefixes and use previously computed results to solve the problem for larger prefixes

Prefix of S: $S_{1\ldots j}$ for $0 \leq j \leq |S|$
($S_{1\ldots 0}$ is the empty prefix)

Global Pairwise Alignment

Given two sequences S and T, find an optimal global alignment $[S : T]$. The term *global* emphasizes the entirety of both sequences

Given S and T, with $|S| = m$ and $|T| = n$ (i.e. $S = S_{1\ldots m}$ and $T = T_{1\ldots n}$).

Let σ be the scoring function

Let $V(i,j)$ be best score of $[S_{1\ldots i} : T_{1\ldots j}]$

Problem: compute $V(m,n)$ and associated $[S_{1\ldots m} : T_{1\ldots n}]$

Three possible cases given best $[S_{1\ldots i} : T_{1\ldots j}]$

1. $(S_i, -)$: S_i aligns to a gap.

$$V(i,j) = V(i-1,j) + \sigma(S_i, -)$$

2. (S_i, T_j): S_i aligns to T_j.

$$V(i,j) = V(i-1,j-1) + \sigma(S_i, T_j)$$

3. $(-, T_j)$: T_j aligns to a gap.

$$V(i,j) = V(i,j-1) + \sigma(-, T_j)$$

Global Alignment by Dynamic Programming

How do we know which case is correct?

An optimal alignment $[S_{1\ldots m} : T_{1\ldots n}]$ is composed of optimal sub-alignments

Inductive assumption:
Scores $V(i-1,j), V(i-1,j-1), V(i,j-1)$ are optimal

Then

$$V(i,j) = \max \begin{cases} V(i-1,j) & + \quad \sigma(S_i,-) \\ V(i-1,j-1) & + \quad \sigma(S_i,T_j) \\ V(i,j-1) & + \quad \sigma(-,T_j) \end{cases}$$

Base cases: Given $i, j > 0$ we have

$$\begin{aligned} V(0,0) &= 0 \\ V(i,0) &= V(i-1,0) + \sigma(S_i,-) = \textstyle\sum_{k=0}^{k=i} \sigma(S_k,-) \\ V(0,j) &= V(0,j-1) + \sigma(-,T_j) = \textstyle\sum_{k=0}^{k=j} \sigma(-,T_k) \end{aligned}$$

The basis for $V(i,0)$ says that if i characters of S are to be aligned with 0 characters of T, then they must all be aligned with gaps. The basis for $V(0,j)$ is analogous

The Needleman-Wunsch Algorithm

```
Algorithm:  Needleman-Wunsch
  Input:  $\sigma$, $S$ and $T$
  Output:  Maximum similarity between $S$ and $T$
  $V(0,0) = 0$

  $V(i,0) = \sum_{k=0}^{k=i} \sigma(S_k, -)$

  $V(0,j) = \sum_{k=0}^{k=j} \sigma(-, T_k)$

  For $i = 1$ to $m$ do
    For $j = 1$ to $n$ do
      $V[i,j] \leftarrow \max \begin{cases} V[i-1,j] & + & \sigma[S_i,-] & \texttt{case 1} \\ V[i-1,j-1] & + & \sigma[S_i,T_j] & \texttt{case 2} \\ V[i,j-1] & + & \sigma[-,T_j] & \texttt{case 3} \end{cases}$

      $Ptr[i,j] \leftarrow \max \begin{cases} L & \texttt{if case 1} \\ D & \texttt{if case 2} \\ U & \texttt{if case 3} \end{cases}$
  Return $V[m,n]$
```

Use two matrices of size $(m+1) \times (n+1)$

- Similarity matrix V to store the maximum score of each sub-alignments $[S_{1...i} : T_{1...j}]$
- Pointer matrix to *trace back* an optimal alignment. It stores the path to take: L = Left, D= Diagonal, U = Up

Time and space complexity = $O(mn)$

Recovering the Optimal Alignments

```
Recursive Algorithm:  Align
  Input:  indices $i$, $j$, $\sigma$ and matrix $V$ of Algorithm Similarity
  Output:  optimal alignment between $S$ and $T$ in align-S, align-T, length $l$
  If $i = 0$ and $j = 0$ then
    $l \leftarrow 0$
  Else
    if $i > 0$ and $V[i,j] = V[i-1,j] + \sigma(S_i, -)$ then
      Align$(i-1, j, l)$
      $l \leftarrow l + 1$
      align-S$[l] \leftarrow S[i]$
      align-T$[l] \leftarrow -$
    Else
      if $i > 0$ and $j > 0$ and $V[i,j] = V[i-1,j-1] + \sigma(S_i, T_j)$ then
        Align$(i-1, j-1, l)$
        $l \leftarrow l + 1$
        align-S$[l] \leftarrow S[i]$
        align-T$[l] \leftarrow -T[j]$
      Else // has to be $j > 0$ and $V[i,j] = V[i,j-1] + \sigma(-, T_j)$
        Align$(i, j-1, l)$
        $l \leftarrow l + 1$
        align-S$[l] \leftarrow -$
        align-T$[l] \leftarrow T[j]$
```

Method 1: Use the algorithm above if pointer matrix was not defined. Retrace `Needleman-Wunsch` algorithm from $V[m,n]$ back to $V[0,0]$, writing down the column associated with each $V(i,j)$.

