B I O I N F O R M A T I C S

Kristel Van Steen, PhD^2

Montefiore Institute - Systems and Modeling

GIGA - Bioinformatics

ULg

kristel.vansteen@ulg.ac.be

CHAPTER 4: SEQUENCE COMPARISON

1 The biological problem

Paralogs and homologs

2 Pairwise alignment

3 Global alignment

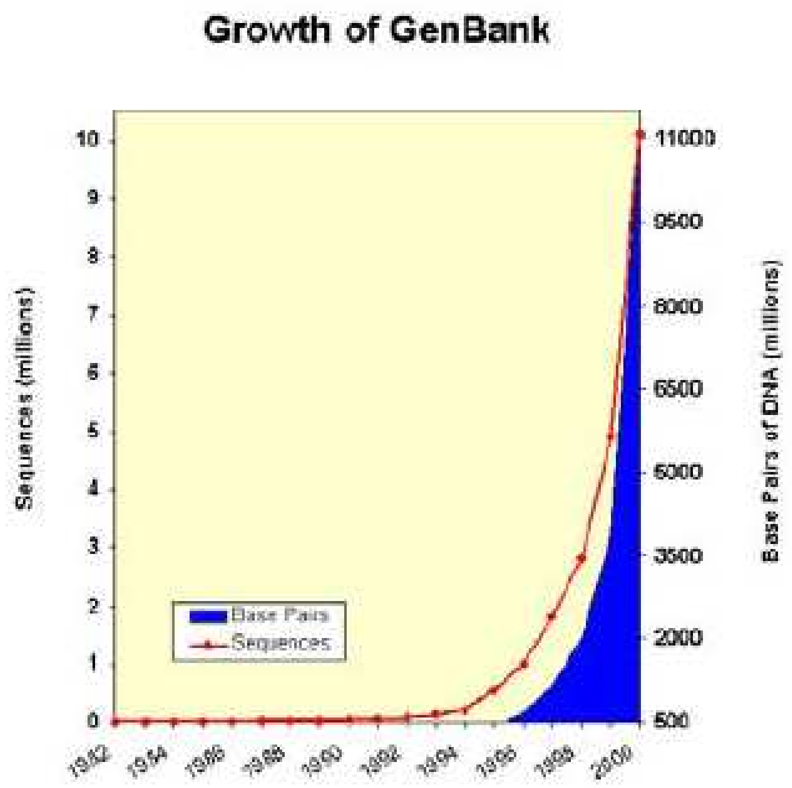
4 Local alignment

5 Rapid alignment methods

1 The biological problem

Introduction

- Ever-growing data sequence data bases make available a wealth of data to explore.
  - Recall that these data bases have a tendency of doubling approximately every 14 months and that
  - they comprise a total of over 11 billion bases from more than 100,000 species



Introduction

- Much of biology is based on recognition of shared characters among organisms
- The advent of protein and nucleic acid sequencing in molecular biology made possible comparison of organisms in terms of their DNA or the proteins that DNA encodes.

Introduction

- These comparisons are important for a number of reasons.
  - First, they can be used to establish evolutionary relationships among organisms using methods analogous to those employed for anatomical characters.
  - Second, comparison may allow identification of functionally conserved sequences (e.g., DNA sequences controlling gene expression).
  - Finally, such comparisons between humans and other species may identify corresponding genes in model organisms, which can be genetically manipulated to develop models for human diseases.

Evolutionary basis of alignment

- This lies in the fact that sequence alignment enables the researcher to determine if two sequences display sufficient similarity to justify the inference of homology.
- Understanding the difference between similarity and homology is of utmost importance:
  - Similarity is an observable quantity that may be expressed as a % identity or some other measure.
  - Homology is a conclusion drawn from the data that the two genes share a common evolutionary history.

(S-star Subbiah)

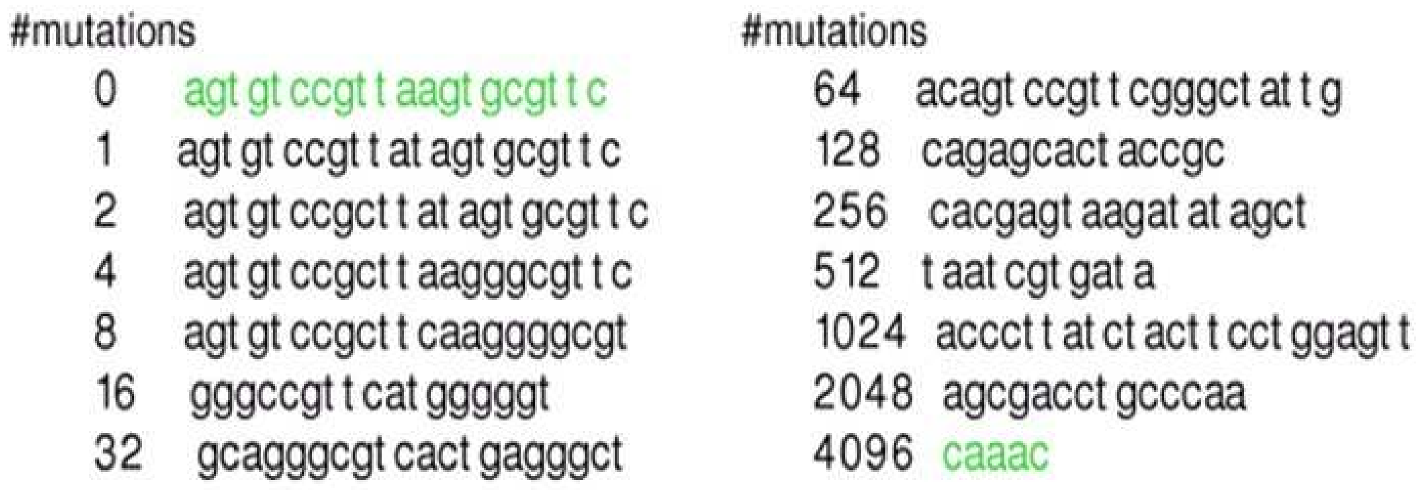
Evolutionary basis of alignment

- Genes are either homologous or not homologous:
  - they either share or
  - do not share a common evolutionary history.
- There are no degrees of homology as are there in similarity.
- While it is presumed that the homologous sequences have diverged from a common ancestral sequence through iterative molecular changes we do not actually know what the ancestral sequence was.
- In contrast to homology, there is another concept called homoplasy: Similar characters that result from independent processes (i.e., convergent evolution) are instances of *homoplasy*

(S-star Subbiah)

Homology versus similarity

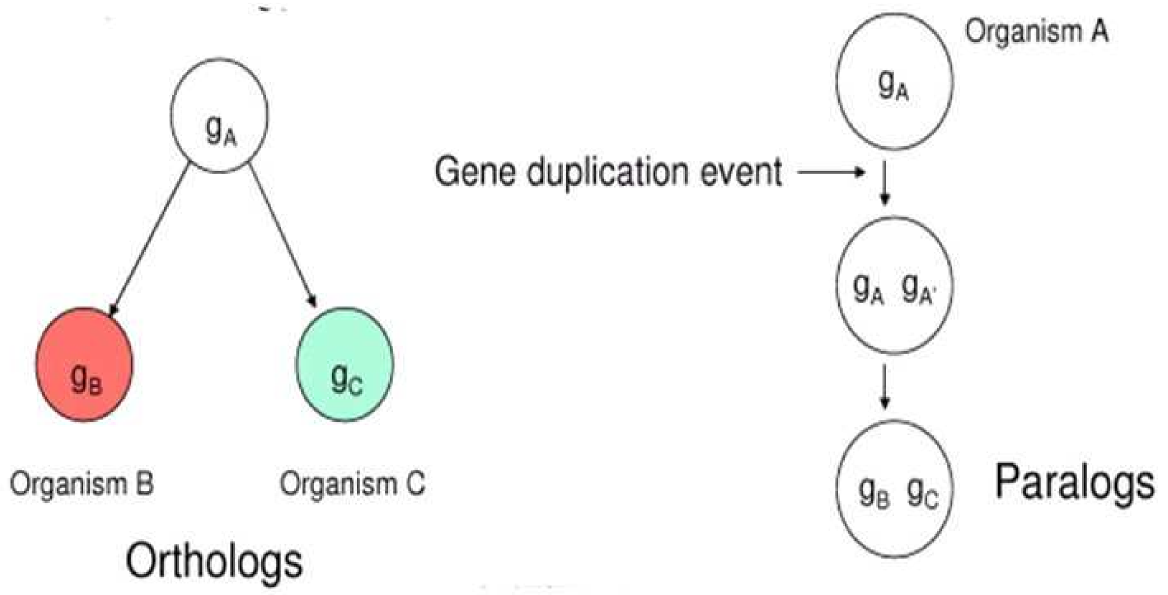
- Hence, sequence similarity is not sequence homology and can occur by chance ...
- If two sequences have accumulated enough mutations over time, then the similarity between them is likely to be low.
- Consequently, homology is more difficult to detect over greater evolutionary distances



#mutations		#mutations	
0	agt gt ccgt t aagt gcgt t c	64	acagt ccgt t cgggct at t g
1	agt gt ccgt t at agt gcgt t c	128	cagagcact accgc
2	agt gt ccgct t at agt gcgt t c	256	cacgagt aagat at agct
4	agt gt ccgct t aagggcgt t c	512	t aat cgt gat a
8	agt gt ccgct t caaggggcgt	1024	accct t at ct act t cct ggagt t
16	gggccgt t cat gggggt	2048	agcgacct gcccaa
32	gcagggcgt cact gagggct	4096	caaac

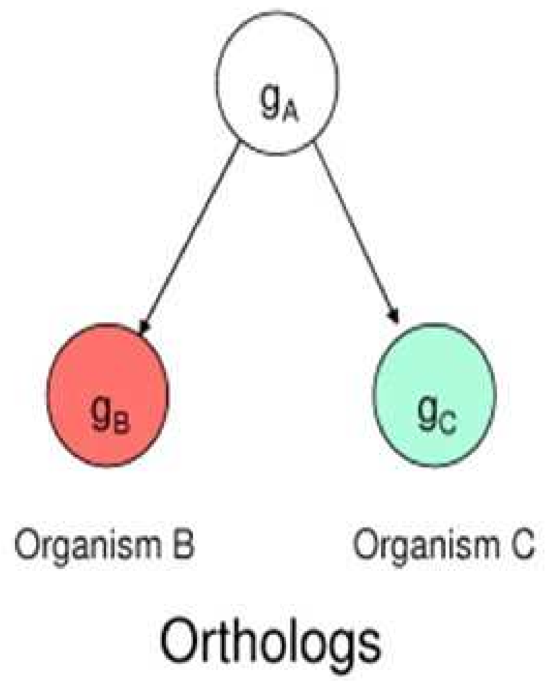
Orthologs and paralogs

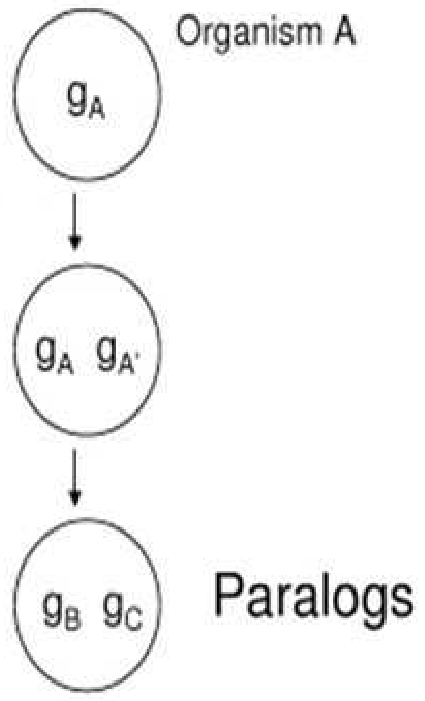
- We distinguish between two types of homology
  - Orthologs: homologs from two different species, separated by a, what is called, a *speciation event*
  - Paralogs: homologs within a species, separated by a *gene duplication event*.



Orthologs and paralogs

- Orthologs typically retain the original function
- In paralogs, one copy is free to mutate and acquire new function.

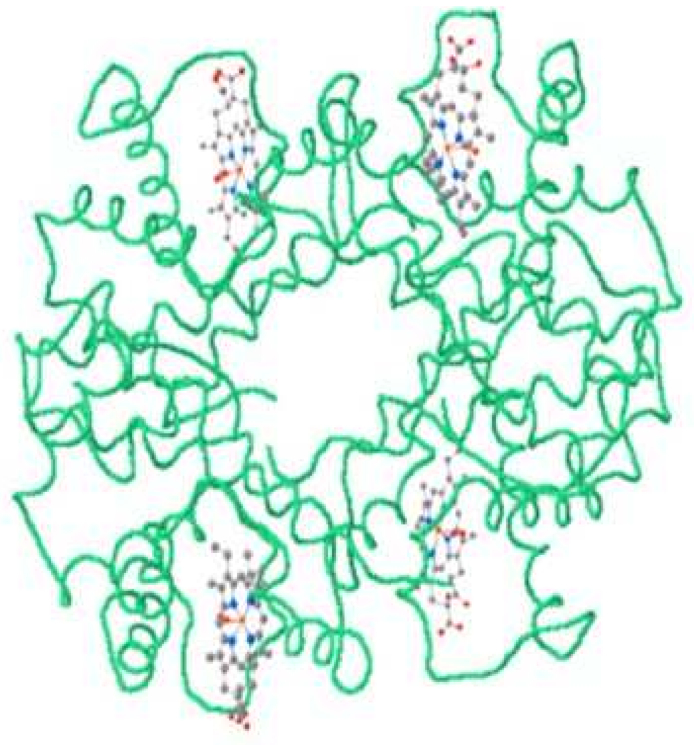




Example of paralogy: hemoglobin

- Hemoglobin is a protein complex that transports oxygen
- In adults, 3 types are normally present:
  - Hemoglobin A
  - Hemoglobin A2
  - Hemoglobin F

  each with a different combination of subunits.
- Each subunit is encoded by a separate gene.

