Bio2

Pair-wise Sequence Alignment

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?

Sequence Alignment Intro

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

- Way of comparing two sequences and assessing the similarity or difference between them
- Can align DNA or Protein sequences
- Matches/substitutions scored from a look-up matrix
- Insertion/deletions scored by some gap-penalty formula

BLOSUM 62 Matrix

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

How do we do it?

- A Dynamic Programming algorithm is used to find the optimal scored alignment (and non-optimal scores)
  - MPSearch
- Heuristic approaches improve speed but sacrifice some accuracy
  - BLAST
  - FASTA

Alignment Types

- Global: used to compare to similar sized sequences.

- Local: used to find similar subsequences.

- Ends Free: used to find joins/overlaps.

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

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

Needleman-Wunsch algorithm

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

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

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 : $\sigma(S_i,-) = -1$

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

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: $\sigma(S_i,T_j)$

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
```

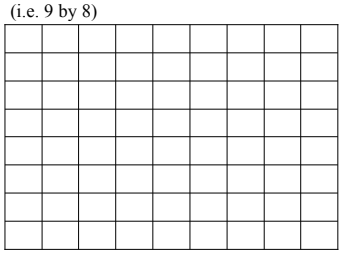
Length of $S = m = 8$

Length of T = $n = 7$

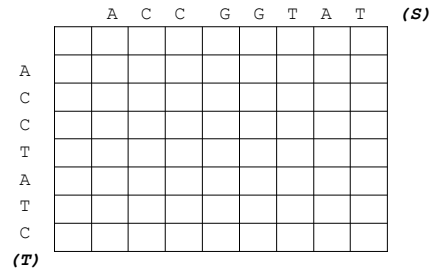
(lengths approx equal so OK for Global Alignment)

Create array m+1 by n+1

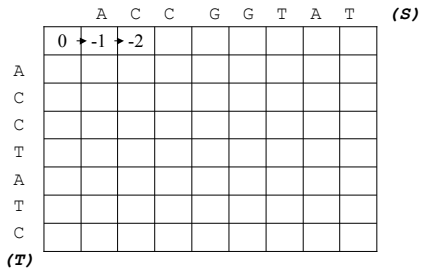
(i.e. 9 by 8)



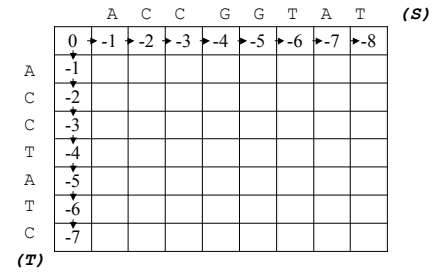
Add on bases from each sequence



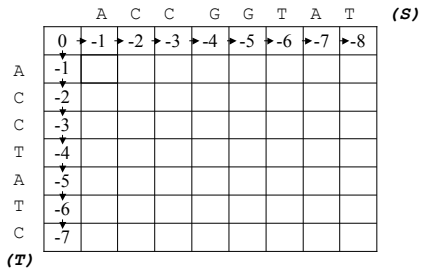
Represent scores for gaps in row/col 0



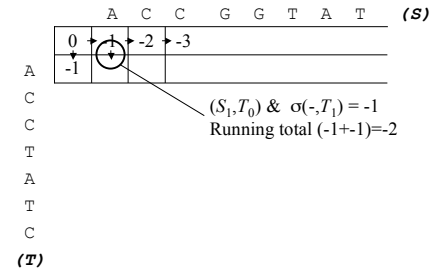
Represent scores for gaps in row/col 0



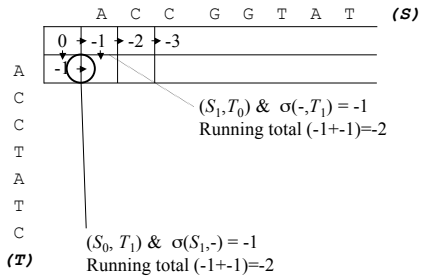
For each cell consider the 'best' path



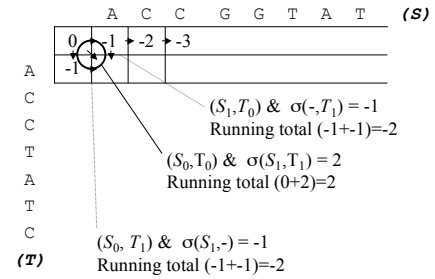
For each cell consider the 'best' path



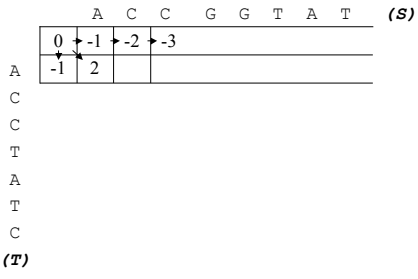
For each cell consider the 'best' path



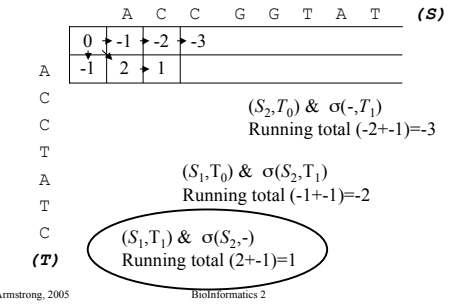
For each cell consider the 'best' path



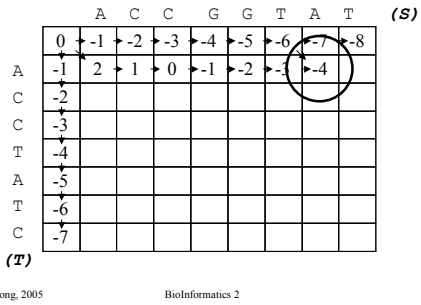