Method 2: Create an optimal alignment in reverse order from $V(m,n)$. Use pointer matrix Ptr to obtain the column of the alignment.

Time complexity is $O(m+n)$

Space complexity is $O(mn)$

Example for Needleman-Wunsch Algorithm

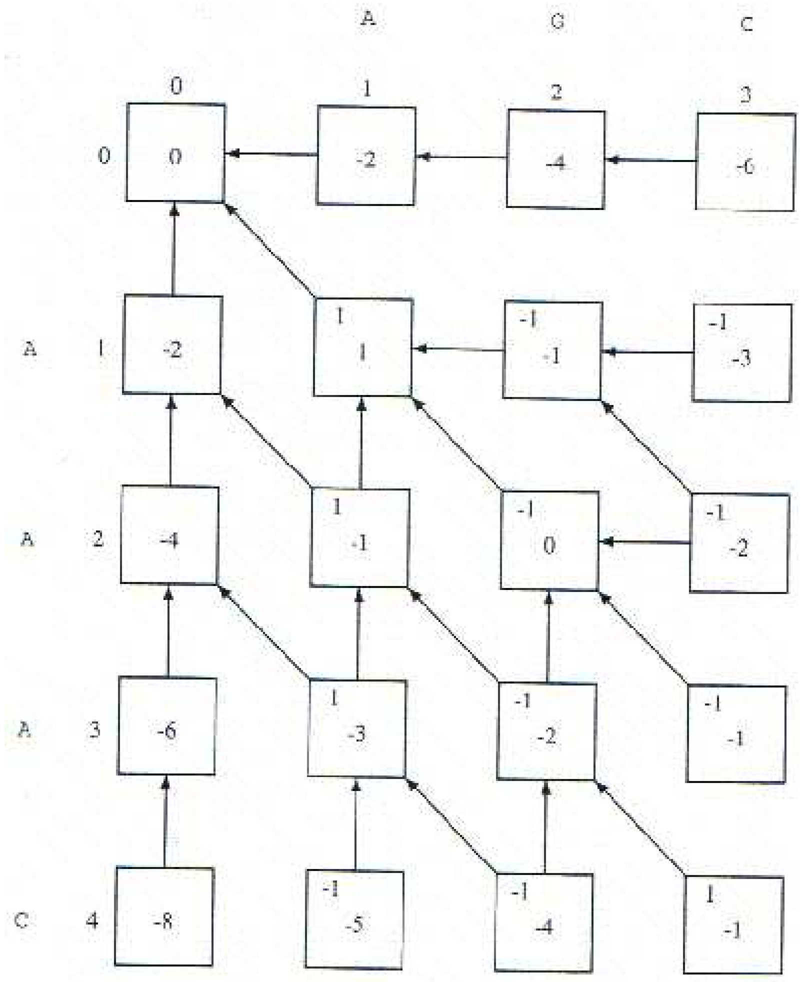
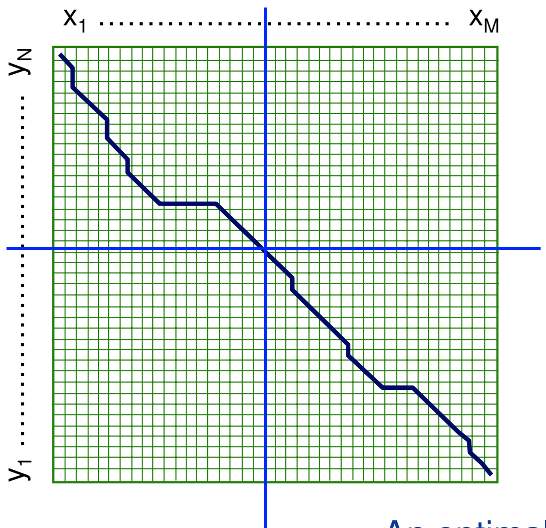


FIGURE 3.1

Bidimensional array for computing optimal alignments. The value in the upper left corner of cell (i, j) indicates whether $s[i] == t[j]$. Indices of rows and columns start at zero.



Every nondecreasing path

from (0,0) to (M, N)

corresponds to
an alignment
of the two sequences

An optimal alignment is composed of optimal subalignments

End-Space Free Alignment

Given two sequences S and T, find an optimal alignment $[S:T]$ between substrings of S and T when at least one of these substrings is a prefix of the original sequence and one (not necessarily the other) is a suffix

Why end-space free alignment?

- Searching for region of a *long* sequence that is similar to short sequence
- Searching for sequence that overlaps a given sequence

End spaces are those spaces that appear before first or after the last character in a sequence.