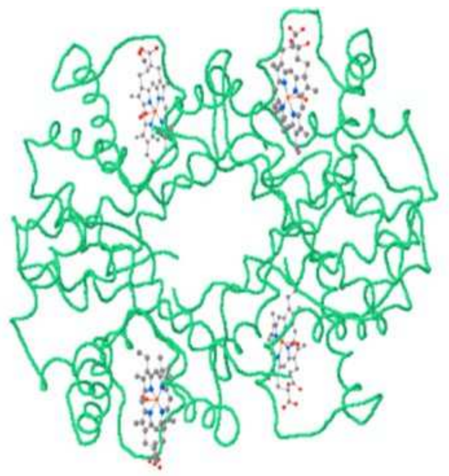


Hemoglobin A,
www.rcsb.org/pdb/explore.do?structureId=1GZX

Example of paralogy: hemoglobin

- The subunit genes are paralogs of each other. In other words, they have a common ancestor gene
- Check out the hemoglobin human paralogs using the NCBI sequence databases:

  http://www.ncbi.nlm.nih.gov/sites/entrez?db=nucleotide


Hemoglobin A,
www.rcsb.org/pdb/explore.do?structureId=1GZX

Sickle cell disease

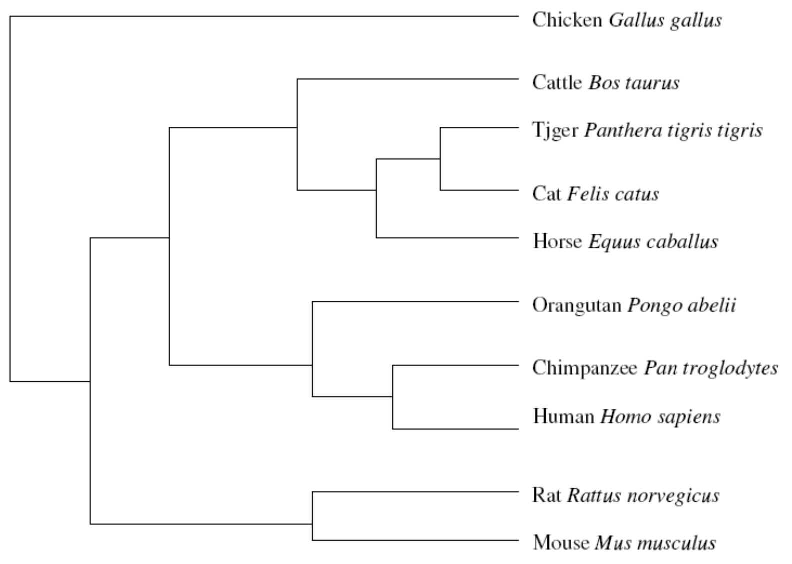
- Sickle-cell disease, or sickle-cell anaemia (or drepanocytosis), is a life-long blood disorder characterized by red blood cells that assume an abnormal, rigid, sickle shape.
- Sickling decreases the cells' flexibility and results in a risk of various complications.

(Wikipedia)

- The sickling occurs because of a mutation in the hemoglobin gene. Life expectancy is shortened, with studies reporting an average life expectancy of 42 and 48 years for males and females, respectively

Example of orthology: insulin

- The genes that code for insulin in humans (Homo Sapiens) and mouse (Mus musculus) are orthologs
- They have a common ancestor gene in the ancestor species of human and mouse



(example of a phylogenetic tree - Lakshmi)

Structural basis for alignment (as opposed to evolutionary basis)

- It is well-known that when two protein sequences have more than 20-30% identical residues aligned the corresponding 3-D structures are almost always structurally very similar.
- Therefore, sequence alignment is often an approximate predictor of the underlying 3-D structural alignment
- Form often follows function. So sequence similarity by way of structural similarity implies similar function.

(S-star Subbiah)

2 Pairwise alignment

Introduction

- An important activity in biology is identifying DNA or protein sequences that are similar to a sequence of experimental interest, with the goal of finding sequence homologs among a list of *similar* sequences.
- By writing the sequence of gene g_A and of each candidate homolog as strings of characters, with one string above the other, we can determine at which positions the strings do or do not match.
- This is called an alignment.

  Aligning polypeptide sequences with each other raises a number of additional issues compared to aligning nucleic acid sequences, because of particular constraints on protein structures and the genetic code (not covered in this class).

Introduction

- There are many different ways that two strings might be aligned. Ordinarily, we expect homologs to have more matches than two randomly chosen sequences.
- The seemingly simple alignment operation is not as simple as it sounds.
- Example (matches are indicated by . and - is placed opposite bases not aligned):

```
ACGTCTAG     2 matches
..           5 mismatches
ACTCTAG-     1 not aligned
```

Introduction

- We might instead have written the sequences

```
ACGTCTAG      5 matches
   .....      2 mismatches
-ACTCTAG      1 not aligned
```

- We might also have written

```
ACGTCTAG      7 matches
.. .....      0 mismatches
AC-TCTAG      1 not aligned
```

Which alignment is better?
What does better mean?

Introduction

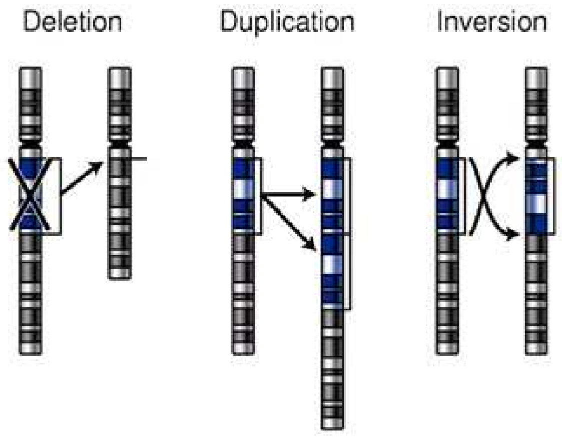
- Next, consider aligning the sequence TCTAG with a long DNA sequence:

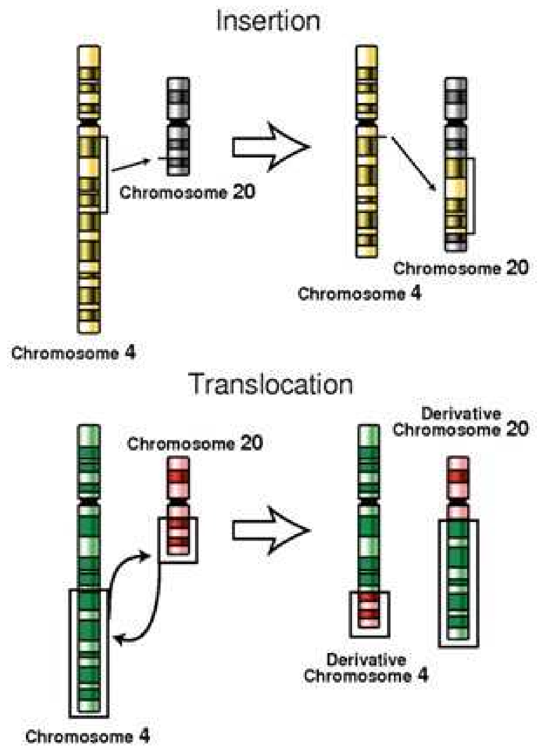
```
...AACTGAGTTTACGCTCATAGA...
            T---CT-A--G
```

- We might suspect that if we compared any string of modest length with another very long string, we could obtain perfect agreement if we were allowed the option of "not aligning" with a sufficient number of letters in the long string.
- Don’t we prefer some type of parsimonious alignment?
- What is parsimonious?

Introduction

- The approach adopted is guided by biology





NIH – National Human Genome Research Institute)

Introduction

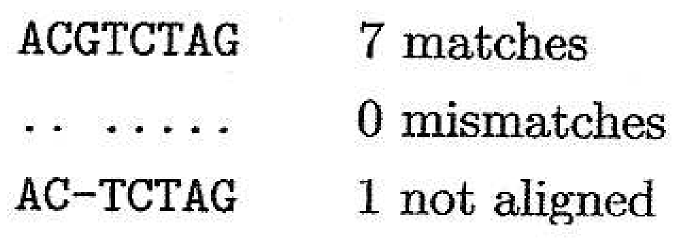
- Mutations such as segmental duplication, inversion, translocation often involve DNA segments larger than the coding regions of genes.
- Point mutations, insertion or deletion of short segments are important in aligning targets whose size are less than or equal to the size of coding regions of genes
- Therefore, these insertions and deletions need to be explicitly acknowledged in the alignment process.

Introduction

- There are multiple ways of aligning two sequence strings, and we may wish to compare our target string (target meaning the given sequence of interest) to entries in databases containing more than 10^7 sequence entries or to collections of sequences that are billions of letters long.
- How do we do this?
  - We differentiate between alignment of two sequences with each other
    - which can be done using the entire strings (global alignment)
    - or by looking for shorter regions of similarity contained within the strings (local alignment)
  - and multiple-sequence alignment (alignment of more than two strings)

Introduction

- Consider again the following alignment; we would like to acknowledge insertions and deletions when “aligning”



```
ACGTCTAG     7 matches
.. .....     0 mismatches
AC-TCTAG     1 not aligned
```

- We can't tell whether the string at the top resulted from the insertion of G in ancestral sequence ACTCTAG or whether the sequence at the bottom resulted from the deletion of G from ancestral sequence ACGTCTAG.
- For this reason, alignment of a letter opposite nothing is simply described as an indel (i.e., insertion/deletion).

Toy example, using the sequence as a whole

- Suppose that we wish to align WHAT with WHY.
- Our *goal* is to find the highest-scoring alignment. This means that we will have to devise a scoring system to characterize each possible alignment.
- One possible alignment solution is

```
WHAT
WH-Y
```

- However, we need a rule to tell us how to calculate an alignment score that will, in turn, allow us to identify which alignment is best.
- Let's use the following scores for each instance of match, mismatch, or indel:

- identity (match)	+1
- substitution (mismatch)	-μ
- indel	- δ

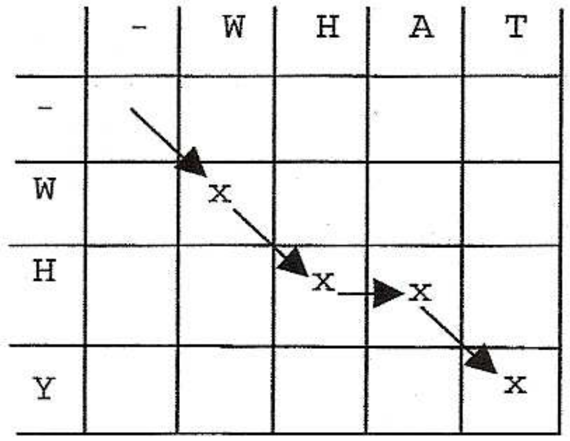
Toy example

- The minus signs for substitutions and indels assure that alignments with many substitutions or indels will have low scores.
- We can then define the score *S* as the sum of individual scores at each position:

$$S(\text{WHAT}/\text{WH}-\text{Y}) = 1 + 1 - \delta - \mu$$

- There is a more general way of describing the scoring process (not necessary for "toy" problems such as the one above).
- In particular: <u>write the target sequence (WHY) and the search space (WHAT) as rows and columns of a matrix:</u>

Toy example



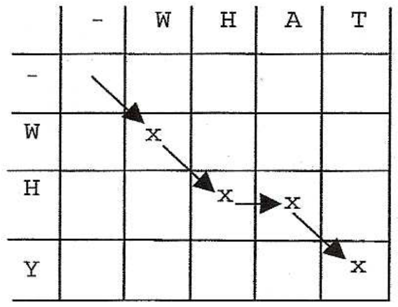
- We have placed an x in the matrix elements corresponding to a particular alignment
- We have included one additional row and one additional column for initial indels (-) to allow for the possibility (not applicable here) that alignments do not start at the initial letters (W opposite W in this case).

Toy example

- We can indicate the alignment

WHAT
WH-Y

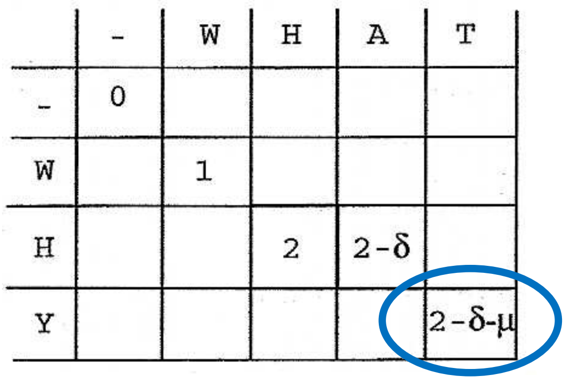
as a path through elements of the matrix (arrows).



- If the sequences being compared were identical, then this path would be along the diagonal.

Toy example

- Other alignments of WHAT with WHY would correspond to paths through the matrix other than the one shown.
- Each step from one matrix element to another corresponds to the incremental shift in position along one or both strings being aligned with each other, and we could write down in each matrix element <u>the running score</u> up to that point instead of inserting x (final score is indicated in blue).



