Sequence Alignment

Armstrong, 2007

Why?

- Genome sequencing gives us new gene sequences
- Network biology gives us functional information on genes/proteins
- Analysis of mutants links unknown genes to diseases
- *Can we learn anything from other known sequences about our new gene/protein?*

Armstrong, 2007

What is it?

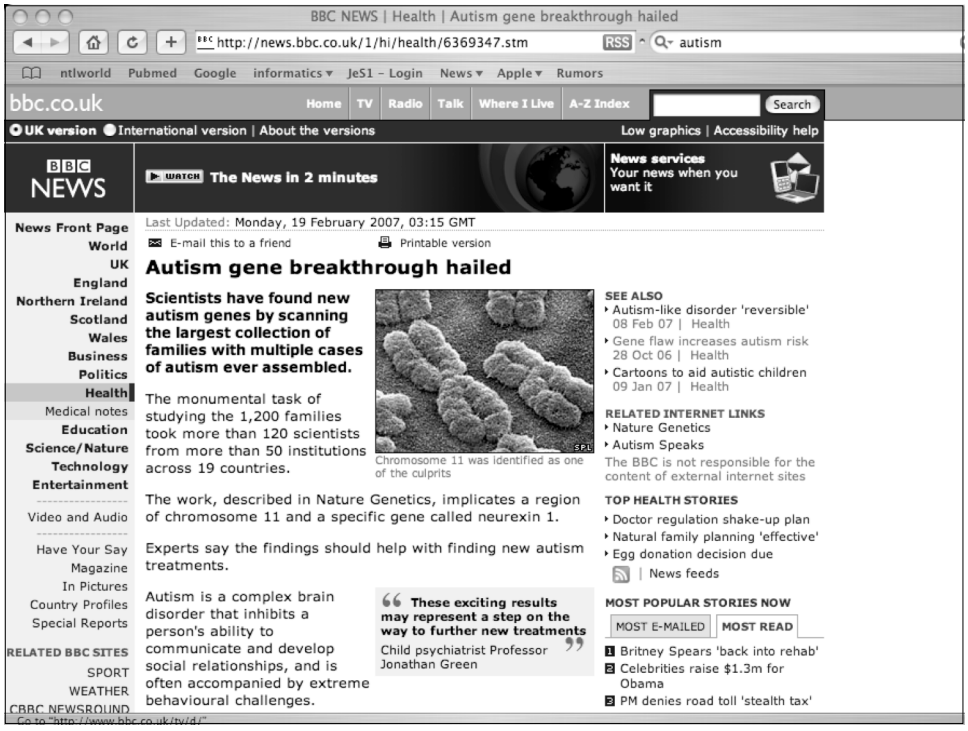
ACCGGTATCCTAGGAC

ACCTATCTTAGGAC

Are these two sequences related?

How similar (or dissimilar) are they?

Armstrong, 2007



What is it?

```
ACCGGTATCCTAGGAC
|||  ||||  ||||||
ACC--TATCTTAGGAC
```

- Match the two sequences as closely as possible = aligned
- Therefore, alignments need a score

Armstrong, 2007

Why do we care?

- DNA and Proteins are based on linear sequences
- Information is encoded in these sequences
- All bioinformatics at some level comes back to matching sequences that might have some noise or variability

Armstrong, 2007

Alignment Types

- Global: used to compare to similar sized sequences.
  - Compare closely related genes
  - Search for mutations or polymorphisms in a sequence compared to a reference.

Armstrong, 2007

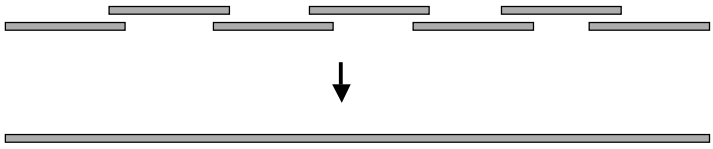
Alignment Types

- Local: used to find shared subsequences.
  - Search for protein domains
  - Find gene regulatory elements
  - Locate a similar gene in a genome sequence.

Armstrong, 2007

Alignment Types

- Ends Free: used to find joins/overlaps.
  - Align the sequences from adjacent sequencing primers.



Armstrong, 2007

How do we score alignments?

```
ACCGGTATCCTAGGAC
|||   ||||  ||||||
ACC--TATCTTAGGAC
```

- Assign a score for each match along the sequence.

Armstrong, 2007

How do we score alignments?

```
ACCGGTATCCTAGGAC
|||  |||| ||||||
ACC--TATCTTAGGAC
```

- Assign a score (or penalty) for each substitution.

Armstrong, 2007

How do we score alignments?

```
ACCGGTATCCTAGGAC
|||  |||| ||||||
ACC--TATCTTAGGAC
```

- Assign a score (or penalty) for each insertion or deletion.
- insertions/deletions otherwise known as indels

Armstrong, 2007

How do we score alignments?

```
ACCGGTATCCTAGGAC
|||  ||||  ||||||
ACC--TATCTTAGGAC
```

- Matches and substitutions are ‘easy’ to deal with.
  - We’ll look at substitution matrices later.
- How do we score indels: gaps?

Armstrong, 2007

How do we score gaps?

```
ACCGGTATCC---GAC
|||  ||||    |||
ACC--TATCTTAGGAC
```

- A gap is a consecutive run of indels
- The gap length is the number of indels.
- The simple example here has two gaps of length 2 and 3

Armstrong, 2007

How do we score gaps?

```
ACCGGTATCC---GAC
|||   ||||     |||
ACC--TATCTTAGGAC
```

- Constant: Length independent weight
- Affine: *Open* and *Extend* weights.
- Convex: Each additional gap contributes less
- Arbitrary: Some arbitrary function on length

Armstrong, 2007

Choosing Gap Penalties

- The choice of Gap Scoring Penalty is very sensitive to the context in which it is applied:
  - introns vs exons
  - protein coding regions
  - mis-matches in PCR primers

Armstrong, 2007

Substitution Matrices

- Substitution matrices are used to score substitution events in alignments.
- Particularly important in Protein sequence alignments but relevant to DNA sequences as well.
- Each scoring matrix represents a particular theory of evolution

Armstrong, 2007

Similarity/Distance

- Distance is a measure of the cost or replacing one residue with another.
- Similarity is a measure of how similar a replacement is.

e.g. replacing a hydrophobic residue with a hydrophilic one.

- The logic behind both are the same and the scoring matrices are interchangeable.

Armstrong, 2007

DNA Matrices

Identity matrix

	A	C	G	T
A	1	0	0	0
C	0	1	0	0
G	0	0	1	0
T	0	0	0	1

BLAST

	A	C	G	T
A	5	−4	−4	−4
C	−4	5	−4	−4
G	−4	−4	5	−4
T	−4	−4	−4	5

However, some changes are more likely to occur than others (even in DNA). When looking at distance, the ease of mutation is a factor. a.g. A-T and A-C replacements are rarer than A-G or C-T.

Armstrong, 2007

Protein Substitution Matrices

How can we score a substitution in an aligned sequence?