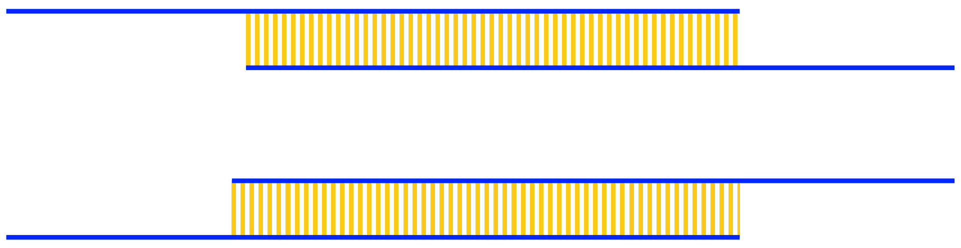
Example Consider the two alignments below

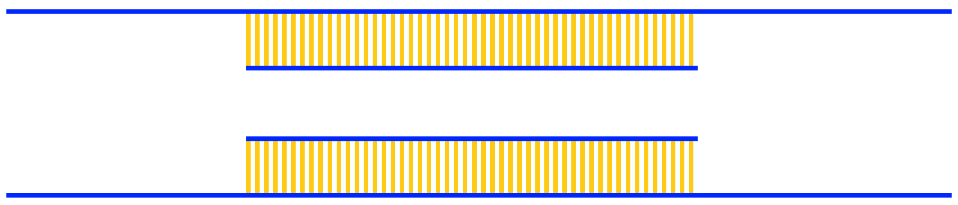
```
SEQ 1 CAGCA-CTTGGATTCTCGG
SEQ 2 ---CAGCGTGG--------

SEQ 1 CAGCACTTGGATTCTCGG
SEQ 2 CAGC-----G-T----GG
```

with (global) scores of -19 and -12, respectively. The second is an optimal global alignment, however it is not so interesting. If we ignore end spaces, the first is pretty good, with score of 3

Types of Alignments with End-Spaces

In many situations it is *ok* to have end-spaces at the start or end of an alignment

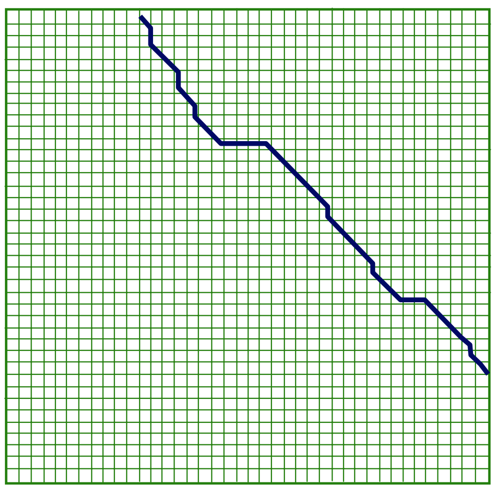




Then we should not penalize gaps in the ends

```
----------CTATCACCTGACCTCCAGGCCGATGCCCCTTCCGGC
GCGAGTTCATCTATCAC--GACCGC--GGTCG--------------
```

Alignments are scored ignoring some of the end-spaces



End-Space Free Alignment Algorithm

Variant of Needleman-Wunsch algorithm

Base cases: $V(0,0) = V(i,0) = V(0,j) = 0$

This allows zero weight to leading indels in (at most) one of the sequences

Recurrence relation for $i, j > 0$

$$V(i,j) = \max \begin{cases} V(i-1,j) & + \quad \sigma(S_i, -) \\ V(i-1,j-1) & + \quad \sigma(S_i, T_j) \\ V(i,j-1) & + \quad \sigma(-, T_j) \end{cases}$$

This is exactly the same recurrence relation as in global alignment algorithm

Optimal score is $V_{opt} = \max \begin{cases} \max_{1<i<m} V(i,n) \\ \max_{1<j<n} V(m,j) \end{cases}$

We search for the largest value in last column and last row. Thus allowing (at most) one sequence to end before the other, with zero weight for all indels from there on

We reconstruct the optimal end-space alignment by re-tracing from V_{opt} to first row or first column

Local Pairwise Alignment

Given two sequences S and T, find sub-strings S' and T' whose global similarity is maximum

Why local alignment? In many applications, two sequences may not be highly similar as a whole, but may contain sections with high resemblance.

- Searching for exons or genes
- Searching for protein domains
- Comparing two genomes or genes

Example Consider the two sequences

```
S = g g t c t g a g
T = a a a c g a
```

If match = 2 and indel/substitution = -1, then the best local alignment is

```
c t g a (∈ S)
c - g a (∈ T)
```

Exhaustive search: generate all pairs (S', T') of substrings and return one with maximum global similarity. Too slow. Better: use Smith-Waterman algorithm