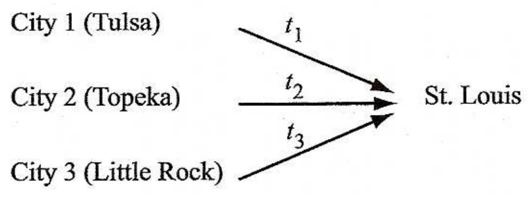
	-	W	H	A	T
-	0				
W		1			
H			2	$2-\delta$	
Y					$2-\delta-\mu$

Toy example

- What we seek is the path through the matrix that produces the greatest possible score in the element at the lower right-hand corner.
- That is our "destination," and it corresponds to having used up all of the letters in the search string (first column) and search space (first row)-this is the meaning of *global alignment.*
- Using a scoring matrix such as this employs a particular trick of thinking ...

Toy example

- For example, what is the "best" driving route from Los Angeles to St. Louis? We could plan our trip starting in Los Angeles and then proceed city to city considering different routes. For example, we



  might go through Phoenix, Albuquerque, Amarillo, etc., or we could take a more northerly route through Denver. We seek an itinerary (best route) that minimizes the driving time. One way of analyzing alternative routes is to consider the driving time to a city relatively close to St. Louis and add to it the driving time from that city to St. Louis.

Toy example

- Analyzing the best alignment using an alignment matrix proceeds similarly, first filling in the matrix by working forward and then working backward from the "destination" (last letters in a global alignment) to the starting point.
- This general approach to problem-solving is called *dynamic programming.*

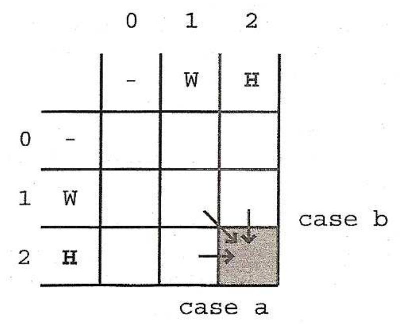
Dynamic programming

- Dynamic programming; a computer algorithmic technique invented in the 1940’s.
- Dynamic programming (DP) has applications to many types of problems.
- Key properties of problems solvable with DP include that the optimal solution typically contains optimal solutions to subproblems, and only a “small” number of subproblems are needed for the optimal solution.

(T.H. Cormen et al., Introduction to Algorithms, McGraw-Hill 1990).

Toy example (continued)

- We all agree that the best alignment is revealed by beginning at the destination (lower right-hand corner matrix element) and working backward, identifying the path that maximizes the score at the end.



- To do this, we will have to calculate scores for all possible paths. into each matrix element ("city") from its neighboring elements above, to the left, and diagonally above.
- We have now added row and column numbers to help us keep track of matrix elements.

Toy example

- There are three possible paths into element (2, 2) (aligning left to right with respect to both strings; letters not yet aligned are written in parentheses):

  Case a.

  If we had aligned WH in WHY with W in WHAT (corresponding to element (2, 1)), adding H in WHAT without aligning it to H in WHY corresponds to an insertion of H (relative to WHY) and advances the alignment from element (2, 1) to element (2,2) (horizontal arrow):

```
 (W)  H (AT)
(WH) -  (Y)
```

Toy example

Case b.

If we had aligned W in WHY with WH in WHAT (corresponding to element (1, 2)), adding the H in WHY without aligning it to H in WHAT corresponds to insertion of H (relative to WHAT) and advances the alignment from element (1, 2) to element (2, 2) (vertical arrow):

(WH)-(AT)
(W)H (Y)

Toy example

Case c.

If we had aligned W in WHY with W in WHAT (corresponding to element (1,1)), then we could advance to the next letter in both strings, advancing the alignment from (1,1) to (2,2) (diagonal arrow above):

(W)H(AT)
(W)H (y)

- Note that horizontal arrows correspond to adding indels to the string written vertically and that vertical arrows correspond to adding indels to the string written horizontally.

Toy example

- Associated with each matrix element *(x, y)* from which we could have come into (2,2) is the score $S_{x,y}$ up to that point.
- Suppose that we assigned scores based on the following scoring rules:

identity (match)	+1
substitution (mismatch)	-1
indel	-2

- Then the scores for the three different routes into (2,2) are

Case a : $S_{2,2} = S_{2,1} - 2,$

Case b : $S_{2,2} = S_{1,2} - 2,$

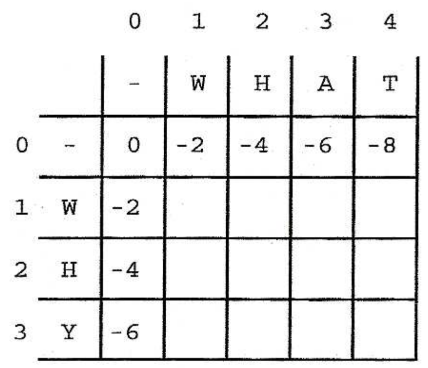
Case c : $S_{2,2} = S_{1,1} + 1.$

Toy example

- The path of the cases a, b, or c that yields the highest score for $S_{2,2}$ is the preferred one, telling us which of the alignment steps is best.
- Using this procedure, we will now go back to our original alignment matrix and fill in all of the scores for all of the elements, *keeping track of the path into each element that yielded the maximum score to that element.*
- The initial row and column labelled by (-) corresponds to sliding WHAT or WHY incrementally to the left of the other string without aligning against any letter of the other string.
- Aligning (-) opposite (-) contributes nothing to the alignment of the strings, so element (0,0) is assigned a score of zero.
- Since penalties for indels are -2, the successive elements to the right or down from element (0, 0) each are incremented by -2 compared with the previous one.

Toy example

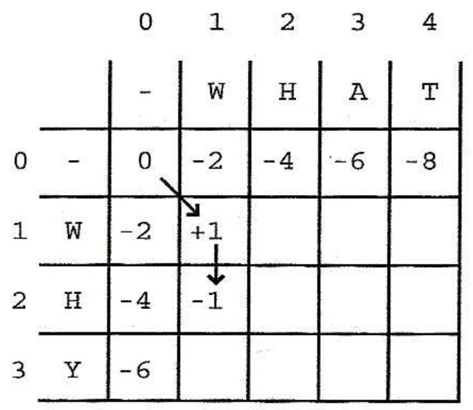
- Thus $S_{0,1}$ is -2, corresponding to W opposite -, $S_{0,2}$ is -4, corresponding to WH opposite **-,** etc., where the letters are coming from WHAT.
- Similarly, $S_{1,0}$ is -2, corresponding to - opposite *W,* $S_{2,0}$ is -4, corresponding to **-** opposite WH, etc., where the letters are coming from WHY.
- The result up to this point is



		0	1	2	3	4
		-	W	H	A	T
0	-	0	-2	-4	-6	-8
1	W	-2				
2	H	-4				
3	Y	-6				

Toy example

- Now we will calculate the score for (1,1). This is the greatest of $S_{0,0}$ + 1 (W matching W, starting from (0, 0)), $S_{0,1}$ - 2, or $S_{1,0}$ - 2.
- Clearly, $S_{0,0}$ + 1 = 0 + 1 = 1 "wins". (The other sums are -2 - 2 = -4.)
- We record the score value +1 in element (1, 1) and record an arrow that indicates where the score came from.



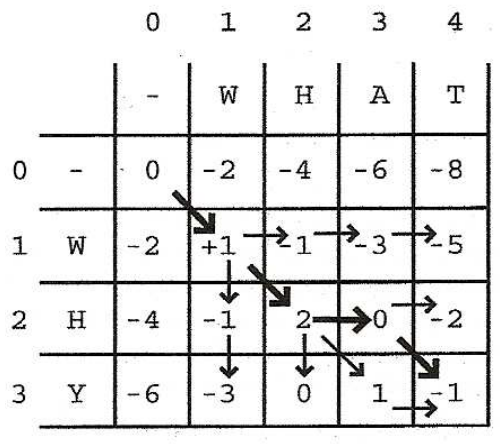
		0	1	2	3	4
		-	W	H	A	T
0	-	0	-2	-4	-6	-8
1	W	-2	+1			
2	H	-4	-1			
3	Y	-6				

Toy example

- The same procedure is used to calculate the score for element (2,1) (see above).
- Going from (1,0) to (2,1) implies that H from WHY is to be aligned with W of WHAT (after first having aligned W from WHY with (-)). This would correspond to a substitution, which contributes -1 to the score. So one possible value of $S_{2,1}$= $S_{1,0}$- 1 = -3.
- But (2, 1) could also be reached from (1, 1), which corresponds to aligning H in WHY opposite an indel in WHAT (i.e., not advancing a letter in WHAT). From that direction, $S_{2,1}=S_{1,1}$- 2 = 1 - 2 = -1.
- Finally, (2, 1) could be entered from (2,0), corresponding to aligning W in WHAT with an indel coming after H in WHY. In that direction, $S_{2,1}$. = $S_{2,0}$ - 2 = -4 - 2 = -6.
- We record the maximum score into this cell ($S_{2,1}$ = $S_{1,1}$ - 2 = -1) and the direction from which it came.

Toy example

- The remaining elements of the matrix are filled in by the same procedure, with the following result:



		0	1	2	3	4
		-	W	H	A	T
0	-	0	-2	-4	-6	-8
1	W	-2	+1	-1	-3	-5
2	H	-4	-1	2	0	-2
3	Y	-6	-3	0	1	-1

- The final score for the alignment is $S_{3,4}$ = -1.
- The score could have been achieved by either of two paths (implied by two arrows into (3, 4) yielding the same score).

Toy example

- The path through element (2,3) (upper path, bold arrows) corresponds to the alignment

```
WHAT
WH-Y
```

  which is read by tracing back through all of the elements visited in that path.
- The lower path (through element (3, 3)) corresponds to the alignment

```
WHAT
WHY-
```

- Each of these alignments is equally good (two matches, one mismatch, one indel).

Toy example

- Note that we always recorded the score for the best path into each element.
- There are paths through the matrix corresponding to very "bad" alignments. For example, the alignment corresponding to moving left to right along the first row and then down the last column is

```
WHAT---
----WHY
```

  with score -14.
- For this simple problem, the computations were not tough. But when the problems get bigger, there are so many different possible aligmnents that an organized approach is essential.
- Biologically interesting alignment problems are far beyond what we can handle with a No. 2.pencil and a sheet of paper, like we just did.

3 Global alignment

Formal development

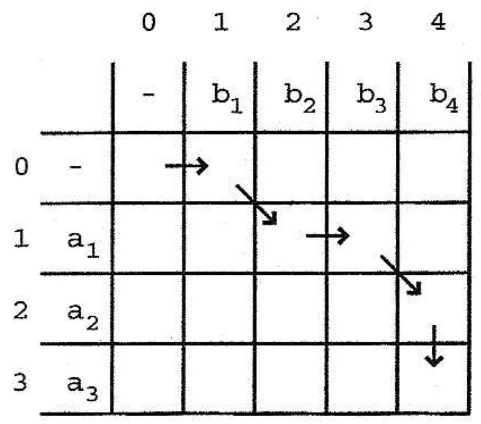
- We are given two strings, not necessarily of the same length, but from the same alphabet:

$$A = a_1 a_2 a_3 \cdots a_n,$$
$$B = b_1 b_2 b_3 \cdots b_m.$$

- Alignment of these strings corresponds to consecutively selecting each letter or inserting an indel in the first string and matching that particular letter or indel with a letter in the other string, or introducing an indel in the second string to place opposite a letter in the first string.

Formal development

- Graphically, the process is represented by using a matrix as shown below for *n* = 3 and m = 4:



- The alignment corresponding to the path indicated by the arrows is

$$\begin{array}{ccccc} b_1 & b_2 & b_3 & b_4 & - \\ - & a_1 & - & a_2 & a_3 \end{array}$$

Formal development

- Any alignment that can be written corresponds to a unique path through the matrix.
- The quality of an alignment between *A* and *B* is measured by a score, *S(A, B),* which is large when *A* and B have a high degree of similarity.
  - If letters a_i and b_j are aligned opposite each other and are the same, they are an instance of an **identity.**
  - If they are different, they are said to be a **mismatch.**
- The score for aligning the first i letters of *A* with the first j letters of *B* is

$$S_{i,j} = S\left(a_1 a_2 \cdots a_i, b_1 b_2 \cdots b_j\right).$$

Formal development

- *$S_{i,j}$* is computed recursively as follows. There are three different ways that the alignment of $a_1\, a_2 \dots a_i$ with $b_1\, b_2 \dots b_j$ can end:

$$\begin{array}{lll}
\text{Case a:} & (a_1\ a_2\ \cdots\ a_i) & - \\
 & (b_1\ b_2\ \cdots\ b_{j-1}) & b_j, \\
\text{Case b:} & (a_1\ a_2\ \cdots\ a_{i-1}) & a_i \\
 & (b_1\ b_2\ \cdots\ b_j) & -, \\
\text{Case c:} & (a_1\ a_2\ \cdots\ a_{i-1}) & a_i \\
 & (b_1\ b_2\ \cdots\ b_{j-1}) & b_j,
\end{array}$$

  where the inserted spaces "-" correspond to insertions or deletions ("indels") in *A* or *B.*
- Scores for each case are defined as follows:

$$\begin{aligned}
s(a_i, b_j) &= \text{score of aligning } a_i \text{ with } b_j \\
&\qquad (= +1 \text{ if } a_i = b_j, -\mu \le 0 \text{ if } a_i \ne b_j, \text{ for example}), \\
s(a_i, -) &= s(-, b_j) = -\delta \le 0 \text{ (for indels).}
\end{aligned}$$

Formal development

- With global alignment, indels will be added as needed to one or both sequences such that the resulting sequences (with indels) have the same length. The best alignment up to positions i and j corresponds to the case a, b, or c before that produces the largest score for *$S_{i,j}$:*

$$S_{i,j} = \max \left\{ \begin{array}{c} S_{i-1,j-1} + s(a_i, b_i) \\ S_{i-1,j} - \delta \\ S_{i,j-1} - \delta \end{array} \right\} \quad \begin{array}{l} \text{Case c} \\ \text{Case b.} \\ \text{Case a} \end{array}$$

- The ''max'' indicates that the· one of the three expressions that yields the maximum value will be employed to calculate $S_{i,j}$

Formal development

- Except for the first row and first column in the alignment matrix, the score at each matrix element is to ·be determined with the aid of the scores in the elements immediately above, immediately to the left, or diagonally above and to the left of that element.
- The scores for elements in the first row and column of the alignment matrix are given by

$$S_{i,0} = -i\delta, \quad S_{0,j} = -j\delta.$$

- The score for the best global alignment of *A* with *B* is *S*(*A, B*) = $S_{n,m}$ and it corresponds to the highest-scoring path through the matrix and ending at element (*n, m).* It is determined by tracing back element by element along the path that yielded the maximum score into each matrix element.