- Identity matrix like the simple DNA one.
- Genetic Code Matrix:

  For this, the score is based upon the minimum number of DNA base changes required to convert one amino acid into the other.

Armstrong, 2007

Second base of codon

First base of codon	U	C	A	G	Third base of codon
U	UUU Phe, UUC Phe, UUA Leu, UUG Leu	UCU SER, UCC SER, UCA SER, UCG SER	UAU Tyr, UAC Tyr, UAA, UAG	UGU Cys, UGC Cys, UGA, UGG Trp	U, C, A, G
C	CUU Leu, CUC Leu, CUA Leu, CUG Leu	CCU Pro, CCC Pro, CCA Pro, CCG Pro	CAU His, CAC His, CAA Gln, CAG Gln	CGU Arg, CGC Arg, CGA Arg, CGG Arg	U, C, A, G
A	AUU Ile, AUC Ile, AUA Ile, AUG Met	ACU Thy, ACC Thy, ACA Thy, ACG Thy	AAU Asn, AAC Asn, AAA Lys, AAG Lys	AGU Ser, AGC Ser, AGA Arg, AGG Arg	U, C, A, G
G	GUU Val, GUC Val, GUA Val, GUG Val	GCU Ala, GCC Ala, GCA Ala, GCG Ala	GAU Asp, GAC Asp, GAA Glu, CAG Glu	GGU Gly, GGC Gly, GGA Gly, GGG Gly	U, C, A, G

The genetic code, written by convention in the form in which the Codons appear in mRNA. The three terminator codons, UAA, UAG, and UGA, are boxed in red; the AUG initiator codon is shown in green.

Armstrong, 2007

Protein Substitution Matrices

How can we score a substitution in an aligned sequence?

- Amino acid property matrix

  Assign arbitrary values to the relatedness of different amino acids:

  e.g. hydrophobicity , charge, pH, shape, size

Armstrong, 2007

Matrices based on Probability

$$S_{ij} = \log (q_{ij}/p_i p_j)$$

Sij is the log odds ratio of two probabilities: amino acids i and j are aligned by evolutionary descent and the probability that they aligned at random**.**

This is the basis for commonly used substitution matrices.

Armstrong, 2007

PAM matrices

Dayhoff, Schwarz and Orcutt 1978 took these into consideration when constructing the PAM matrices:

Took 71 protein families - where the sequences differed by no more than 15% of residues (i.e. 85% identical)

- Aligned these proteins
- Build a theoretical phylogenetic tree
- Predicted the most likely residues in the ancestral sequence

Armstrong, 2007

PAM Matrices

- Ignore evolutionary direction
- Obtained frequencies for residue X being substituted by residue Y over time period Z
- Based on 1572 residue changes
- They defined a substitution matrix as 1 PAM (point accepted mutation) if the expected number of substitutions was 1% of the sequence length.

Armstrong, 2007

PAM Matrices

To increase the distance, they multiplied the the PAM1 matrix.

PAM250 is one of the most commonly used.

Armstrong, 2007

PAM - notes

The PAM matrices are rooted in the original datasets used to create the theoretical trees

They work well with closely related sequences

Based on data where substitutions are most likely to occur from single base changes in codons.

Armstrong, 2007

PAM - notes

Biased towards conservative mutations in the DNA sequence (rather than amino acid substitutions) that have little effect on function/structure.

Replacement at any site in the sequence depends only on the amino acid at that site and the probability given by the table.This does not represent evolutionary processes correctly. Distantly related sequences usually have regions of high conservation (blocks).

Armstrong, 2007

PAM - notes

36 residue pairs were not observed in the dataset used to create the original PAM matrix

A new version of PAM was created in 1992 using 59190 substitutions: Jones, Taylor and Thornton 1992 CAMBIOS 8 pp 275

BLOSUM matrices

Henikoff and Henikoff 1991

Took sets of aligned ungapped regions from protein families from the BLOCKS database.

The BLOCKS database contain short protein sequences of high similarity clustered together. These are found by applying the MOTIF algorithm to the SWISS-PROT and other databases. The current release has 8656 Blocks.

BLOSUM matrices

Sequences were clustered whenever the %identify exceeded some percentage level.

Calculated the frequency of any two residues being aligned in one cluster also being aligned in another

Correcting for the size of each cluster.

Armstrong, 2007

BLOSUM matrices

Resulted in the fraction of observed substitutions between any two residues over all observed substitutions.

The resulting matrices are numbered inversely from the PAM matrices so the BLOSUM50 matrix was based on clusters of sequence over 50% identity, and BLOSUM62 where the clusters were at least 62% identical.

Armstrong, 2007

BLOSUM 62 Matrix

Armstrong, 2007

Summary so far…

- Gaps
  - Indel operations
  - Gap scoring methods
- Substitution matrices
  - DNA largely simple matrices
  - Protein matrices are based on probability
  - PAM and BLOSUM

Armstrong, 2007

How do we do it?

- Like everything else there are several methods and choices of parameters
- The choice depends on the question being asked
  - What kind of alignment?
  - Which substitution matrix is appropriate?
  - What gap-penalty rules are appropriate?
  - Is a heuristic method good enough?

Armstrong, 2007

Working Parameters

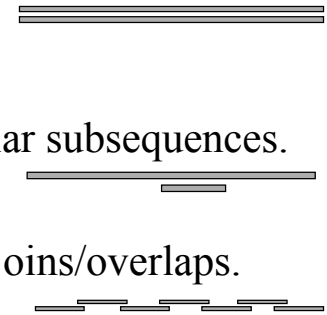
- For proteins, using the affine gap penalty rule and a substitution matrix:

Query Length	Matrix	Gap (open/extend)
<35	PAM-30	9,1
35-50	PAM-70	10,1
50-85	BLOSUM-80	10,1
>85	BLOSUM-62	11,1

Armstrong, 2007

Alignment Types

- Global: used to compare to similar sized sequences.
- Local: used to find similar subsequences.
- Ends Free: used to find joins/overlaps.



Armstrong, 2007

Global Alignment

- Two sequences of similar length
- Finds the best alignment of the two sequences
- Finds the score of that alignment
- Includes **ALL** bases from both sequences in the alignment and the score.
- Needleman-Wunsch algorithm

Armstrong, 2007

Needleman-Wunsch algorithm

- Gaps are inserted into, or at the ends of each sequence.
- The sequence length (bases+gaps) are identical for each sequence
- Every base or gap in each sequence is aligned with a base or a gap in the other sequence

Armstrong, 2007

Needleman-Wunsch algorithm

- Consider 2 sequences *S* and *T*
- Sequence *S* has *n* elements
- Sequence *T* has *m* elements
- Gap penalty ?