Smith-Waterman Algorithm

Variant of Needleman-Wunsch algorithm

1. Scoring system uses negative scores for mismatches

Ignore badly aligning regions

2. Minimum score for $V(i,j)$ is 0

Should not penalize sub-alignments for their global effects

3. Best score is sought anywhere in matrix V

Alignments can occur anywhere

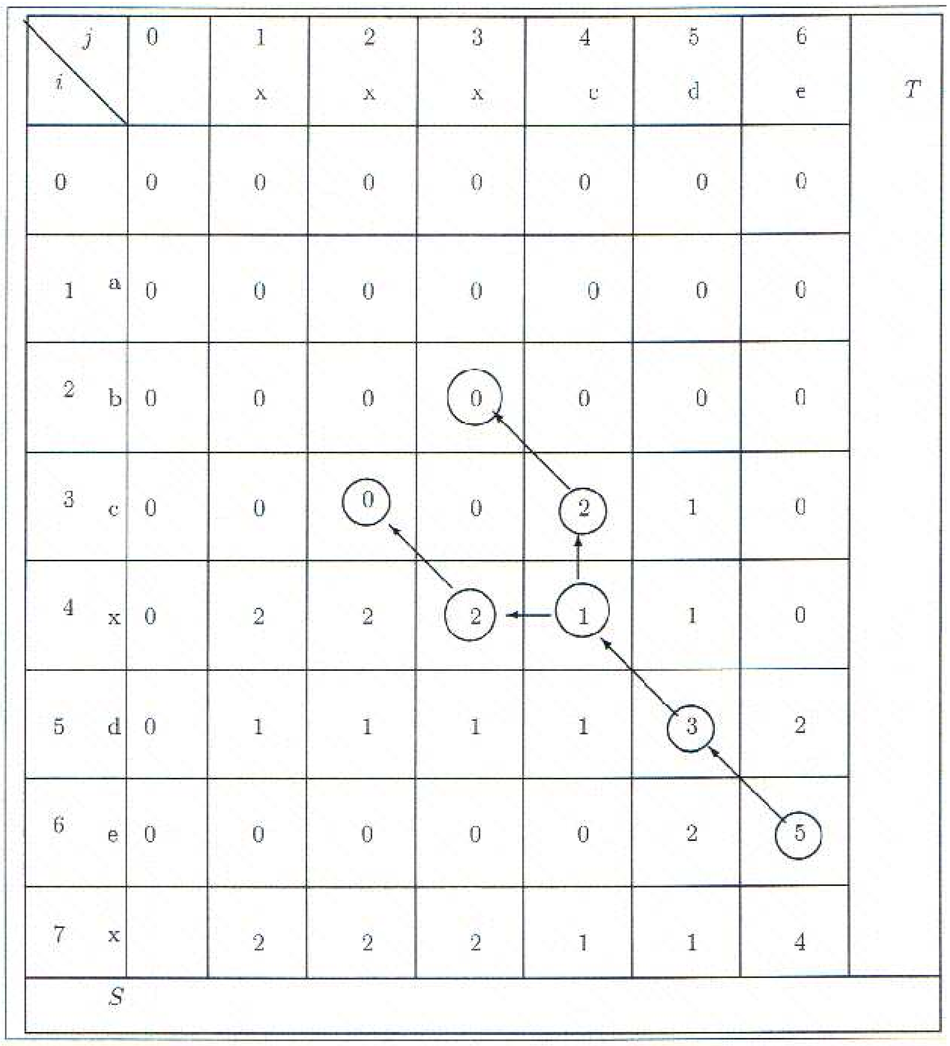
Base cases: $V(i,0) = V(0,j) = 0$

Recurrence relation for $i,j > 0$

$$V(i,j) = \max \begin{cases} 0 & \\ V(i-1,j) & + \;\; \sigma(S_i, -) \\ V(i-1,j-1) & + \;\; \sigma(S_i, T_j) \\ V(i,j-1) & + \;\; \sigma(-, T_j) \end{cases}$$

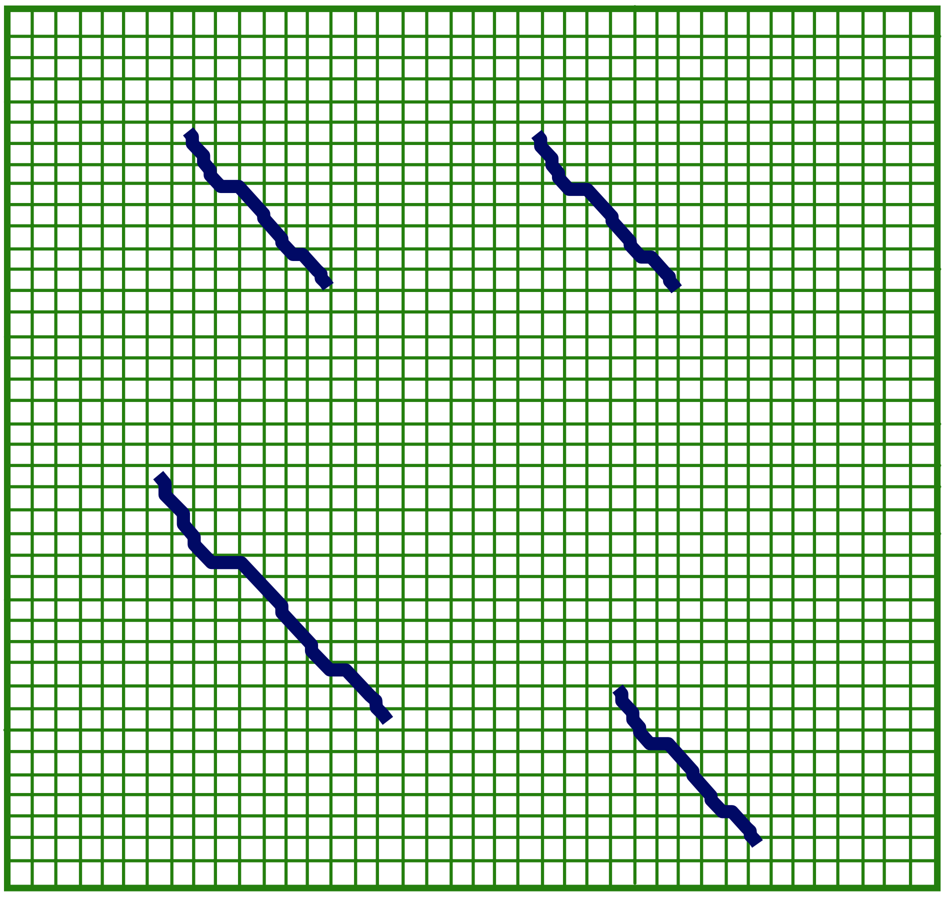
Optimal score is $V_{opt} = \max_{1 \le i \le m, 1 \le j \le n} V(i,j)$ and we trace-back to obtain the associated local alignment