Exercise

- What is the maximum score and corresponding alignment for aligning

A=ATCGT with B=TGGTG?

- For scoring, take
  - $s(a_i, b_j) = +1$ if $a_i = b_j$,
  - $s(a_i, b_j) = -1$ if $a_i \neq b_j$ and
  - $s(a_i, -) = s(-, b_j) = -2$

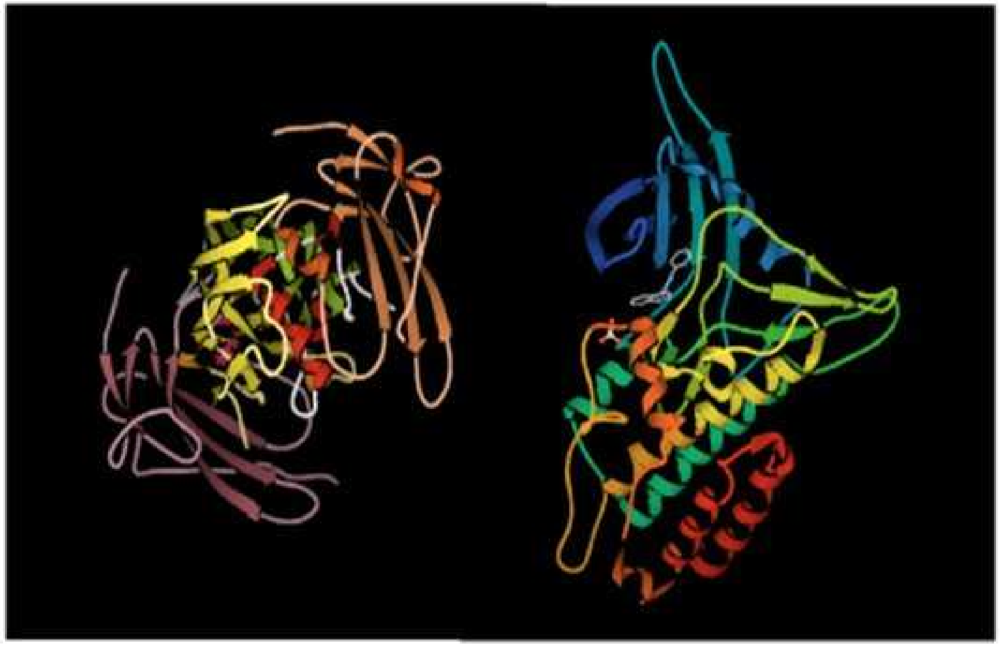
4 Local alignment

Rationale

- Proteins may be multifunctional. Pairs of proteins that share one of these functions may have regions of similarity embedded in otherwise dissimilar sequences.

Human bone morphogenic protein receptor type II precursor (left) has a 300 aa region that resembles 291 aa region in TGF-β receptor (right).

The shared function here is protein kinase.



Rationale

- With only partial sequence similarity and very different lengths, attempts at global alignment of two sequences such as these would lead to huge cumulative indel penalties.
- What we need is a method to produce the best *local alignment;* that is, an alignment of segments contained *within* two strings (Smith and Waterman, 1981).
- As before, we will need an alignment matrix, and we will seek a high-scoring path through the matrix.
- Unlike before, the path will traverse only *part* of the matrix. Also, we do not apply indel penalties if strings *A* and *B* fail to align “at the ends”.

Rationale

- Hence, instead of having elements -$i\delta$ and - $j\delta$ in the first row and first column, respectively (-δ being the penalty for each indel), all the elements in the first row and first column will now be zero.
- Moreover, since we are interested in paths that yield high scores over stretches less than or equal to the length of the smallest string, there is no need to continue paths whose scores become too small.
  - If the best path to an element from its immediate neighbors above and to the left (including the diagonal) leads to a negative score, we will arbitrarily assign a 0 score to that element.
  - We will identify the best local alignment by tracing back from the matrix element having the highest score. This is usually not (but occasionally may be) the element in the lower right-hand corner of the matrix.

Mathematical formulation

- We are given two strings $A = a_l a_2 a_3 \ldots a_n$ and $B = b_1 b_2 b_3 \ldots b_m$
- Within each string there are intervals I and J that have simillar sequences. I and J are *intervals* of A and B, respectively [$I \subset A$ and $J \subset B$, where "$\subset$" means "is an interval of"]
- The best local alignment score,M(A, B), for strings A and B is

$$M(A, B) = \max\{S(I, J) : I \subset A, J \subset B\}$$

  where $S(I, J)$ is the score for subsequences I and J and $S(\emptyset,\emptyset) = 0$.
- Elements of the alignment matrix are $M_{i,j}$, and since we are not applying indel penalties at the ends of A and B, we write

$$M_{i,0} = M_{0,i} = 0.$$

Mathematical formulation

- The score up to and including the matrix element $M_{i,j}$ is calculated by using scores for the elements immediately above and to the left (including the diagonal) ,but this time scores that fall below zero will be replaced by zero.
- The scoring for matches, mismatches, and indels is otherwise the same as for global alignment.
- The resulting expression for scoring $M_{i,j}$ is

$$M_{i,j} = \max \left\{ \begin{array}{l} M_{i-1,j-1} + s(a_i, b_i) \\ M_{i-1,j} - \delta \\ M_{i,j-1} - \delta \\ 0 \end{array} \right\}.$$

Mathematical formulation

- The best local alignment is the one that ends in the matrix element having the highest score:

$$\max\{S(I,J) : I \subset A, J \subset B\} = \max_{i,j} M_{i,j}.$$

- Thus, the best local alignment score for strings *A* and *B* is

$$M(A,B) = \max_{i,j} M_{i,j}.$$

Exercise

- Determine the best local alignment and the maximum alignment score for

A=ACCTAAGG and B=GGCTCAATCA

- For scoring, take
  - $s(a_i, b_j) = +2$ if $a_i = b_j$,
  - $s(a_i, b_j) = -1$ if $a_i \neq b_j$ and
  - $s(a_i, -) = s(-, b_j) = -2$

- The best local alignment is given below. The "aligned regions" of A and B are indicated by the blue box

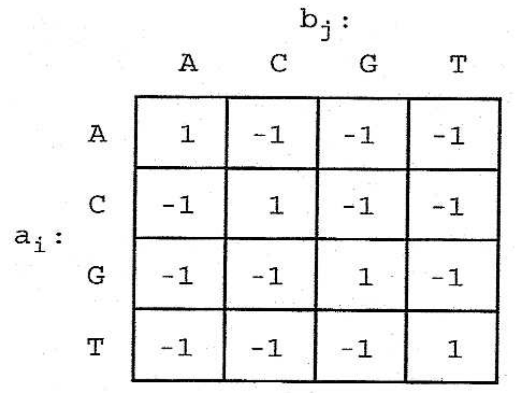
```
A: ACCT – AAGG -

B: GGCTC AATC A
```

Scoring rules

- We have used alignments of nucleic acids as illustrative examples, but it should be noted that for protein alignments the scoring is much more complicated.
- Hence, at this point, we address briefly the issue of assigning appropriate values to $s(a_i,b_j)$, $s(a_i,-)$, and $s(-,b_j)$ for nucleotides.
- For scoring issues in case of amino acids, you can find more details in Deonier et al. 2005 (Chapter 7, p182-189).

Scoring rules

- Considering *s(a_i, b_j)* first, we write down a **scoring matrix** containing *all* possible ways of matching a_i with b_j, a_i, $b_j \in$ {A, C, G, T} and write in each element the scores that we have used for matches +1 and



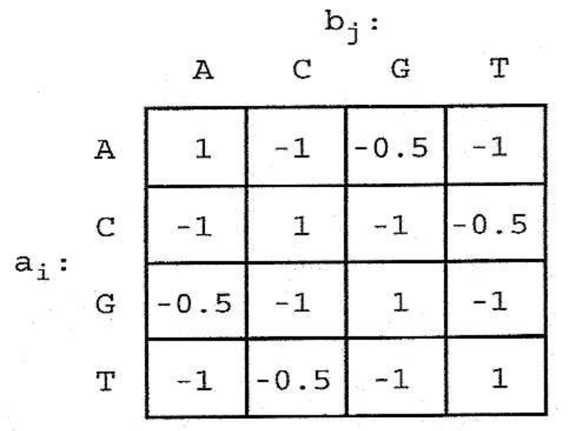
a_i : \ b_j :	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1

mismatches -1.

- Note that this scoring matrix contains the assumption that aligning A with G is just as bad as aligning A with T because the mismatch penalties are the same in both cases.

Scoring rules

- A first issue arises when observing that studies of mutations in homologous genes have indicated that transition mutations (A $\rightarrow$ G, G $\rightarrow$ A, C $\rightarrow$ T, or T $\rightarrow$ C) occur approximately twice as frequently as do transversions (A $\rightarrow$ T, T $\rightarrow$ A, A $\rightarrow$ C, G $\rightarrow$ T, etc.). (A, G are purines, larger molecules than pyrimidines T, C)
- Therefore, it may make sense to apply a lesser penalty for transitions than for transversions
- The collection of s(a_i , b_j) values in that case might be represented as



a_i : \ b_j :	A	C	G	T
A	1	-1	-0.5	-1
C	-1	1	-1	-0.5
G	-0.5	-1	1	-1
T	-1	-0.5	-1	1

Scoring rules

- A second issue relates to the scoring of gaps (a succession of indels), in that sense that we never bothered about whether or not indels are actually independent.
- Up to now, we have scored a gap of length *k* as

$$\omega(k) = -k\delta$$

- However, insertions and deletions sometimes appear in "chunks" as a result of biochemical processes such as replication slippage.
- Also, deletions of one or two nucleotides in protein-coding regions would produce frameshift mutations (usually non-functional – see before), but natural selection might allow small deletions that are integral multiples of 3, which would preserve the reading frame and some degree of function.

Scoring rules

- The aforementioned examples suggest that it would be better to have gap penalties that are not simply multiples of the number of indels.
- One approach is to use an expression such as

$$\omega(k) = -\alpha-\beta(k-1)$$

- This would allow us to impose a larger penalty for opening a gap *(-α)* and a smaller penalty for gap extension (-β for each additional base in the gap).

5 Rapid alignment methods

5.a Introduction

- The need for automatic (and rapid!) approaches also arises from the interest in comparing multiple sequences at once (and not just 2)

- Recall that $A = a_1a_2 \dots a_i$ and $B = b_1b_2 \dots b_j$ have alignments that can end in three possibilities: $(-,b_j)$, (a_i,b_j), or $(a_i, -)$.
- The number of alternatives for aligning a_i with b_j can be calculated as $3 = 2^2 - 1$
- If we now introduce a third sequence $c_1c_2 \dots c_k$, the three-sequence alignment can end in one of seven ways ($7 = 2^3 -1$): $(a_i,-,-)$,$(-,b_j,-)$, $(-,-,c_k)$,$(-,b_j,c_k)$,$(a_i,-,c_k)$,$(a_i,b_j,-)$, and (a_i, b_j,c_k) In other words, the alignment ends in 0, 1, or 2 indels.

5.b Search space reduction

- We can reduce the search space by analyzing word content (see Chapter 3). Suppose that we have the query string I indicated below:

```
I:   TGATGATGAAGACATCAG
```

- This can be broken down into its constituent set of overlapping k-tuples. For k = 8, this set is

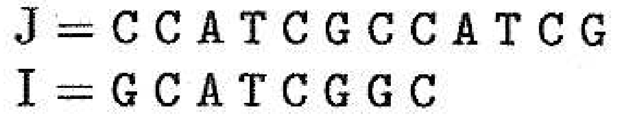
```
TGATGATG
 GATGATGA
  ATGATGAA
   TGATGAAG
        .
        .
        .
          GACATCAG
```

Search space reduction

- If a string is of length n, then there are n - k + 1 k-tuples that are produced from the string. If we are comparing string I to another string J (similarly broken down into words), the absence of anyone of these words is sufficient to indicate that the strings are not identical.
  - If I and J do not have at least some words in common, then we can decide that the strings are not similar.
- We know that when P(A) = P(C) = P(G) = P(T) = 0.25, the probability that an octamer beginning at any position in string J will correspond to a particular octamer in the list above is $1/4^8$.
  - Provided that J is short, this is not very probable.
  - If J is long, then it is quite likely that one of the eight-letter words in I can be found in J by chance.
- The appearance of a subset of these words is a *necessary but not sufficient* condition for declaring that I and J have meaningful sequence similarity.