Armstrong, 2007

How do we score gaps?

```
ACCGGTATCC---GAC
|||   ||||     |||
ACC--TATCTTAGGAC
```

- Constant: Length independent weight
- Affine: *Open* and *Extend* weights.
- Convex: Each additional gap contributes less
- Arbitrary: Some arbitrary function on length
  - Lets score each gap as –1 times length

Armstrong, 2007

Needleman-Wunsch algorithm

- Consider 2 sequences *S* and *T*
- Sequence *S* has *n* elements
- Sequence *T* has *m* elements
- Gap penalty –1 per base (arbitrary gap penalty)
- An alignment between base *i* in *S* and a gap in *T* is represented: $(S_i,-)$
- The score for this is represented : $\sigma(S_i,-) = -1$

Armstrong, 2007

Needleman-Wunsch algorithm

- Substitution/Match matrix for a simple alignment
- Several models based on probability….

	A	C	G	T
A	2	-1	-1	-1
C	-1	2	-1	-1
G	-1	-1	2	-1
T	-1	-1	-1	2

Armstrong, 2007

Needleman-Wunsch algorithm

- Substitution/Match matrix for a simple alignment
- Simple identify matrix (2 for match, -1 for mismatch)
- An alignment between base *i* in *S* and base *j* in *T* is represented: (S_i,T_j)
- The score for this occurring is represented: $\sigma(S_i,T_j)$

Armstrong, 2007

Needleman-Wunsch algorithm

- Set up a array V of size n+1 by m+1
- Row 0 and Column 0 represent the cost of adding gaps to either sequence at the start of the alignment
- Calculate the rest of the cells row by row by finding the optimal route from the surrounding cells that represent a gap or match/mismatch
  - This is easier to demonstrate than to explain

Needleman-Wunsch algorithm

- lets start by trying out a simple example alignment:

S = ACCGGTAT

T = ACCTATC

Needleman-Wunsch algorithm

– Get lengths

S = ACCGGTAT

T = ACCTATC

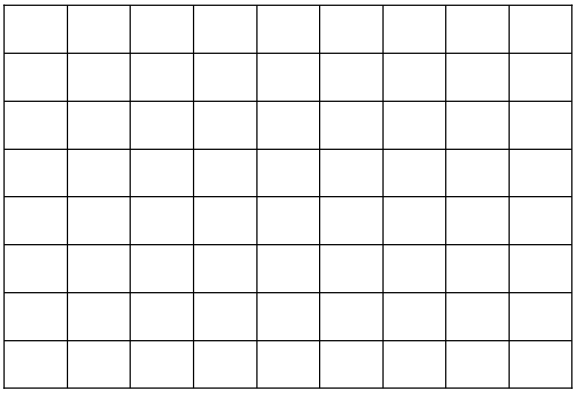
Length of $S = m = 8$

Length of T = $n = 7$

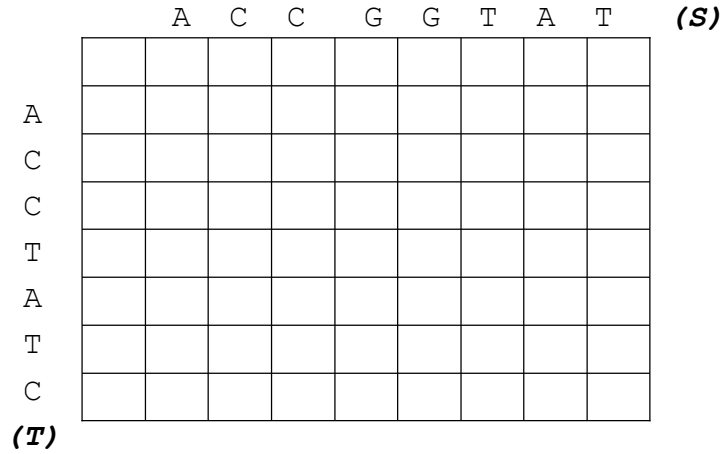
(lengths approx equal so OK for Global Alignment)

Create array m+1 by n+1

(i.e. 9 by 8)