Example for Smith-Waterman Algorithm



i \ j		0	1 x	2 x	3 x	4 c	5 d	6 e	T
0		0	0	0	0	0	0	0	
1	a	0	0	0	0	0	0	0	
2	b	0	0	0	0	0	0	0	
3	c	0	0	0	0	2	1	0	
4	x	0	2	2	2	1	1	0	
5	d	0	1	1	1	1	3	2	
6	e	0	0	0	0	0	2	5	
7	x		2	2	2	1	1	4	
	S								

Figure 2.5: Finding local alignment



Gap Penalties

A *gap* is any block of spaces in a single sequence of a given alignment. The *length* of the gap is the number of indels in it

```
SEQ 1 attc--ga-tggacc
SEQ 2 a--cgtgatt---cc
```

This alignment has four gaps and eight indels

Consider the identity scoring function and

```
SEQ 1 aatgc        aatg-c
SEQ 2 ag-gc        -a-ggc
```

Which is "better" or "biologically sound"?

Why better gap penalties?

1. Block insertions or deletions often occurs as a single mutational event and are as likely as single insertions or deletions
2. Two proteins sequences might be relatively similar over several intervals but differ in intervals where one contains a protein subunit that the other does not.
3. Complementary DNA (cDNA) matching to find exons

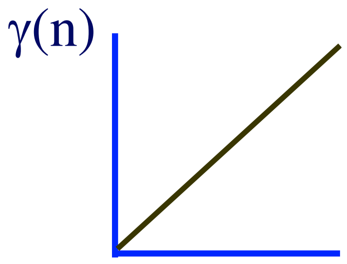
Need to score a gap . . .

- more accurately since the penalty introduced by the gap increases with the length of the gap
- as a whole when trying to align two sequences as to avoid assigning high cost to long gaps

Gap Penalty Functions

1. Linear gap penalty function:

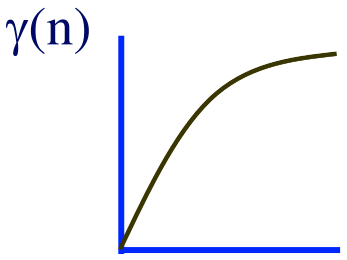
   $\gamma(n) = n \cdot g$, where $g = \sigma(a, -) = \sigma(-, a)$



Penalty increases too fast for large gaps

2. Convex gap penalty function:

   $\gamma(n+1) - \gamma(n) \leq \gamma(n) - \gamma(n-1)$



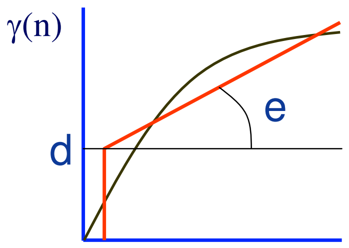
Penalty increases too slow for large gaps

3. Affine gap penalty function:

   $\gamma(n) = g + (n-1) \cdot e$

   $g = \sigma(a, -) = \sigma(-, a)$ is the cost of opening the gap

   e is the cost of extending the gap



Compromise between linear and convex gap function

Alignment Algorithm with Convex Gap Function

Given two sequences S and T and a convex gap penalty function, find an optimal alignment $[S : T]$

Base cases: depends on type of alignment (global or local or end-space)

Recurrence relation

$$V(i,j) = \max \begin{cases} [0 & \text{if} \quad \text{local alignment}] \\ \max_{0 \le k \le i-1} V(k,j) & + \quad \gamma(i-k) \\ V(i-1, j-1) & + \quad \sigma(S_i, T_j) \\ \max_{0 \le k \le j-1} V(i,k) & + \quad \gamma(j-k) \end{cases}$$

Optimal score and alignment: depends on type of alignment (global or local or end-space)

Time complexity is $O(n^2 m)$ (assume $n > m$)

Alignment Algorithm with Affine Gap Function

$V(i,j)$: Best score of alignment $[S_{1\ldots i} : T_{1\ldots j}]$

$F(i,j)$: Best score of $[S_{1\ldots i} : T_{1\ldots j}]$ if S_i aligns to T_j

$$F(i,j) = V(i-1,j-1) + \sigma(S_i, T_j)$$

$G(i,j)$: Best score of $[S_{1\ldots i} : T_{1\ldots j}]$ if S_i aligns to a gap after T_j