Word Lists and Comparison by Content

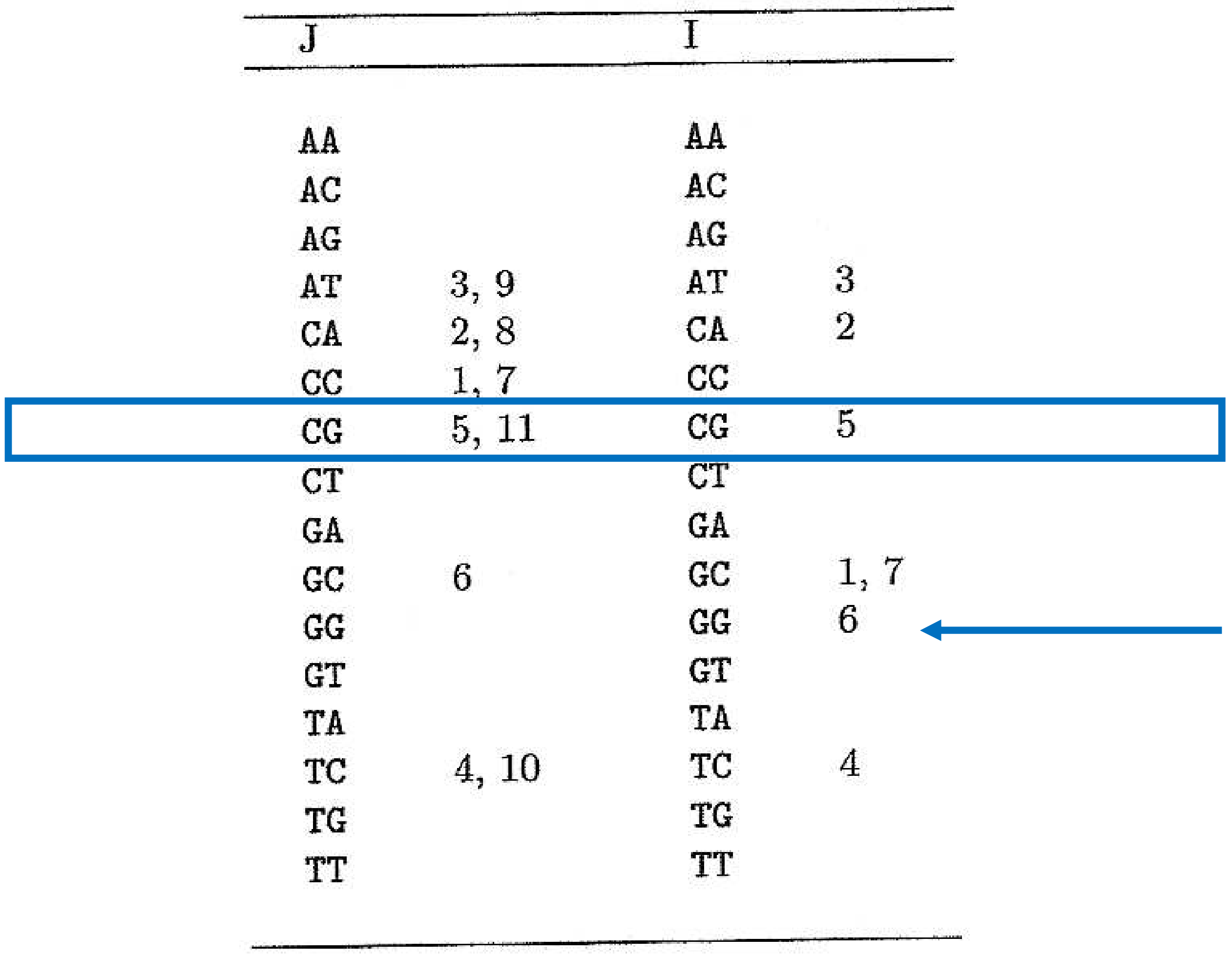
- Rather than scanning each sequence for each k-word, there is a way to collect the k-word information in a set of lists.
- A list will be a row of a table, where the table has 4^k rows, each of which corresponds to one k-word.
- For example, with k = 2 and the sequences below,



J = C C A T C G C C A T C G
I = G C A T C G G C

we obtain the word lists shown in the table on the next slide (Table 7.2, Deonier et al 2005).

Search space reduction



J		I	
AA		AA	
AC		AC	
AG		AG	
AT	3, 9	AT	3
CA	2, 8	CA	2
CC	1, 7	CC	
CG	5, 11	CG	5
CT		CT	
GA		GA	
GC	6	GC	1, 7
GG		GG	6
GT		GT	
TA		TA	
TC	4, 10	TC	4
TG		TG	
TT		TT	

Search space reduction

- Thinking of the rows as k-words, we denote the list of positions in the row corresponding to the word w as $L_w(J)$
  - e.g., with w = CG, $L_{CG}(J) = \{5,11\}$
- These tables are sparse, since the sequences are short
- Time is money: limit detailed comparisons only to those sequences that share enough "content"

Content sharing = = k-letter words in common

Search space reduction

- The statistic that counts k-words in common is

$$\sum_{i=1}^{n-k+1} \sum_{j=1}^{m-k+1} X_{i,j},$$

  where $X_{i,j} = 1$ if $I_i I_{i+1} \ldots I_{i+k-1} = J_j J_{j+1} \ldots J_{j+k-1}$ and 0 otherwise.
- The computation time is proportional to n x m, the product of the sequence lengths.
- To improve this, note that for each w in I, there are $\#L_w(J)$ occurrences in J. So the sum above is equal to:

$$\sum_w (\#L_w(\mathrm{I})) \times (\#L_w(\mathrm{J}))\,.$$

- Note that this equality is a restatement of the relationship between + and x

Search space reduction

- The second computation is much easier.
  - First we find the frequency of k-letter words in each sequence. This is accomplished by scanning each sequence (of lengths n and m).
  - Then the word frequencies are multiplied and added.
- For our sequence of numbers of 2-word matches, the statistic above is

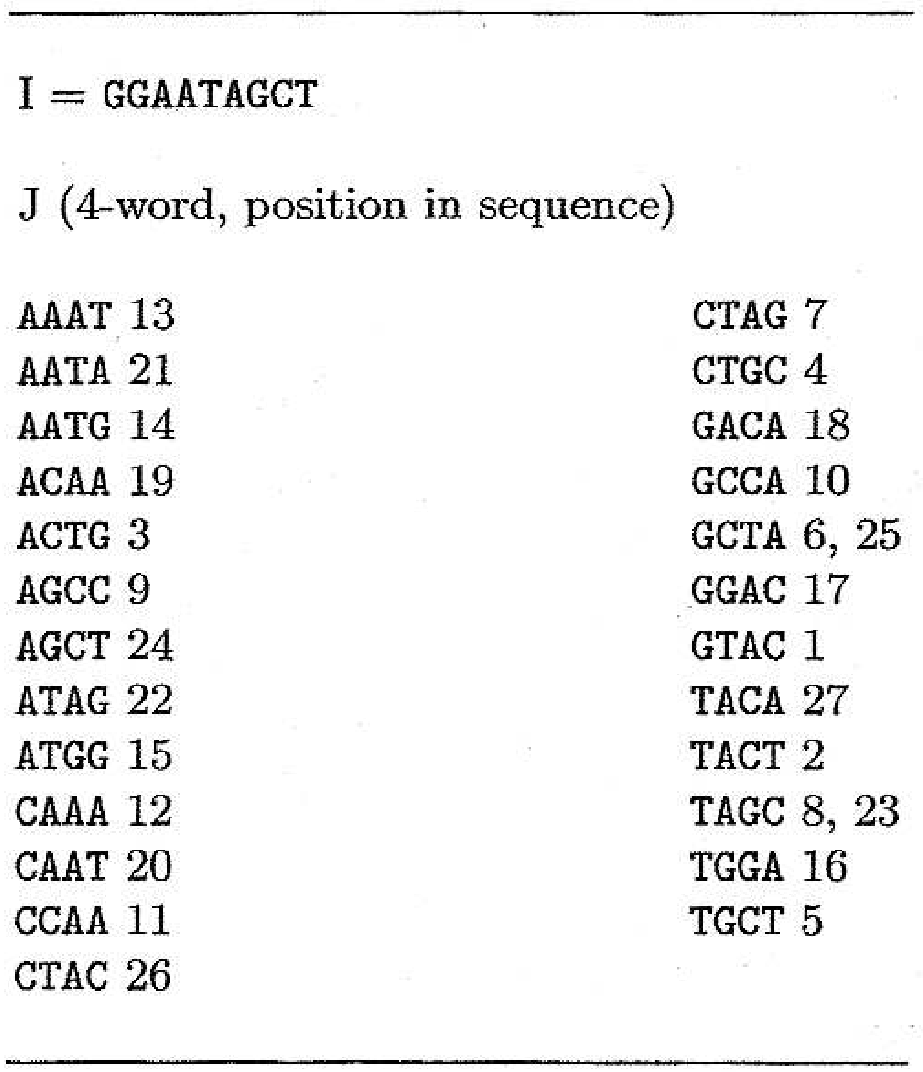
$$0^2+0^2+0^2+2\times 1+2\times 1+2\times 0+2\times 1+0^2+0^2+1\times 2+0\times 1$$
$$+0^2+0^2+2\times 1+0^2+0^2=10$$

- If 10 is above a threshold that we specify, then a full sequence comparison can be performed.
  - Low thresholds require more comparisons than high thresholds.

Search space reduction

- This aforementioned method is quite fast, but the comparison totally ignores the relative positions of the k-words in the sequence.
- A more sensitive method is needed....
- Also, what if you have a word list of over 5000 entries and you are looking at k's of more than 10? Is there a more efficient way of browsing through the list?

- Suppose I = GGAATAGCT, J = GTACTGCTAGCCAAATGGACAATAGCTACA, and we wish to find all k-word matches between the sequences with k = 4.
- Our method using k = 4 depends on putting the 4-words in J into a list ordered alphabetically as in the table below (Table 7.1.; Deonier et al 2005



I = GGAATAGCT

J (4-word, position in sequence)

AAAT	13	CTAG	7
AATA	21	CTGC	4
AATG	14	GACA	18
ACAA	19	GCCA	10
ACTG	3	GCTA	6, 25
AGCC	9	GGAC	17
AGCT	24	GTAC	1
ATAG	22	TACA	27
ATGG	15	TACT	2
CAAA	12	TAGC	8, 23
CAAT	20	TGGA	16
CCAA	11	TGCT	5
CTAC	26		

5.c Binary Searches

- Beginning with GGAA, we look in the J list for the 4-words contained in I by binary search.
- Since list J (of length m = 25) is stored in a computer, we can extract the entry number m/2, which in this example is entry 13, CTAC.
- In all cases we round fractions up to the next integer.
- Then we proceed as follows:

Binary Searches

- Step 1: Would GGAA be found before entry 13 in the alphabetically sorted list?
  - Since it would not, we don't need to look at the first half of the list.
- Step 2: In the second half of the list, would GGAA occur before the entry at position m/2 + m/4
  - i.e., before entry (18.75 hence) 19, GGAC
  - GGAA would occur before this entry, so that after only two comparisons -we have eliminated the need to search 75% of the list and narrowed the search to one quarter of the list.
- Step 3: Would GGAA occur after entry 13 but at or before entry 16?
  - We have split the third quarter of the list into two m/8 segments.
  - Since it would appear after entry 16 but at or before entry 19, we need only examine the three remaining entries.
- Steps 4 and 5: Two more similar steps are needed to conclude that GGAA is not contained in J.

Binary Searches

- Had we gone through the whole ordered list sequentially, 19 steps would have been required to determine that the word GGAA is absent from J.
- With the binary search, we used only five steps.
- We proceed in similar fashion with the next 4-word from I, GAAT. We also fail to find this word in J, but the word at position 3 of I, AATA, is found in the list of 4-words from J, corresponding to position 21 of J.
- Words in I are taken in succession until we reach the end.
- Remark:
  With this method, multiple matchings are found. This process is analogous to finding a word in a dictionary by successively splitting the remaining pages in half until we find the page containing our word.

Binary Searches

- In general, if we are searching a list of length m starting at the top and going item by item, on average we will need to search half the list before we find the matching word (if it is present).
- If we perform a binary search as above, we will need only $\log_2(m)$ steps in our search.
- This is because $m = 2^{\log_2(m)}$, and we can think of all positions in our list of length m as having been generated by $\log_2(m)$ doublings of an initial position.
- In the example above, m=30. Since $32 = 2^5$, we should find any entry after five binary steps. Note $\log_2(30)=4.9$

Toy example

- Suppose a dictionary has 900 pages
- Then finding the page containing the 9-letter word "crescendo" in this dictionary, using the binary search strategy, should require how many steps?
  - The dictionary is 864 pages long say
  - $2^9 = 512$
  - $2^{10} = 1024$
  - $512 < 864 < 1024$; $\log_2(864) = 9.75 \sim 10$
- You need 10 binary steps to find the appropriate page...
- Within the page, you can again adopt a binary search to find the correct word among the list of words on that page ...