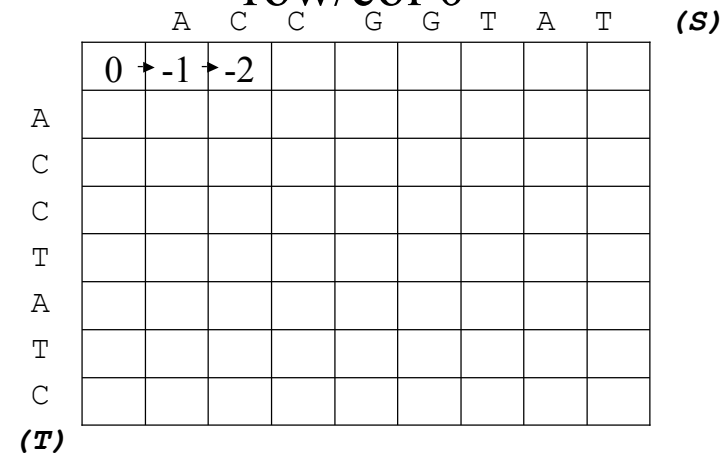


Add on bases from each sequence



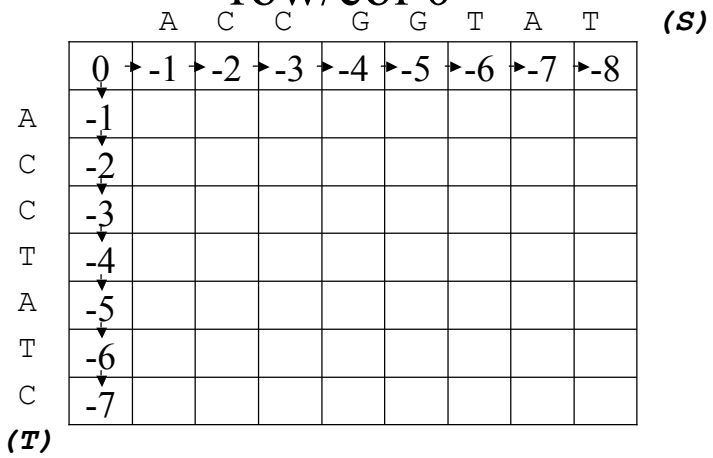
Armstrong, 2007

Represent scores for gaps in row/col 0



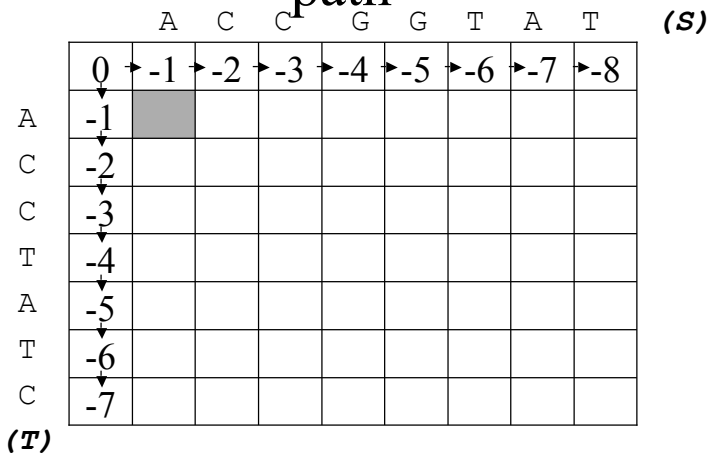
Armstrong, 2007

Represent scores for gaps in row/col 0



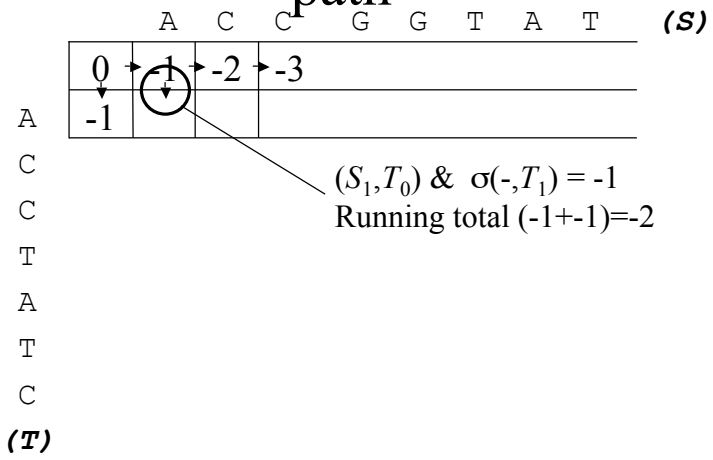
Armstrong, 2007

For each cell consider the 'best' path



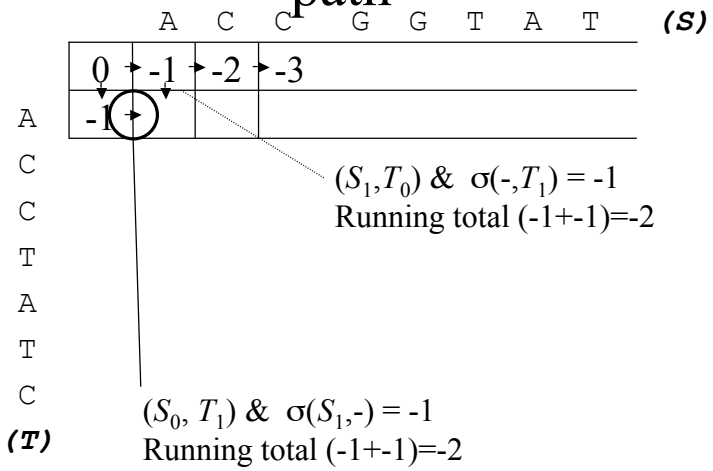
Armstrong, 2007

For each cell consider the 'best' path



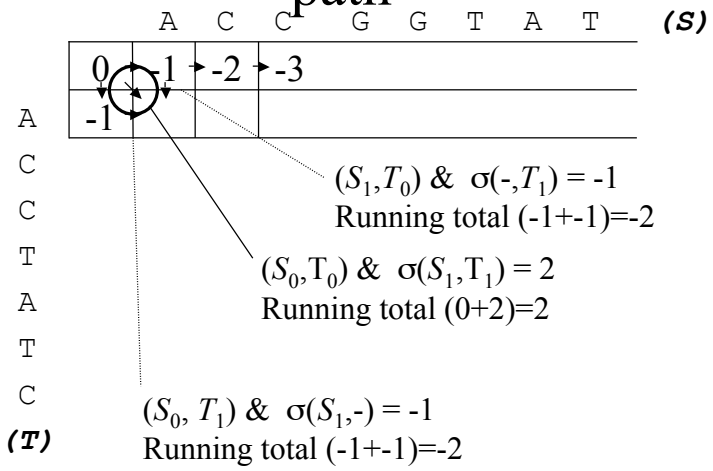
Armstrong, 2007

For each cell consider the 'best' path



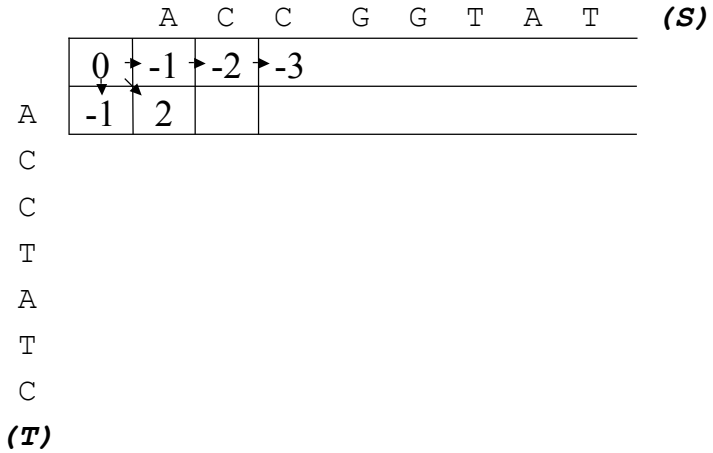
Armstrong, 2007

For each cell consider the 'best' path



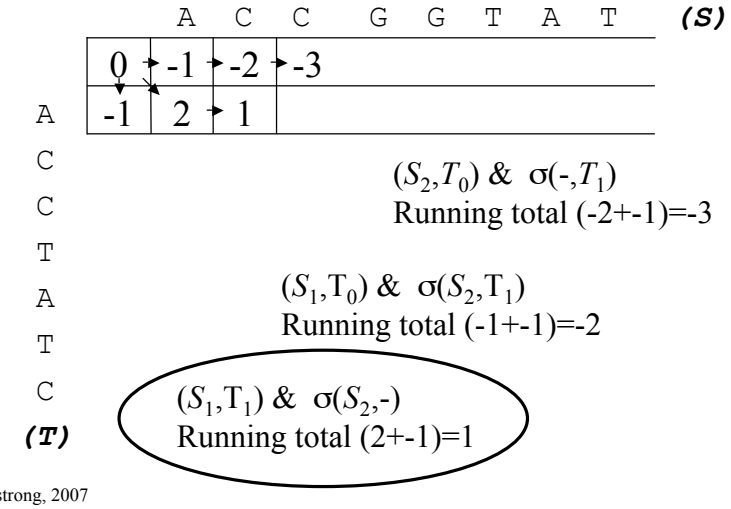
Armstrong, 2007

Choose and record 'best' path



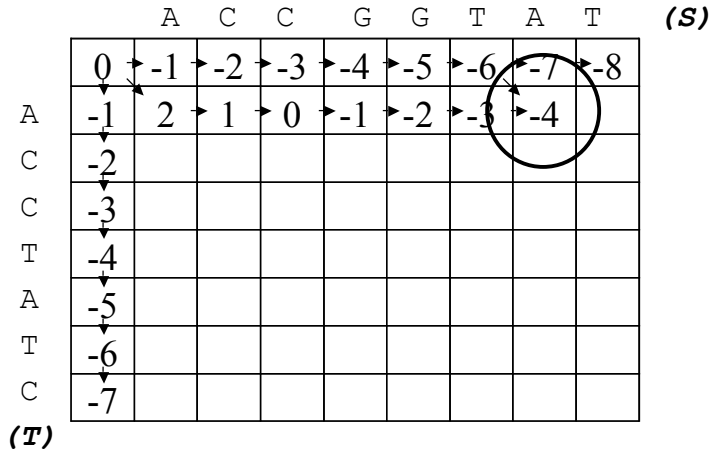
Armstrong, 2007

Choose and record 'best' path



Armstrong, 2007

Continue….