1. If S_{i-1} aligns to T_j

$$\begin{array}{lll} S_1 \ldots\ldots\ldots & S_{i-1} & S_i \\ T_1 \ldots\ldots\ldots & T_j & - \end{array}$$

$$G(i,j) = V(i-1,j) + g$$

2. If S_{i-1} aligns to a gap

$$\begin{array}{lll} S_1 \ldots\ldots\ldots & S_{i-1} & S_i \\ T_1 \ldots\ldots T_j & - & - \end{array}$$

$$G(i,j) = G(i-1,j) + e$$

$H(i,j)$: Best score of $[S_{1\ldots i} : T_{1\ldots j}]$ if T_j aligns to a gap after S_i

1. If T_{j-1} aligns to S_i

$$\begin{array}{lll} S_1 \ldots\ldots\ldots & S_i & - \\ T_1 \ldots\ldots\ldots & T_{j-1} & T_j \end{array}$$

$$H(i,j) = V(i,j-1) + g$$

2. If T_{j-1} aligns to a gap

$$\begin{array}{lll} S_1 \ldots\ldots S_i & - & - \\ T_1 \ldots\ldots\ldots & T_{j-1} & T_j \end{array}$$

$$H(i,j) = H(i,j-1) + e$$

Needleman-Wunsch with Affine Gap Function

Given two sequences S and T and an affine gap penalty function, find an optimal alignment $[S : T]$

Base cases:

$$\begin{aligned} V(0,0) &= 0 \\ V(i,0) &= g + (i-1)e \\ V(0,j) &= g + (j-1)e \end{aligned}$$

Recurrence relations

$$V(i,j) = \max \begin{cases} G(i,j) = \max \begin{cases} V(i-1,j) + g \\ G(i-1,j) + e \end{cases} \\ F(i,j) = V(i-1,j-1) + \sigma(S_i, T_j) \\ H(i,j) = \max \begin{cases} V(i,j-1) + g \\ H(i,j-1) + e \end{cases} \end{cases}$$

Return $V(m,n)$ and trace-back an optimal alignment from $V(m,n)$

Substitution Matrices

Due to physico-chemical properties, certain pairs of residues tend to substitute each other more frequently than other pairs

How often one residue is substituted for another in related sequence?

1. Manually align a set of related sequences at given sites
2. Obtain frequency $s(a,b)$ of each type of substitution (a,b)
3. Matrix entry is $\sigma(a,b) = \frac{\lg \frac{s(a,b)}{p(a)p(b)}}{\lambda}$

   $p(x) =$ probability of residue x

   λ is a range parameter

   In general $\sigma(a,b) = \lg \frac{\text{Observed frequency of (a, b)}}{\text{Expected frequency of (a, b)}}$

   $+$ score $\leftarrow$ More likely than random substitution

   0 score $\leftarrow$ Random substitution

   $-$ score $\leftarrow$ Less likely than random substitution

We can then use the substitution matrix σ and a gap penalty function γ to score alignments