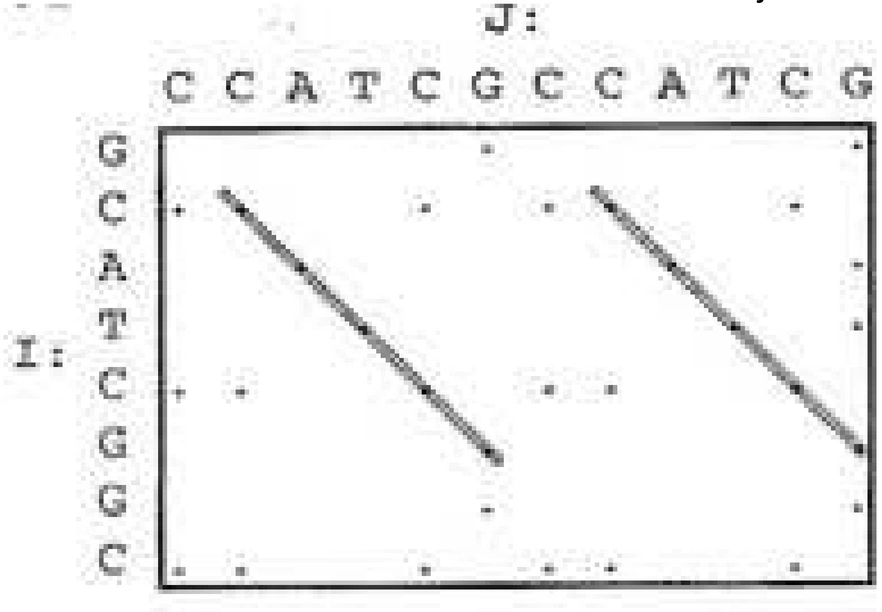
- Although binary searches are very efficient, we could use some automated steps to reduce the search space.

5.d Looking for regions of similarity using FASTA

- FASTA (Pearson and Lipman, 1988) is a rapid alignment approach that combines methods to reduce the search space
- FASTA (pronounced FAST-AYE) stands for **FAST**-**A**LL, reflecting the fact that it can be used for a fast protein comparison or a fast nucleotide comparison.
- It depends on k-tuples and (Smith-Waterman) local sequence alignment.
- As an introduction to the rationale of the FASTA method, we begin by describing dot matrix plots, which are a very basic and simple way of visualizing regions of sequence similarity between two different strings. It allows us to identify k-tuple correspondences (*first crucial step in FASTA*)

Dot Matrix Comparisons

- Dot matrix comparisons are a special type of alignment matrix with positions *i* in sequence I corresponding to rows, positions *j* in sequence *J* corresponding to columns
- Moreover, sequence identities are indicated by placing a dot at matrix element (i,j) if the word or letter at I_i is identical to the word or letter at J_j.
- An example for two DNA strings is shown in the right panel. The string CATCG in I appears twice in J, and these regions of local sequence similarity appear as two diagonal arrangements of dots: diagonals represent regions having sequence similarity.



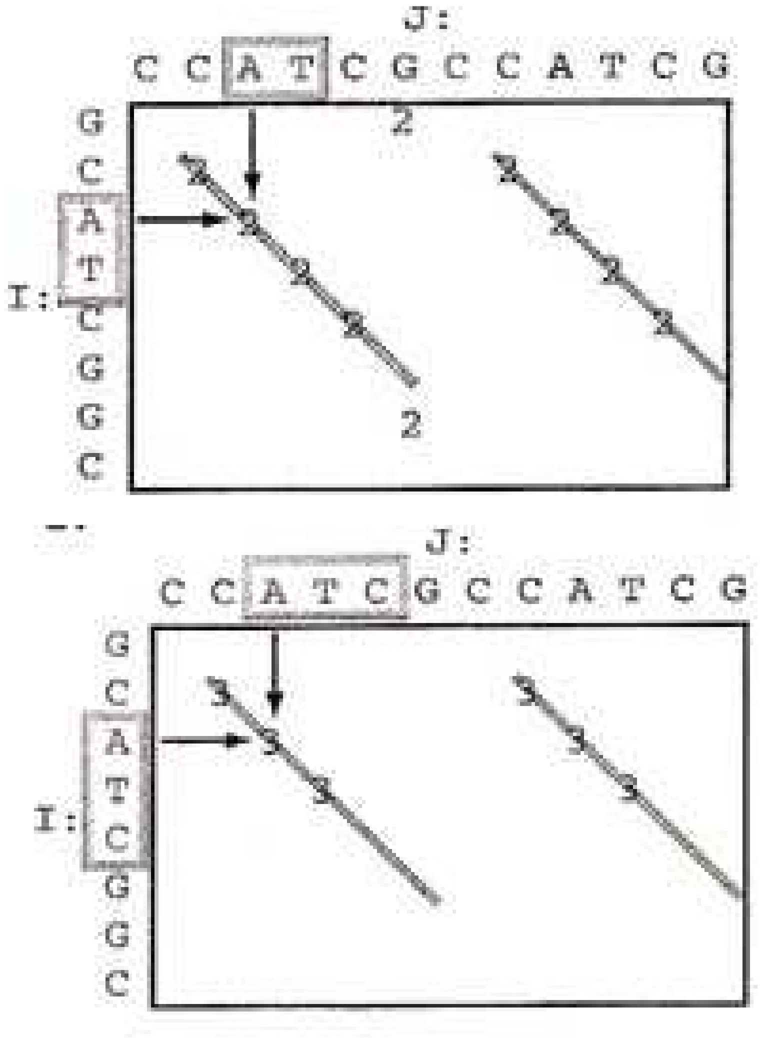
Dot Matrix Comparisons

- When I and J are DNA sequences and are short, the patterns of this type are relatively easy to see.
- When I and J are DNA sequences and very long, there will be many dots in the matrix since, for any letter at position j in J, the probability of having a dot at any position i in I will equal the frequency of the letter J_j in the DNA.
- When I and J are proteins, dots in the matrix elements record matches between amino acid residues at each particular pair of positions.

FASTA: Rationale

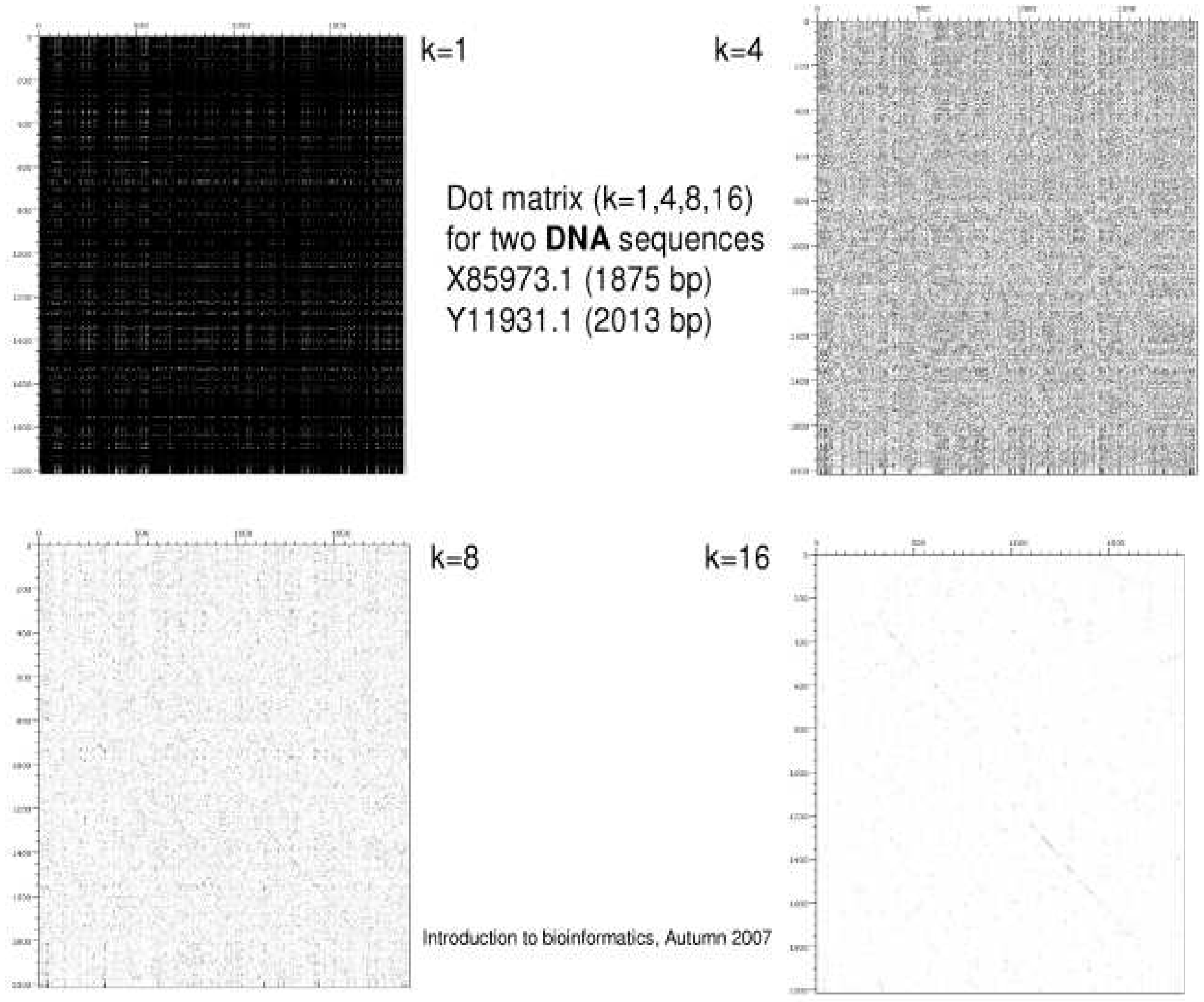
- How is this used within FASTA (Wilbur and Lipman, 1983)?
- The rationale for FASTA can be visualized by considering what happens to a dot matrix plot when we record matches of k-tuples (k > 1)
  instead of recording matches of single letters

  (Fig. 7.1; Deonier et al 2005).



FASTA: Rationale

- We again place entries in the alignment matrix, except this time we only make entries at the first position of each dinucleotide or trinucleotide (k-tuple matches having k = 2 (plotted numerals 2) or k = 3 (plotted numerals 3).
- By looking for words with k > 1, we find that we can ignore most of the alignment matrix since the absence of shared words means that sub-sequences don't match well.
    - There is no need to examine areas of the alignment matrix where there are no word matches. Instead, we only need to focus on the areas around any diagonals.
- Our task is now to compute efficiently diagonal sums of scores, S_l, for diagonals and the question is how can we compute these scores?



Dot matrix (k=1,4,8,16) for two **DNA** sequences X85973.1 (1875 bp) Y11931.1 (2013 bp)

Introduction to bioinformatics, Autumn 2007

The number of matrix entries is reduced as k increases.

FASTA: Rationale

- Consider again the two strings I and J that we used before:

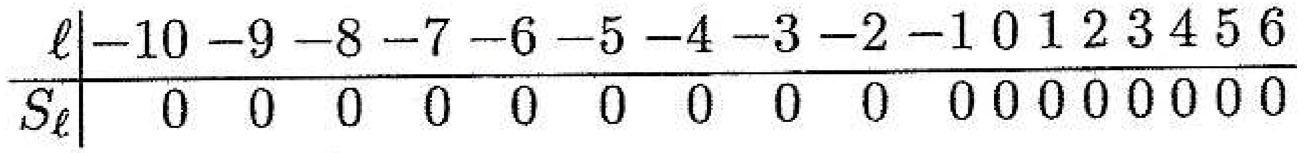
$$J = C\,C\,A\,T\,C\,G\,C\,C\,A\,T\,C\,G$$
$$I = G\,C\,A\,T\,C\,G\,G\,C$$

- Scores can be computed in the following way:
    - Make a k-word list for J.

J		I	
AA		AA	
AC		AC	
AG		AG	
AT	3, 9	AT	3
CA	2, 8	CA	2
CC	1, 7	CC	
CG	5, 11	CG	5
CT		CT	
GA		GA	
GC	6	GC	1, 7
GG		GG	6
GT		GT	
TA		TA	
TC	4, 10	TC	4
TG		TG	
TT		TT	

FASTA: Rationale

- Then initialize all row sums to 0:



ℓ	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
S_ℓ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(Why does l not range from -11 to 7?)

- Next proceed with the 2-words of I, beginning with i = 1, GC. Looking in the list for J, we see that $L_{GC}(J)=\{6\}$, so we know that at l = 1 -6 = -5 there is a 2-word match of GC.
  - Therefore, we replace $S_{-5}= 0$ by $S_{-5} = 0 + 1 = 1$.
- Next, for i = 2, we have $L_{CA}(J)=\{2,8\}$.
  - Therefore replace $S_{2-2} = S_0 = 0$ by $S_0 = 0 + 1$, and replace $S_{2-8} = S_{-6} = 0$ by $S_{-6} = 0 + 1$.

FASTA: Rationale

- These operations, and the operations for all of the rest of the 2-words in I, are summarized below.
  - Note that for each successive step, the then-current score at Si is employed: S_0 was set to 1 in step 2, so l is incremented by 1 in step 3.

$$
\begin{array}{lll}
i = 1,\ \texttt{GC} & L_{\texttt{GC}}(\mathrm{J})=\{6\} & l = 1 - 6 = -5 \\
 & & S_{-5} = 0 \to S_{-5} = 0 + 1 = 1 \\
i = 2,\ \texttt{CA} & L_{\texttt{CA}}(\mathrm{J})=\{2,8\} & l = 2 - 2 = 0 \\
 & & S_0 = 0 \to S_0 = 0 + 1, \text{ and} \\
 & & l = 2 - 8 = -6 \\
 & & S_{-6} = 0 \to S_{-6} = 0 + 1
\end{array}
$$

FASTA: Rationale

$i = 3$, AT	$L_{\mathtt{AT}}$(J)={3,9}	$l = 3 - 3 = 0$
		$S_0 = 1 \rightarrow S_0 = 1 + 1 = 2$
		$l = 3 - 9 = -6$
		$S_{-6} = 1 \rightarrow S_{-6} = 1 + 1 = 2$
$i = 4$, TC	$L_{\mathtt{TC}}$(J)={4,10}	$l = 4 - 4 = 0$
		$S_0 = 2 \rightarrow S_0 = 2 + 1 = 3$
		$l = 4 - 10 = -6$
		$S_{-6} = 2 \rightarrow S_{-6} = 2 + 1 = 3$
$i = 5$, CG	$L_{\mathtt{CG}}$(J)={5,11}	$l = 5 - 5 = 0$
		$S_0 = 3 \rightarrow S_0 = 3 + 1 = 4$
		$l = 5 - 11 = -6$
		$S_{-6} = 3 \rightarrow S_{-6} = 3 + 1 = 4$
$i = 6$, GG	$L_{\mathtt{GG}}$(J)={∅}	$L_{\mathtt{GG}}$(J) is the empty set: no sums are increased
$i = 7$, GC	$L_{\mathtt{GC}}(J) = \{6\}$	$l = 7 - 6 = 1$
		$S_1 = 0 \rightarrow S_1 = 0 + 1 = 1$

ℓ	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
S_ℓ	0	0	0	0	4	1	0	0	0	0	4	1	0	0	0	0	0

FASTA: Rationale

- In conclusion, in our example there are 7 x 11 = 77 potential 2-matches, but in reality there are ten 2-matches with four nonzero diagonal sums.