Armstrong, 2007

Continue….

		A	C	C	G	G	T	A	T	**(S)**
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3									
T	-4									
A	-5									
T	-6									
C	-7									
(T)										

Armstrong, 2007

Continue….

		A	C	C	G	G	T	A	T	**(S)**
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4									
A	-5									
T	-6									
C	-7									
(T)										

Armstrong, 2007

Continue….

		A	C	C	G	G	T	A	T	**(S)**
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5									
T	-6									
C	-7									
(T)										

Armstrong, 2007

Continue….

		A	C	C	G	G	T	A	T	**(S)**
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6									
C	-7									
(T)										

Armstrong, 2007

Continue….

		A	C	C	G	G	T	A	T	**(S)**
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7									
(T)										

Armstrong, 2007

Finally.

		A	C	C	G	G	T	A	T	**(S)**
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	**= Score**
(T)										

Armstrong, 2007

Finally.

		A	C	C	G	G	T	A	T	(S)
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

We recreate the alignment using by following the pointers back through the array to the origin

		A	C	C	G	G	T	A	T	(S)
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

- (*S*)

C (*T*)

		A	C	C	G	G	T	A	T	*(S)*
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

T- (*S*)

|

TC (*T*)

		A	C	C	G	G	T	A	T	*(S)*
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

AT- (*S*)
||
ATC (*T*)

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

(S)

(T)

Armstrong, 2007

TAT- (*S*)
|||
TATC (*T*)

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

(S)

(T)

Armstrong, 2007

GTAT- (*S*)
|||
-TATC (*T*)

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

(T) ***(S)***

Armstrong, 2007

GGTAT- (*S*)
|||
--TATC (*T*)

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

(T) ***(S)***

Armstrong, 2007

CGGTAT- (*S*)
| |||
C--TATC (*T*)

		A	C	C	G	G	T	A	T	***(S)***
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

CCGGTAT- (*S*)
|| |||
CC--TATC (*T*)

		A	C	C	G	G	T	A	T	***(S)***
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

ACCGGTAT- (*S*)
||| |||
ACC--TATC (*T*)

		A	C	C	G	G	T	A	T	***(S)***
	0	-1	-2	-3	-4	-5	-6	-7	-8	
A	-1	2	1	0	-1	-2	-3	-4	-5	
C	-2	1	4	3	2	1	0	-1	-2	
C	-3	0	3	6	5	4	3	2	1	
T	-4	-1	2	5	4	4	6	5	4	
A	-5	-2	1	4	4	3	5	8	7	
T	-6	-3	0	3	3	3	5	7	10	
C	-7	-4	-1	2	2	2	4	6	**9**	
(T)										

Armstrong, 2007

Checking the result

ACCGGTAT- (*S*)
||| |||
ACC--TATC (*T*)

- Our alignment considers ALL bases in each sequence
- 6 matches = 12 points, 3 gaps = -3 points
- Score = 9 confirmed.

Armstrong, 2007

A bit more formally..

Base conditions:

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

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

Recurrence relation: for $1<=i<=n$, $1<=j<=m$:

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

Armstrong, 2007

Time Complexity

- Each cell is dependant on three others and the two relevant characters in each sequence
- Hence each cell takes a constant time
- $(n+1)$ x $(m+1)$ cells

- Complexity is therefore $O(nm)$

Armstrong, 2007

Space Complexity

- To calculate each row we need the current row and the row above only.
- Therefore to get the score, we need O(n+m) space

- However, if we need the pointers as well, this increases to O(nm) space
- This is a problem for very long sequences
  - think about the size of whole genomes

Armstrong, 2007

Global alignment in linear space

- Hirschberg 1977 applied a 'divide and conquer' algorithm to Global Alignment to solve the problem in linear space.
- Divide the problem into small manageable chunks
- The clever bit is finding the chunks

Armstrong, 2007

dividing...

Compute matrix V(A,B) saving the values for $^{n}/_{2}{}^{th}$ row
- call this matrix F

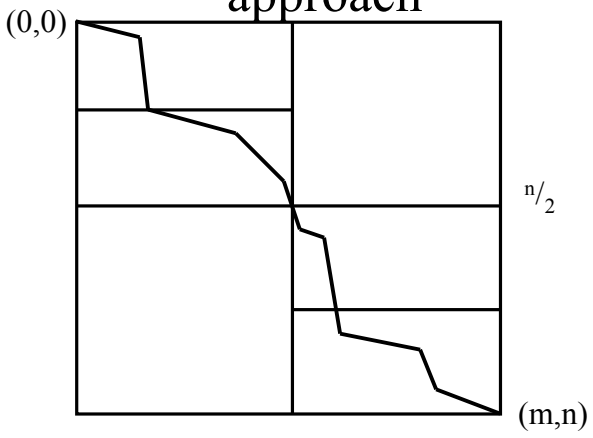
Compute matrix V(A^r,B^r) saving the values for $^{n}/_{2}{}^{th}$ row
- call this matrix B

Find column *k* so that the crossing point ($^{n}/_{2}$,*k*) satisfies:
F($^{n}/_{2}$,*k*) + B($^{n}/_{2}$,m-*k*) = F(n,m)

Now we have two much smaller problems:
(0,0) -> ($^{n}/_{2}$,*k*) and (n,m) -> ($^{n}/_{2}$,*m*-*k*)

Armstrong, 2007

Hirschberg's divide and conquer approach



Armstrong, 2007

Complexity

- After applying Hirschberg's divide and conquer approach we get the following:
  - Complexity O(mn)
  - Space O(min(m,n))
- For the proofs, see D.S. Hirschberg. (1977) Algorithms for the longest common subsequence problem. J. A.C.M 24: 664-667

OK where are we?

- The Needleman-Wunsch algorithm finds the optimum alignment and the best score.
  - NW is a dynamic programming algorithm
- Space complexity is a problem with NW
- Addressed by a divide and conquer algorithm
- What about local and ends-free alignments?

Smith-Waterman algorithm

- Between two sequences, find the best two subsequences and their score.
- We want to ignore badly matched sequence
- Use the same types of substitution matrix and gap penalties
- Use a modification of the previous dynamic programming approach.

Armstrong, 2007

Smith-Waterman algorithm

- If S_i matches T_j then $\sigma(S_i,T_j) >= 0$
- If they do not match or represent a gap then <=0

- Lowest allowable value of any cell is 0
- Find the cell with the highest value (i,j) and extend the alignment back to the first zero value
- The score of the alignment is the value in that cell
- A quick example if best...