Types of Substitution Matrices

1. Matrices based on observed rates of substitutions in sequences aligned using structural criteria

PAM or BLOSUM matrices

2. Matrices based on analysis of residues properties and similarities to predict substitutability

3. Matrices based on analysis of environments in which substitution occurs

WAC matrix

4. Position-specific scoring matrices and profiles

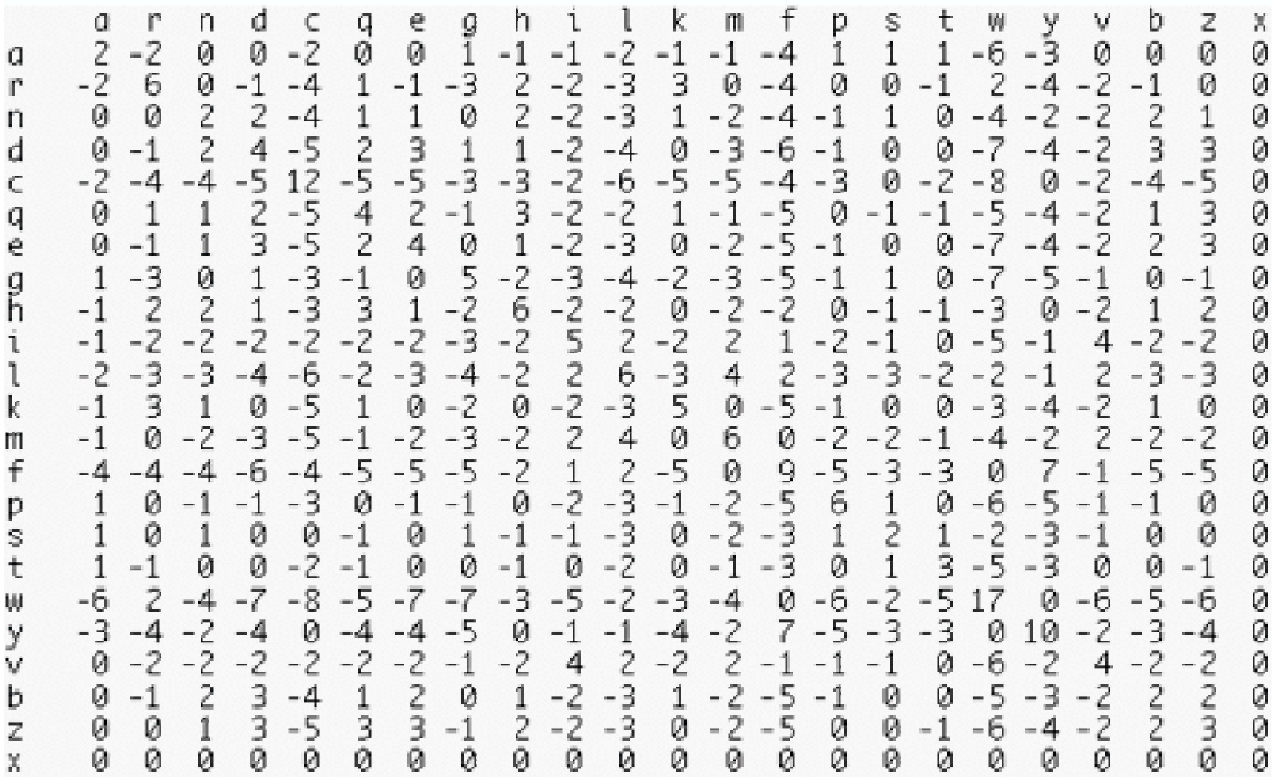
Changing matrix changes alignments

- Context-specific matching
- Finding homologous sequences
- Modeling evolution with different matrices

Alphabet need not be the same and matrix need not be symmetric

Examples of Substitution Matrices

A Sample Match Matrix for the amino acids (PAM-250).



	a	r	n	d	c	q	e	g	h	i	l	k	m	f	p	s	t	w	y	v	b	z	x
a	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-4	1	1	1	-6	-3	0	0	0	0
r	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	0	0
n	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-4	-1	1	0	-4	-2	-2	2	1	0
d	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	3	0
c	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	0
q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	3	0
e	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	2	3	0
g	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	-1	1	0	-7	-5	-1	0	-1	0
h	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	2	0
i	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	-2	0
l	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	-3	0
k	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	0	0
m	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	-2	0
f	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-5	-5	0
p	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	0	0
s	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	0	0
t	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	0
w	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-6	0
y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-4	0
v	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	-2	0
b	0	-1	2	3	-4	1	2	0	1	-2	-3	1	-2	-5	-1	0	0	-5	-3	-2	2	2	0
z	0	0	1	3	-5	3	3	-1	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

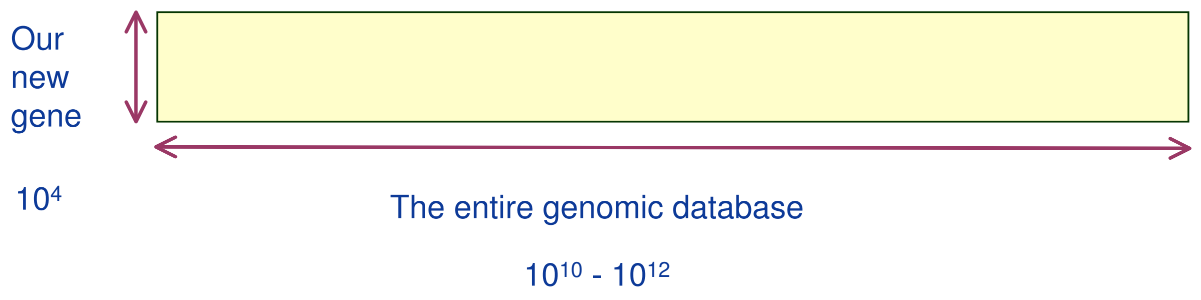
Can align amino acid sequences to environmental sequences...

	E1	E2	E3	E4	E5 ...
A	-0.77	–1.05	-0.54	-0.65	-1.52
R	-1.80	-1.52	-2.35	-0.11	-0.41
N	-1.76	-2.18	-2.61	-0.48	-0.26
D	-2.48	-1.80	-2.63	-0.80	-2.08
C	-0.43	-0.45	-0.59	0.15	-0.72
Q	-1.38	-2.03	-0.84	0.16	-0.79

Applications of Alignment Algorithms

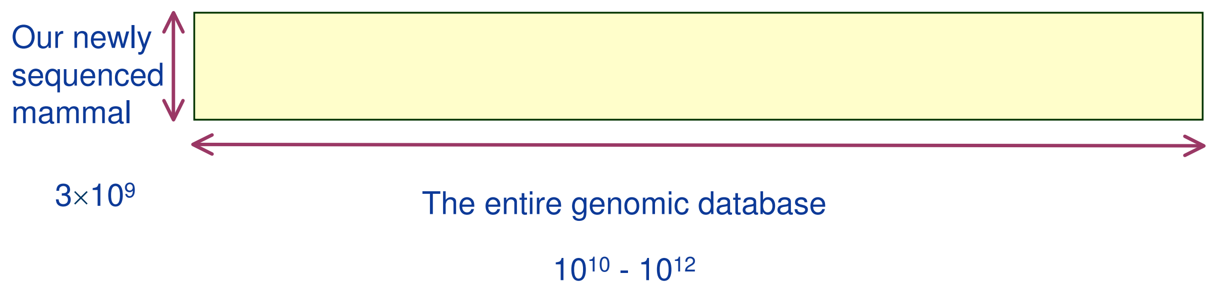
Given a newly discovered gene

- Does it occurs in other species?
- Other information about the gene?