Where do 7 and 11 come from?

- We have indexed diagonals by the offset, $l = i - j$.
- In this notation, the nonzero diagonal sums are $S_{+1} = 1$, $S_0 = 4$, $S_{-5} = 1$, and $S_{-6} = 4$.

- Notice that we only performed additions when there were 2-word matches.
- It is possible to find local alignments using a gap length penalty of $-gx$ for a gap of length x along a diagonal.
- We are now in good shape to understand the 5 steps involved in FASTA aligning

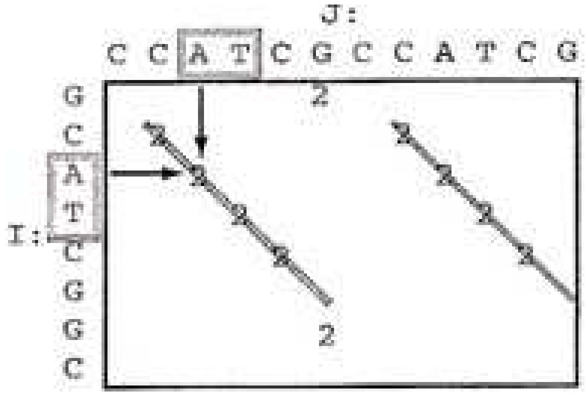
FASTA steps

Five steps are involved in FASTA:

- Step 1: Use the look-up table to identify k-tuple identities between I and J.
- Step 2: Score diagonals containing k-tuple matches, and identify the ten best diagonals in the search space.
- Step 3: Rescore these diagonals using an appropriate scoring matrix (especially critical for proteins), and identify the subregions with the highest score (initial regions).
- Step 4: Join the initial regions with the aid of appropriate joining or gap penalties for short alignments on offset diagonals.
- Step 5: Perform dynamic programming alignment within a band surrounding the resulting alignment from step 4.

FASTA steps in more detail

- Step 1: ***identify k-type identities***
  - To implement the first step, we pass through I once and create a table of the positions i for each possible word of predetermined size k.
  - Then we pass through the search space J once, and for each k-tuple starting at successive positions j, "look up" in the table the corresponding positions for that k-tuple in I.
  - Record the i,j pairs for which matches are found: the i,j pairs define where potential diagonals in the alignment matrix can be found.



FASTA steps in more detail

- Step 2: ***identify high-scoring diagonals***
  - If I has n letters and J has m letters, then there are n+m-1 diagonals.
    - Think of starting in the upper left-hand corner, drawing successive diagonals all the way down, moving your way through the matrix from left to right (m diagonals).
    - Start drawing diagonals through all positions in I (n diagonals).
    - Since you will have counted the diagonal starting at (1,1) twice, you need to subtract 1.
  - Note that we have seen before how to score diagonals (in particular, in our examples, without accounting for gaps).

FASTA steps in more detail

- Step 2: ***identify high-scoring diagonals (continued)***
  - To score the diagonals, calculate the number of k-tuple matches for every diagonal having at least one k-tuple (*identified in step 1*).
    - Scoring may take into account distances between matching k-tuples along the diagonal.
    - Note that the number of diagonals that needs to be scored will be much less than the number of all possible diagonals (r*eduction of search space*).
  - Identify the significant diagonals as those having significantly more k-tuple matches than the mean number of k-tuple matches.
    - For example, if the mean number of 6-tuples is 5 ± 1, then with a threshold of two standard deviations, you might consider diagonals having seven or more 6-tuple matches as significant.
  - Take the top ten significant diagonals.

FASTA steps in more detail

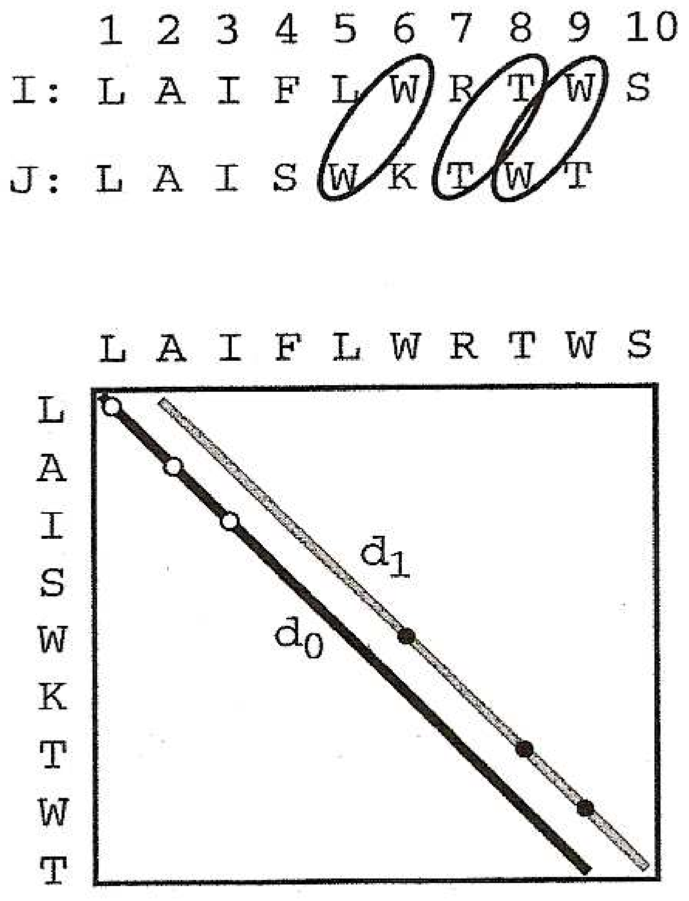
- Step 3: ***Rescore these diagonals***
  - We rescore the diagonals using a scoring table to find subregions with identities shorter than k.
  - Rescoring reveals sequence similarity not detected because of the arbitrary demand for uninterrupted identities of length k.
  - The need for this rescoring is illustrated by the two examples below.

```
I:  C C A T C G C C A T C G        (Number 4-tuple matches: 0)
J:  C C A A C G C A A T C A

I': C C A T C G C C A T C G
J': A C A T C A A A T A A A
```

    - In the first case, the placement of mismatches spaced by three letters means that there are no 4-tuple matches, even though the sequences are 75% identical.
    - The second pair shows one 4-tuple match, but the two sequences are only 33% identical.
  - We retain the subregions with the highest scores.

FASTA steps in more detail

- Step 4: ***Joining diagonals***



j=1, i=1, i-j=0
j=2, i=2, i-j=0
j=3, i=3, i-j=0

j=5, 6=1, i-j=1

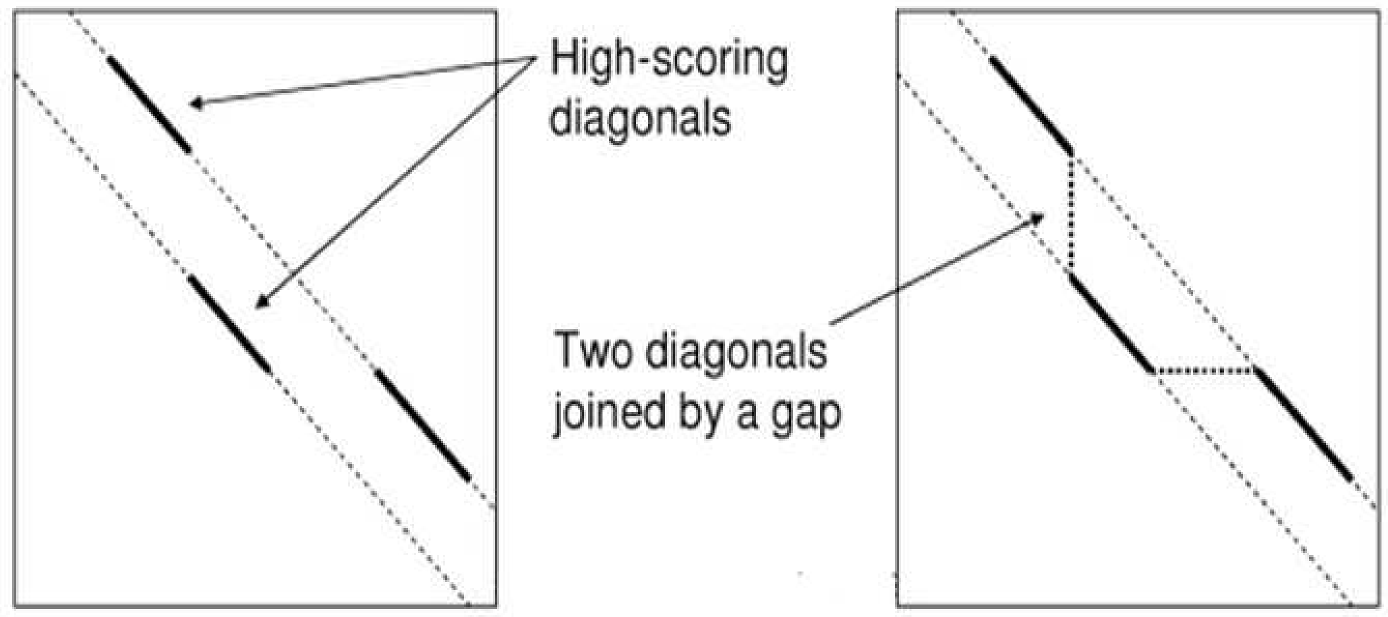
j=7, i=8, i-j=1
j=8, i=9, i-j=1

Offset diagonals

- Diagonals may be offset from each other, if there were a gap in the alignment (i.e., vertical or horizontal displacements in the alignment matrix, as described in the previous chapter).
- Such offsets my indicate indels, suggesting that the local alignments represented by the two diagonals should be joined to form a longer alignment.
- Diagonal d_l is the one having k-tuple matches at positions i in string I and j in string J such that $i – j = l$. As described before, $l = i – j$ is called the offset.

Offset diagonals

- The idea is to find the best-scoring combination of diagonals
- Two offset diagonals can be joined with a gap, if the resulting alignment has a higher score
- Note that different gap open and extension penalties may be are used

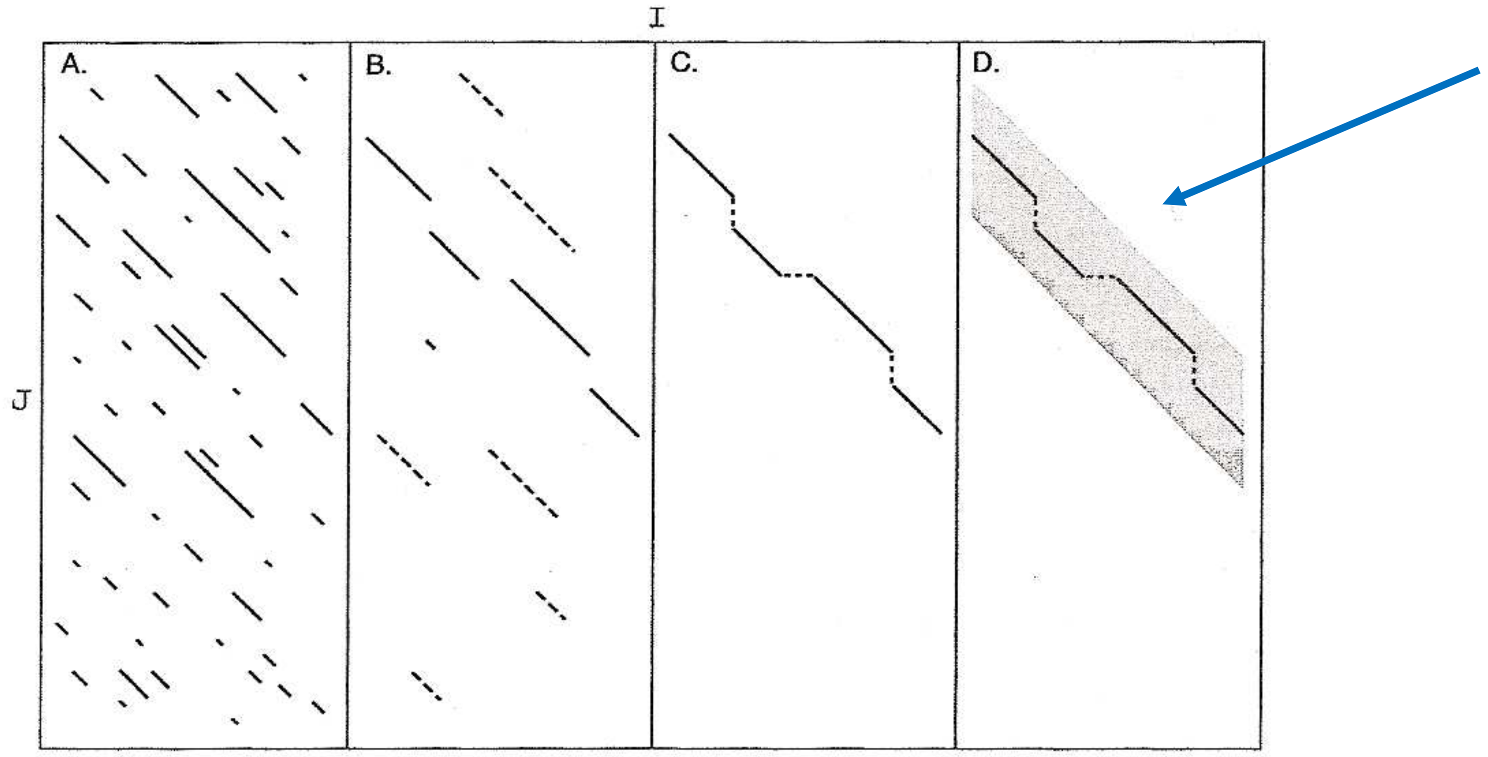


FASTA steps in more detail

- Step 5: ***Smith- Waterman local alignment***
  - The last step of FASTA is to perform local alignment using dynamic programming round the highest-scoring
  - The region to be aligned covers –w and +w offset diagonal to the highest scoring diagonals
  - With long sequences, this region is typically very small compared to the entire nxm matrix (hence, once again, a reduction of the search space was obtained)

FASTA steps in more detail

In FASTA, the alignment step can be restricted to a comparatively narrow window extending +w to the right and -w to the left of the positions included within the highest-scoring diagonal (dynamic programming)



The Smith-Waterman algorithm (local alignment)

- Needleman and Wunsch (1970) were the first to introduce a heuristic alignment algorithm for calculating homology between sequences (global alignment).
- Later, a number of variations have been suggested, among others Sellers (1974) getting closer to fulfilling the requests of biology by measuring the metric distance between sequences [Smith and Waterman, 1981].
- Further development of this led to the Smith-Waterman algorithm based on calculation of local alignments instead of global alignments of the sequences and allowing a consideration of deletions and insertions of arbitrary length.