Armstrong, 2007

min value of any cell is 0

		A	C	C	G	G	T	A	T	*(S)*
	0	0	0	0	0	0	0	0	0	
T	0									
T	0									
G	0									
T	0									
A	0									
T	0									
C	0									
(T)										

Armstrong, 2007

min value of any cell is 0

		A	C	C	G	G	T	A	T	*(S)*
	0	0	0	0	0	0	0	0	0	
T	0	0	0	0	0	0	2	1	2	
T	0	0	0	0	0	0	2	1	3	
G	0									
T	0									
A	0									
T	0									
C	0									
(T)										

Armstrong, 2007

min value of any cell is 0

		A	C	C	G	G	T	A	T	(S)
	0	0	0	0	0	0	0	0	0	
T	0	0	0	0	0	0	2	1	2	
T	0	0	0	0	0	0	2	1	3	
G	0	0	0	0	2	2	1	1	2	
T	0	0	0	0	1	1	4	3	3	
A	0	2	1	0	0	0	3	6	5	
T	0	1	1	0	0	0	2	5	8	
C	0	0	3	4	3	2	1	4	7	
(T)										

Armstrong, 2007

Find biggest cell and map alignment from there

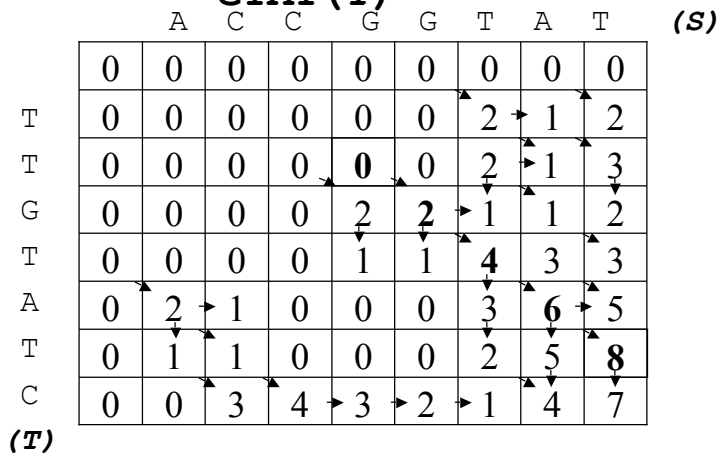
		A	C	C	G	G	T	A	T	(S)
	0	0	0	0	0	0	0	0	0	
T	0	0	0	0	0	0	2	1	2	
T	0	0	0	0	**0**	0	2	1	3	
G	0	0	0	0	2	**2**	1	1	2	
T	0	0	0	0	1	1	**4**	3	3	
A	0	2	1	0	0	0	3	**6**	5	
T	0	1	1	0	0	0	2	5	**8**	
C	0	0	3	4	3	2	1	4	7	
(T)										

Armstrong, 2007

GTAT (*S*)

| | | |

GTAT (*T*)



Armstrong, 2007

Smith-Waterman cont'd

- Complexity
  - Time is O(nm) as in global alignments
  - Space is O(nm) as in global alignments

  - A mod of Hirschbergs algorithm allows O(n+m) (n+m) as two rows need to be stored at a time instead of one as in the global alignment.

Armstrong, 2007

A bit more formally..

Base conditions: $\forall i,j.\ V(i,0) = 0,\ V(0,j) = 0$

Recurrence relation: for $1<=i<=n$, $1<=j<=m$:

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

Compute i^* and j^* $V(i^*,j^*) = \max_{1<=i<=n,1<=j<=m} V(i,j)$

Armstrong, 2007

Ends-free alignment

- Find the overlap between two sequences such start the start of one overlaps is in the alignment and the end of the other is in the alignment.
- Essential to DNA sequencing strategies.
  - Building genome fragments out of shorter sequencing data.
- Another variant of the Global Alignment Problem

Armstrong, 2007

Ends-free alignment

- Set the initial conditions to zero weight
  - allow indels/gaps at the ends without penalty
- Fill the array/table using the same recursion model used in global/local alignment
- Find the best alignment that ends in one row or column
  - trace this back

Armstrong, 2007

min value row0 & col0 is 0

		G	T	T	A	C	T	G	T	***(S)***
	0	0	0	0	0	0	0	0	0	
C	0	-1	-1	-1	-1	2	1	0	-1	
T	0	-1	1	1	0	1	4	3	2	
G	0	2	1	0	0	0	3	6	5	
T	0	1	4	3	2	1	2	5	8	
A	0	0	3	3	5	4	3	4	7	
T	0	-1	2	5	4	4	6	5	6	
C	0	0	1	4	4	6	5	5	5	
(T)										

Armstrong, 2007

Find the best ‘end’ point in an end col or row

		G	T	T	A	C	T	G	T	(S)
	0	0	0	0	0	0	0	0	0	
C	0	-1	-1	-1	-1	2	1	0	-1	
T	0	-1	1	1	0	1	4	3	2	
G	0	2	1	0	0	0	3	6	5	
T	0	1	4	3	2	1	2	5	8	
A	0	0	3	3	5	4	3	4	7	
T	0	-1	2	5	4	4	6	5	6	
C	0	0	1	4	4	6	5	5	5	
(T)										

Armstrong, 2007

Trace the best route from there to the origin and end

		G	T	T	A	C	T	G	T	(S)
	0	0	0	0	0	0	0	0	0	
C	0	-1	-1	-1	-1	2	1	0	-1	
T	0	-1	1	1	0	1	4	3	2	
G	0	2	1	0	0	0	3	6	5	
T	0	1	4	3	2	1	2	5	8	
A	0	0	3	3	5	4	3	4	7	
T	0	-1	2	5	4	4	6	5	6	
C	0	0	1	4	4	6	5	5	5	
(T)										

Armstrong, 2007

```
GTTACTGT---(S)
||||
----CTGTATC(T)
```

		G	T	T	A	C	T	G	T	(S)
	0	0	0	0	0	0	0	0	0	
C	0	-1	-1	-1	-1	2	1	0	-1	
T	0	-1	1	1	0	1	4	3	2	
G	0	2	1	0	0	0	3	6	5	
T	0	1	4	3	2	1	2	5	8	
A	0	0	3	3	5	4	3	4	7	
T	0	-1	2	5	4	4	6	5	6	
C	0	0	1	4	4	6	5	5	5	
(T)										

Armstrong, 2007

A bit more formally..

Base conditions: $\forall i,j.\ V(i,0) = 0,\ V(0,j) = 0$

Recurrence relation: for $1<=i<=n$, $1<=j<=m$:

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

Search for i^* such that: $V(i^*,m)=\max_{1<=i<=n,m} V(i,j)$

Search for j^* such that: $V(n,j^*)=\max_{1<=j<=n,m} V(i,j)$

Define alignment score $V(S,T) = \max \begin{cases} V(n,j^*) \\ V(i^*,m) \end{cases}$

Armstrong, 2007

Summary so far...