Given a newly sequenced organism

- Which subregions align with other organism?
  1. Potential genes
  2. Other biological characteristics



Sequence Database Search

Needleman-Wunsch, Smith-Waterman and others alignment algorithms are too slow ($O(mn)$) for searching large databases of size 10^{10}–10^{12}

Even linear time is slow for database of size 10^{10}

Solutions:

1. Hardware implementation of dynamic programming
2. Parallel or distributed hardware/software
3. Better: Design efficient search heuristics
   - They give only *approximate* solutions
   - Solutions are not guaranteed to be optimal
   - Usually much faster than normal algorithms
   - They are based on

   (a) Preprocessing of the database

   (b) Finding un-gapped small matching sequences

   (c) Extending from smaller to larger matches

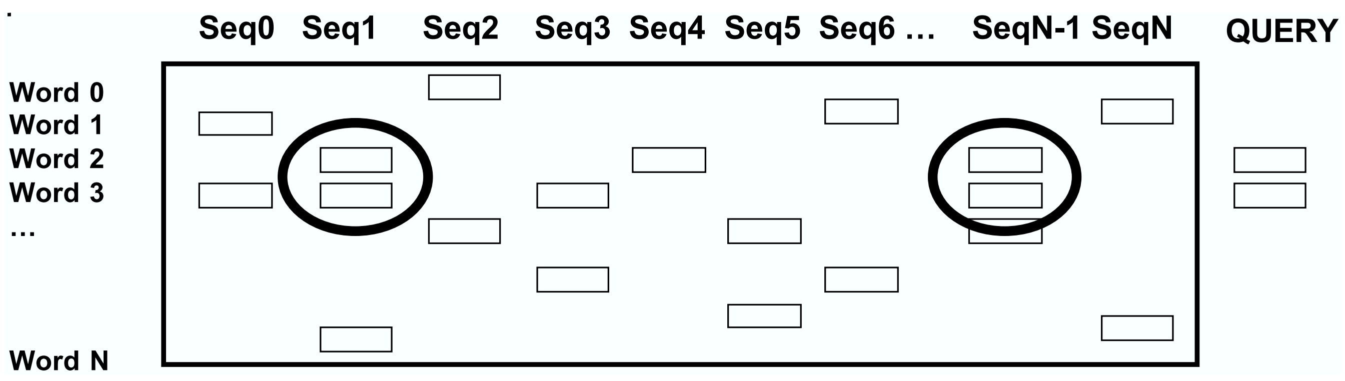
FASTA Heuristic

1. Make an hash table containing short words of the query sequence and of the sequences from a database

2. For each word of the query, find similar words in database by table look-up (hashing)

3. Score each match

4. Extend the good matches empirically



BLAST **(Basic Local Alignment Search Tool)** **Heuristic**

Definitions:

- *Segment pair (SP)*: pair of equal length segments of sequences S_1 and S_2 aligned without gaps

- *Locally maximal segment pair (LMS)*: a SP of S_1 and S_2 whose score cannot be improved by shortening or extending the SP

- *Maximal segment pair (MSP)*: a SP of S_1 and S_2 with the highest local alignment score

- *High scoring pair (HSP)*: a MSP of S_1 and S_2 whose score $> \sigma$

BLAST heuristic:

1. Break query sequence into w-words, $w = 3$

2. For each w-word W of query, find all SP's with score $\geq \tau$

3. Extend each high SP to a LMS

   - Extend one letter to right and left and ...

   - Repeat from **??** until we reach maximal length possible (LMS)

4. Return all HSP's, that is MSP's with score $> \sigma$

Similarity and Homology

Sequence similarity can be measured in many ways

Percentage of identical residues in alignment

Percentage of conservation in alignment

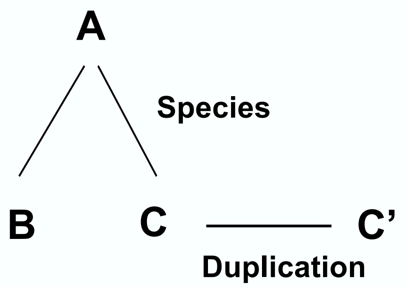
. . .

Homologous sequences refer to sequences that have common ancestor (related by evolution)

Types of homology

Orthologs are genes that have the same function in various species, and that have arisen by speciation.

Paralogs are other members of multigene families



A is the parent gene
Speciation leads to B and C
Duplication leads to C’

B and C are ORTHOLOGS
C and C’ are PARALOGS

Similarity may be evidence of homology but does not necessarily implies homology