The Smith-Waterman algorithm

- The Smith-Waterman algorithm uses individual pair-wise comparisons between characters as:

$$M_{i,j} = \max \left\{ \begin{array}{l} M_{i-1,j-1} + s(a_i, b_i) \\ M_{i-1,j} - \delta \\ M_{i,j-1} - \delta \\ 0 \end{array} \right\}.$$

Do you recognize this formula?

- The Smith-Waterman algorithm is the most accurate algorithm when it comes to search databases for sequence homology but it is also the most time consuming, thus there has been a lot of development and suggestions for optimizations and less time-consuming models. One example is BLAST.

Properties of FASTA

- Fast compared to local alignment only using dynamic programming as such
  - A narrow region of the full alignment matrix is aligned
  - With long sequences, this region is typically very small compared to the whole matrix
- Increasing the parameter k (word length), decreases the number of hits
  - It increases specificity
  - It decreases sensitivity
- FASTA can be very specific when identifying long regions of low similarity
  - Specific method does not produce many incorrect results
  - Sensitive method produces many of the correct results

More info at http://www.ebi.ac.uk/fasta
(parameter ktup in the software corresponds to the parameter k in the class notes)

Sensitivity and specificity reminder

- These concepts come from the world of "clinical test assessment"
  - TP = true positives; FP = false positives
  - TN = true negatives; FN = false negatives
- Sensitivity = TP/(TP+FN)
- Specificity = TN/(TN+FP)

	Signal present	**Signal not present**
Signal detected	TP	FP
Signal not detected	FN	TN

FASTA in practice

- Help on nucleotide searches:
  http://www.ebi.ac.uk/2can/tutorials/nucleotide/fasta.html
- Help on interpretation of such search results:
  http://www.ebi.ac.uk/2can/tutorials/nucleotide/fasta1.html

5.e BLAST

- The most used database search programs are BLAST and its descendants.
- BLAST is modestly named for Basic **Local Alignment** Search Tool, and it was introduced in 1990 (Altschul et al., 1990).
- Whereas FASTA speeds up the search by filtering the k-word matches, BLAST employs a quite different strategy (see later).

BLAST steps

In particular, three steps are involved in BLAST:

- Step 1: Find local alignments between the query sequence and a data base sequence ("seed hits")
- Step 2: Extend the seed hits into high-scoring local alignments
- Step 3: Calculate p-values and a rank ordering of the local alignments

BLAST steps in more detail

- Step 1: ***Finding local matches***
  - First, the query sequence is used as a template to construct a set of subsequences of length k that can score at least T when compared with the query.
  - Take as before

$$J = C\,C\,A\,T\,C\,G\,C\,C\,A\,T\,C\,G$$
$$I = G\,C\,A\,T\,C\,G\,G\,C$$

  - To construct a "neighbourhood" of GCATC (collection of words of length 5), we specify for instance T=4: (scoring matches 1, mismatches 0)

$$\left\{\begin{matrix}A\\C\\T\end{matrix}\right\}\mathrm{CATC},\quad \mathrm{G}\left\{\begin{matrix}A\\G\\T\end{matrix}\right\}\mathrm{ATC},\quad \mathrm{GC}\left\{\begin{matrix}C\\G\\T\end{matrix}\right\}\mathrm{TC},\quad \mathrm{GCA}\left\{\begin{matrix}A\\C\\G\end{matrix}\right\}\mathrm{C},\quad \mathrm{GCAT}\left\{\begin{matrix}A\\G\\T\end{matrix}\right\}$$

- Each of these terms (“seeds”) represents three sequences that at least score T=4 when compared to GCATC.
- So in total there are 1 + (3 x 5) = 16 exact matches to search for in J.
- For the three other 5-word patterns in I (CATCG, ATCGG, and TCGGC), there are also 16 exact 5-words, for a total of 4 x 16 = 64 5-word patterns to locate in J.
- A hit is defined as an instance in the search space (database) of a k-word match, within threshold T, of a k-word in the query sequence.

5-words in I	5-words in J	J position(s)	Score
CATCG	CATCG	2, 8	5
GCATC	CCATC	1	4
ATCGG	ATCGC	3	4
TCGGC	TCGCC	4	4

In actual practice, the hits correspond to a tiny fraction of the entire search space.

- Step 2: ***Extending the alignment, starting from the" seed" hits***

  - Starting from any seed hit, the extension includes successive positions, with corresponding increments to the alignment score.
  - Unlike the earlier version of BLAST, gaps can be accommodated in the later versions.
    - With the original version of BLAST, over 90% of the computation time was employed in producing the un-gapped extensions from the hits.
    - This is because the initial step of identifying the seed hits was effective in making this alignment tool very fast.
    - Later versions of BLAST require the same amount of time to find the seed hits and have reduced the time required for the un-gapped extensions considerably.
  - The newer versions of BLAST run about three times faster than the original version (Altschul et aL, 1997).

- Step 3: ***Assess significance***

  - Another aspect of a BLAST analysis is to rank-order by p-values the sequences found
  - If the database is D and a sequence X scores S(D,X) = s against the database, the p-value is $P(S(D,Y) \geq s)$, where Y is a random sequence.
  - The smaller the p-value, the greater the "surprise" and hence the greater the belief that something real has been discovered.
  - A p-value of 0.1 means that with a collection of query sequences picked at random, in 1/10 of the instances a score that is as large or larger would be discovered.
  - A p-value of 10^{-6} means that only once in a million instances would a score of that size appear by chance alone.

E-values and p-values

- *P-value; probability value*:
  this is the probability that a hit would attain at least the given score, by random chance for the search database
- *E value; expectation value*:
  this is the expected number of hits of at least the given score that you would expect by random chance for the search database

  There is a theoretical foundation that shows that, when the E-value is small enough, it is essentially a p-value (small enough ~E<0.10)

E-values and p-values

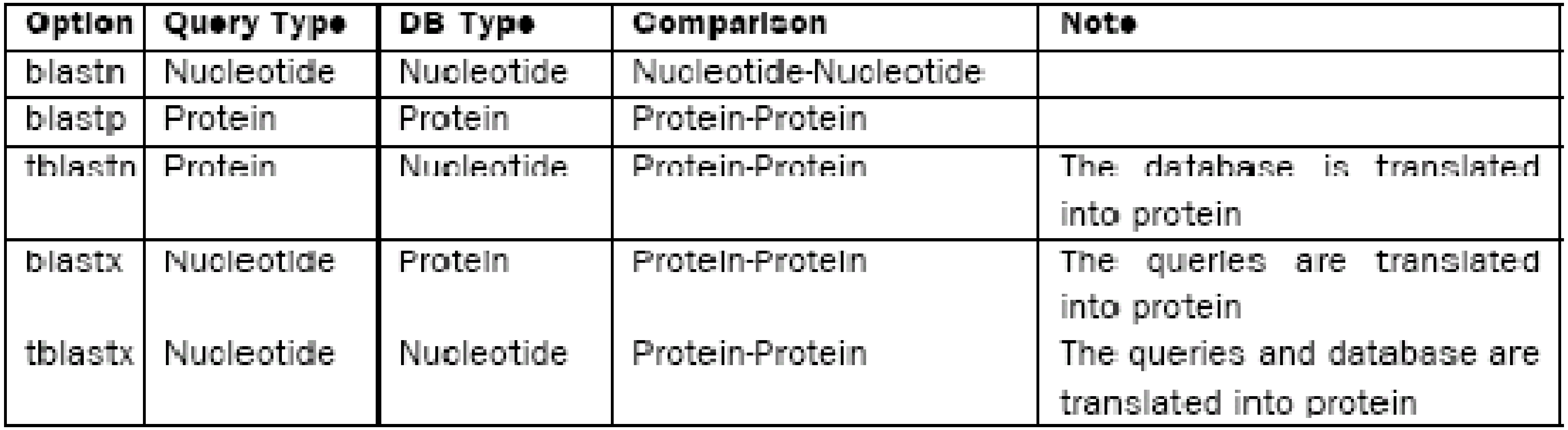
- E-value < 10e-100: Identical sequences.
  - You will get long alignments across the entire query and hit sequence.
- 10e-50 < E-value < 10e-100: Almost identical sequences.
  - A long stretch of the query protein is matched to the database.
- 10e-10 < E-value < 10e-50: Closely related sequences.
  - Could be a domain match or similar.
- 1 < E-value < 10e-6: Could be a true homologue but it is a gray area.

BLAST properties

- BLAST is extremely fast. It can be on the order of 50-100 times faster than the Smith-Waterman (local alignment) approach
- Its main idea is built on the conjecture that homologous sequences are likely to contain a short high-scoring similarity region, a hit. Each hit gives a seed that BLAST tries to extend on both sides.
- It is preferred over FASTA for large database searches
- There exists a statistical theory for assessing significance

BLAST properties

- Many variants exist to the initial BLAST theme ... Depending on the nature of the sequence it is possible to use different BLAST programs for the database search.
- There are five versions of the BLAST program: BLASTN, BLASTP, BLASTX, TBLASTN,TBLASTX:



Option	Query Type	DB Type	Comparison	Note
blastn	Nucleotide	Nucleotide	Nucleotide-Nucleotide	
blastp	Protein	Protein	Protein-Protein	
tblastn	Protein	Nucleotide	Protein-Protein	The database is translated into protein
blastx	Nucleotide	Protein	Protein-Protein	The queries are translated into protein
tblastx	Nucleotide	Nucleotide	Protein-Protein	The queries and database are translated into protein

BLAST in practice

- Relevant questions include:
  - What inferences about this sequence can you make from this information?
  - What is the identity of the sequence?
  - What gene do you think it encodes?
  - What organism do you think it comes from?
  - How reliable do you think this inference is? Why?
- More info via http://www.ncbi.nlm.nih.gov/BLAST/blast_overview.html

Should I use Smith-Waterman or other algorithms for sequence similarity searching?

- The Smith-Waterman algorithm is quite time demanding because of the search for optimal local alignments, and it also imposes some requirements on the computer's memory resources as the comparison takes place on a character-to-character basis.
- The fact that similarity searches using the Smith-Waterman algorithm take a lot of time often prevents this from being the first choice, even though it is the most precise algorithm for identifying homologous regions between sequences.
- A combination of the Smith-Waterman algorithm with a reduction of the search space (such as k-word identification in FASTA) may speed up the process.

Should I use Smith-Waterman or other algorithms for sequence similarity searching?

- BLAST and FASTA are heuristic approximations of dynamic programming algorithms. These approximations are less sensitive (then f.i. Smith-Waterman) and do not guarantee to find the best alignment between two sequences. However, these methods are not as time-consuming as they reduce computation time and CPU usage [Shpaer et al., 1996].
- Hence, the researcher needs to make a choice between
  - Having a fast and effective data analysis (BLAST)
  - Reducing the risk of missing important information by using the most sensitive algorithms for data base searching (Smith-Waterman / FASTA)

Should I use Smith-Waterman or other algorithms for sequence similarity searching?

- Through the Japanese Institute of Bioinformatics Research and Development (BIRD) a public available software version of Smith-Waterman, SSEARCH, is accessible: http://www-btls.jst.go.jp/cgi-bin/Tools/SSEARCH/index.cgi.
- There are also commercial software packages available which perform Smith-Waterman searches.
- Remember that the result of a Smith-Waterman algorithm searching will be only returning one result for each pair of compared sequences: the most optimal alignment

References:

- Deonier et al. *Computational Genome Analysis*, 2005, Springer.
  (Chapters 6,7)

Preparatory Reading:

- Foster et al 2004. Beyond race : towards a whole genome perspective on human populations and genetic variation. Nature Reviews Genetics 5: 790-.
- Balding 2006. A tutorial on statistical methods for population association studies. Nature Reviews Genetics 7: 781-.