- Dynamic programming algorithms can solve global, local and ends-free alignment
- They give the optimum score and alignment using the parameters given
- Divide and conquer approaches make the space complexity manageable for small-medium sized sequences

Armstrong, 2007

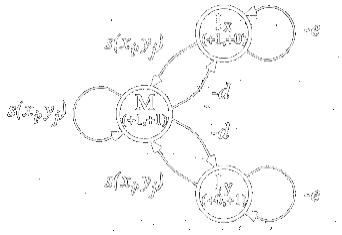
Dynamic Programming Issues

- For huge sequences, even linear space constraints are a problem.
- We used a very simple gap penalty
- The Affine Gap penalty is most commonly used.
  - Cost to open a gap
  - Cost to extend an open gap
- Need to track and evaluate the 'gap' state in the array

Armstrong, 2007

Tracking the gap state

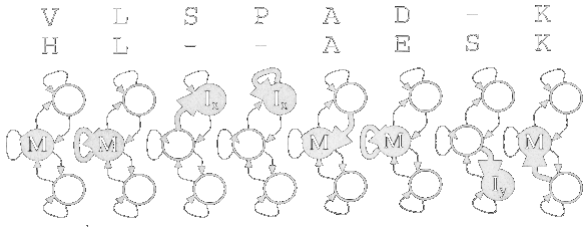
- We can model the matches and gap insertions as a finite state machine:



Taken from Durbin, chapter 2.4

Tracking the gap state

- Working along the alignment process...



Taken from Durbin, chapter 2.4

Real Life Sequence Alignment

- When searching multiple genomes, the sizes still get too big!
- Several approaches have been tried:
- Use huge parallel hardware:
  - Distribute the problem over many CPUs
  - Very expensive
- Implement in Hardware
  - Cost of specialist boards is high
  - Has been done for Smith-Waterman on SUN

Armstrong, 2007

Real Life Sequence Alignment

- Use a Heuristic Method
  - Faster than ‘exact’ algorithms
  - Give an approximate solution
  - Software based therefore cheap
- Based on a number of assumptions:

Armstrong, 2007

Assumptions for Heuristic Approaches

- Even linear time complexity is a problem for large genomes
- Databases can often be pre-processed to a degree
- Substitutions more likely than gaps
- Homologous sequences contain a lot of substitutions without gaps which can be used to help find start points in alignments

Armstrong, 2007

Conclusions

- Dynamic programming algorithms are expensive but they give you the optimum alignment and exact score
- Choice of GAP penalty and substitution matrix are critically important
- Heuristic approaches are generally required for high throughput or very large alignments

Armstrong, 2007

Heuristic Methods

- FASTA
- BLAST
- Gapped BLAST
- PSI-BLAST

Armstrong, 2007

Assumptions for Heuristic Approaches

- Even linear time complexity is a problem for large genomes
- Databases can often be pre-processed to a degree
- Substitutions more likely than gaps
- Homologous sequences contain a lot of substitutions without gaps which can be used to help find start points in alignments

Armstrong, 2007

FASTA

Lipman and Pearson (1988) Improved tools for biological sequence comparison. PNAS 85: 10915-10919

- Compares a query string against a single text string (i.e. for sequence databases, lots of searches)
- Based on the assumption that good local alignment is likely to have some exact matching subsequences
- The algorithm looks for these subsequences first.

Armstrong, 2007

Dot-plot alignment

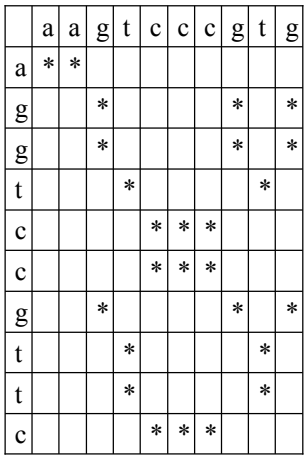
- We can find good subsequences just by looking for diagonal runs of matched bases:

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

Armstrong, 2007

Dot-plot alignment

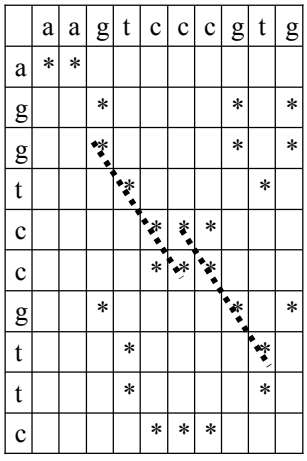
- We can find good subsequences just by looking for diagonal runs of matched bases:
- Mark identical hits



Armstrong, 2007

Dot-plot alignment

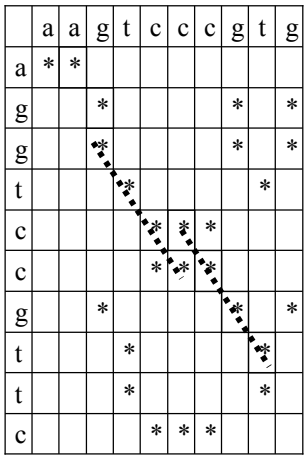
- We can find good subsequences just by looking for diagonal runs of matched bases:
- Find Diagonal Runs:



Armstrong, 2007

Dot-plot alignment

- We can find good subsequences just by looking for diagonal runs of matched bases:
- Compare to DP alignment:



Armstrong, 2007

FASTA Definitions

- *ktup:*
  - (k respective tuples) – an integer value which specifies the word length used to find matching substrings
  - Standard 4-6 for DNA
  - Standard 1 or 2 for proteins
  - Shorter is more sensitive but slower
  - Target databases can be preprocessed into ktup sized chunks before queries are run.

Armstrong, 2007

FASTA Definitions

- *hot spots:*
  - The matching *ktup* length substrings
  - Consecutive *hot-spots* are located along the diagonal
  - See dot-plot for example of 4 length hotspots
  - Often close to the dynamic programming solution
- *diagonal run*:
  - A sequence of nearby *hot-spots* on the same diagonal
  - i.e. spaces between *hot-spots* are allowed

Armstrong, 2007

FASTA Definitions

- $init_1$*:*
  - The best scoring run
- $init_n$:
  - The best local alignment
  - Combination of good diagonal runs and indels/gaps between them.

Armstrong, 2007

FASTA Process

1. Look for *hot-spots*:

- The stage can be done by using a look-up table or a hash.
- Pre-process the database and store the location of each possible *ktup* (AA=20^2, DNA=4^6)
- Move a *ktup* sized window along the query sequence and record the position of matching locations in the database.

Armstrong, 2007

FASTA Process

2. Find best *diagonal runs*:

- Each *hot spot* gets a positive score.
- Distance between *hot spots* is negative and length dependant
- Score of the diagonal run
- Fasta finds and stores the 10 best diagonal runs

Armstrong, 2007

FASTA Process

3. Compute $init_1$ & filter:

- Diagonal runs specify a potential alignment
- Evaluate properly using a substitution matrix
- Define the best scoring run as $init_1$
- Discard any much lower scoring runs

Armstrong, 2007

FASTA Process

4. Combine diagonal runs and compute $init_n$:

- Take the 'good alignments' from previous stage
- Now allow gaps/indels
- Combine them into a single, better scoring alignment
  - Construct a directed weighted graph
    - vertices are the runs
    - edge weights represent gap penalties
  - Find the best path through the graph = $init_n$

Armstrong, 2007

FASTA Process

5. Find the best local alignment

- Use the 'alignments' from the previous stage to define a narrow band through the search space
- Go through that band using a dynamic programming approach
- Size of the band is dependant on *ktup* value
- The best local alignment found in this stage is called *opt*

Armstrong, 2007

FASTA Process

6. Compare the alignments

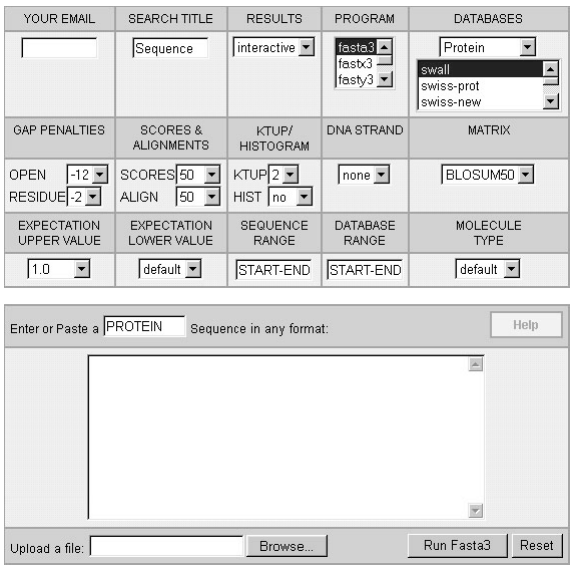
- Take the *opt* or $init_n$ scores for each sequence in the database
- Rank according to score
- Use a full dynamic programming algorithm to align the query sequence with the highest ranking result sequences

Armstrong, 2007

FASTA Programs

- fasta3 scan a protein or DNA sequence library for similar sequences
- fastax/y3 compare a DNA sequence to a protein sequence database, comparing the translated DNA sequence in forward and reverse frames
- tfastax/y3 compares a protein to a translated DNA data bank
- fasts3 compares linked peptides to a protein databank
- fastf3 compares mixed peptides to a protein databank

Armstrong, 2007



Armstrong, 2007

FASTA Summary

- The alignment produced is not always optimal
- The resulting scores usually compare very well with the dynamic programming solutions
- FASTA is much faster than ordinary dynamic programming algorithms

BLAST

Altschul, Gish, Miller, Myers and Lipman (1990) Basic local alignment search tool. J Mol Biol 215:403-410

- Developed on the ideas of FASTA
- Integrates the substitution matrix in the first stage of finding the *hot spots*
- Faster *hot spot* finding

BLAST definitions

- Given two strings S_1 and S_2
- A *segment pair* is a pair of equal lengths substrings of S_1 and S_2 aligned without gaps
- A *locally maximal segment* is a segment whose alignment score (without gaps) cannot be improved by extending or shortening it.
- A *maximum segment pair* (*MSP*) in S_1 and S_2 is a segment pair with the maximum score over all segment pairs.

Armstrong, 2007

BLAST Process

- Parameters:
  - *w*: word length (substrings)
  - *t*: threshold for selecting interesting alignment scores

Armstrong, 2007

BLAST Process

- 1. Find all the *w*-length substrings from the database with an alignment score >*t*
  - Each of these (similar to a hot spot in FASTA) is called a *hit*
  - Does not have to be identical
  - Scored using substitution matrix and score compared to the threshold *t* (which determines number found)
  - Words size can therefore be longer without losing sensitivity: AA - 3-7 and DNA ~12

Armstrong, 2007

BLAST Process

- 2. Extend hits:
  - extend each hit to a local maximal segment
  - extension of initial w size hit may increase or decrease the score
  - terminate extension when a threshold is exceeded
  - find the best ones (HSP)
- This first version of Blast did not allow gaps….

Armstrong, 2007

(Improved) BLAST

Altshul, Madden, Schaffer, Zhang, Zhang, Miller & Lipman (1997) Gapped BLAST and PSI-BLAST:a new generation of protein database search programs. Nucleic Acids Research 25:3389-3402

- Improved algorithms allowing gaps
  - these have superceded the older version of BLAST
  - two versions: Gapped and PSI BLAST

(Improved) BLAST Process

- Find words or hot-spots
  - search each diagonal for two w length words such that score >=*t*
  - future expansion is restricted to just these initial words
  - we reduce the threshold t to allow more initial words to progress to the next stage

(Improved) BLAST Process

- Allow local alignments with gaps
  - allow the words to merge by introducing gaps
  - each new alignment comprises two words with a number of gaps
  - unlike FASTA does not restrict the search to a narrow band
  - as only two word hits are expanded this makes the new blast about 3x faster

Armstrong, 2007

PSI-BLAST

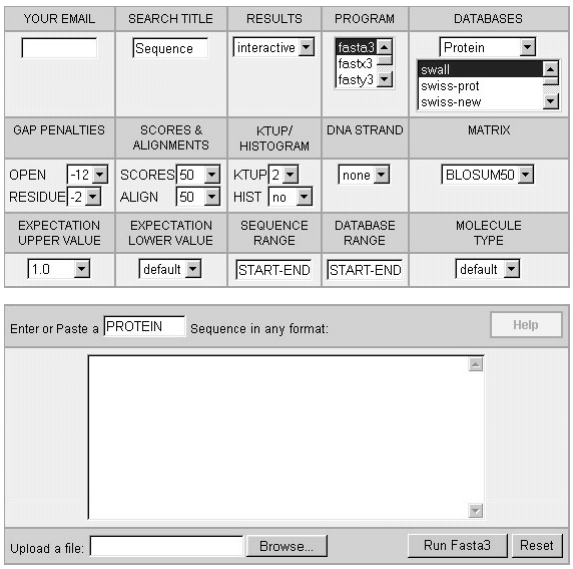
- Iterative version of BLAST for searching for protein domains
  - Uses a dynamic substitution matrix
  - Start with a normal blast
  - Take the results and use these to 'tweak' the matrix
  - Re-run the blast search until no new matches occur
- Good for finding distantly related sequences but high frequency of false-positive hits

Armstrong, 2007

BLAST Programs

- blastp — compares an amino acid query sequence against a protein sequence database.
- blastn — compares a nucleotide query sequence against a nucleotide sequence database.
- blastx — compares a nucleotide query sequence translated in all reading frames against a protein sequence database.
- tblastn — compares a protein query sequence against a nucleotide sequence database dynamically translated in all reading frames.
- tblastx — compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database. (SLOW)

Armstrong, 2007



Armstrong, 2007

Go try them out!

- Links to NCBI and EBI are on the course web site
- Some test sequences will be posted on the course web site

Armstrong, 2007

Alignment Heuristics

- Dynamic Programming is better but too slow
- FASTA and BLAST based on several assumptions about good alignments
  - substitutions more likely than gaps
  - good alignments have runs of identical matches
- FASTA good for DNA sequences but slower
- BLAST better for amino acid sequences and pretty good for DNA, fastest.

Armstrong, 2007