Choose and record 'best' path



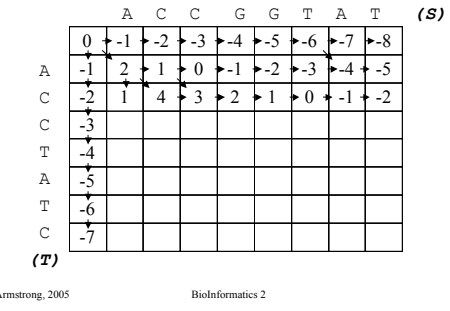
Choose and record 'best' path



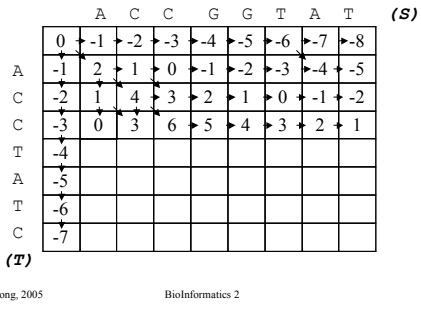
Continue….



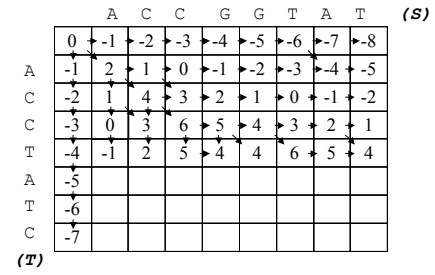
Continue….



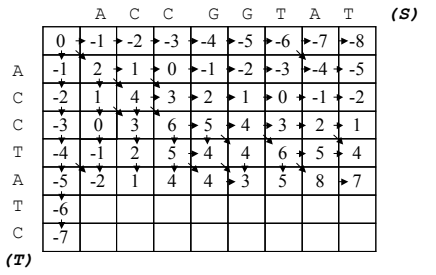
Continue….



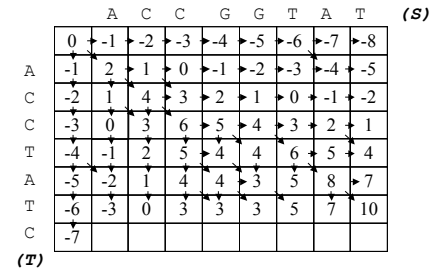
Continue….



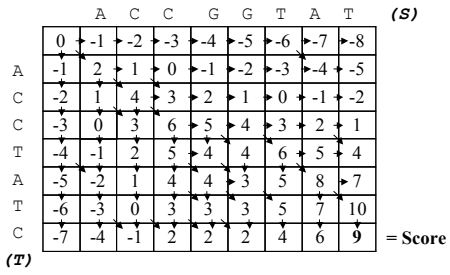
Continue….



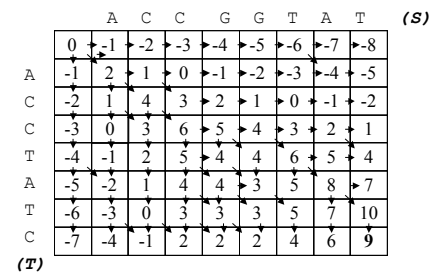
Continue….



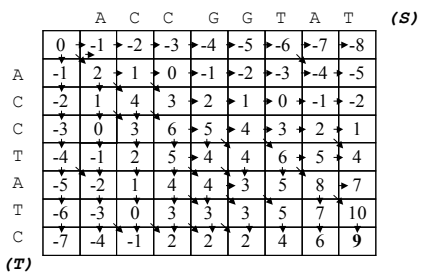
Finally.

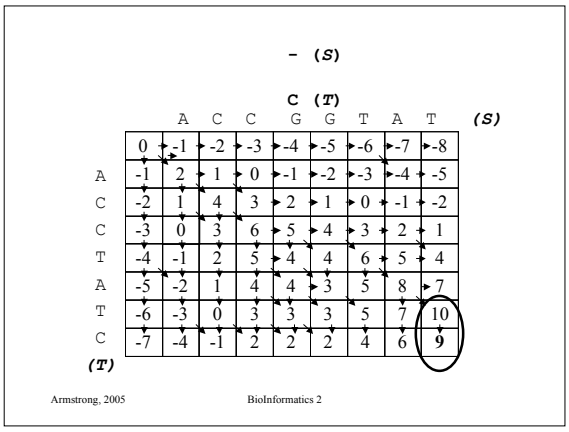


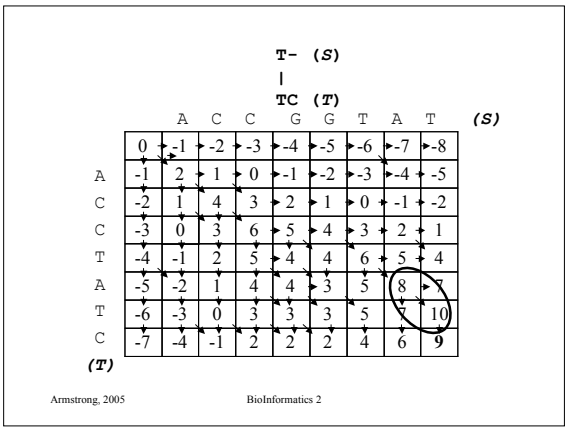
Finally.

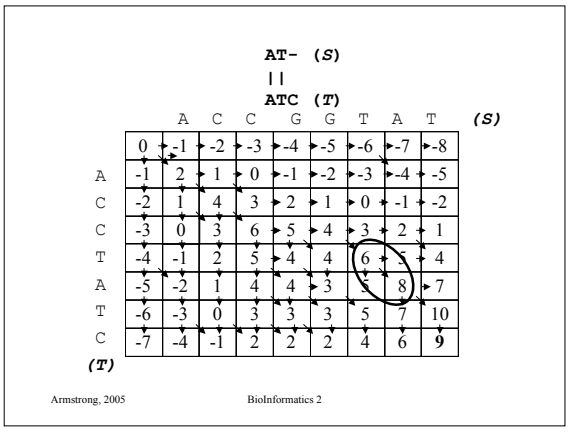


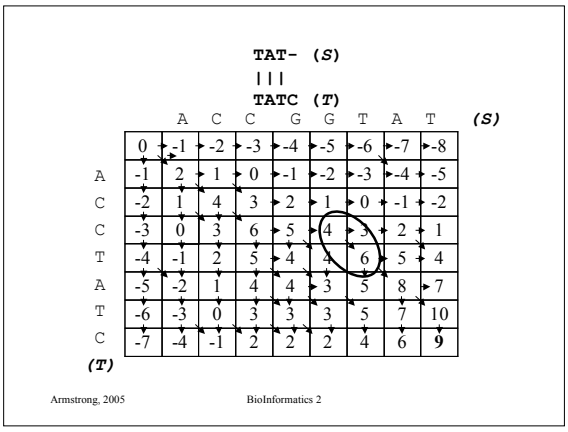
We recreate the alignment using by following the pointers back through the array to the origin

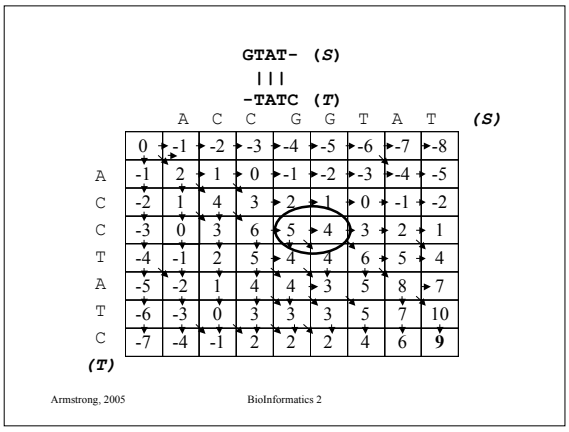


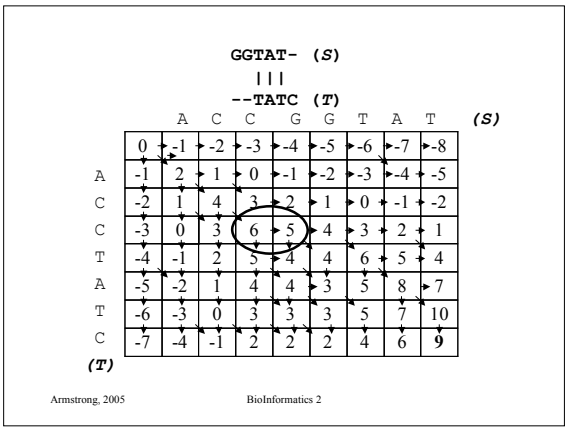

- (S)
C (T)
A C C G G T A T (S)
0 -1 -2 -3 -4 -5 -6 -7 -8
A -1 2 1 0 -1 -2 -3 -4 -5
C -2 1 4 3 2 1 0 -1 -2
C -3 0 3 6 5 4 3 2 1
T -4 -1 2 5 4 4 6 5 4
A -5 -2 1 4 4 3 5 8 7
T -6 -3 0 3 3 3 5 7 10
C -7 -4 -1 2 2 2 4 6 9
(T)


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)


AT- (S)
||
ATC (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)


TAT- (S)
|||
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)


GTAT- (S)
|||
-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)


GGTAT- (S)
|||
--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)

```
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)										


```
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)										


```
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)										

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.

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 $1<=i<=n$, $1<=j<=m$:

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

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 $O(nm)$

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

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

dividing...

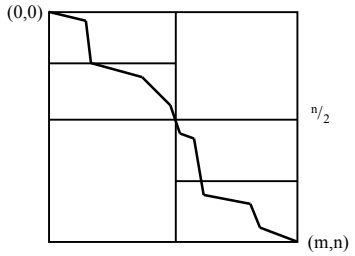
Compute matrix V(A,B) saving the values for $^{n}/_{2}^{th}$ row
- call this matrix F

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*)

Hirschberg's divide and conquer approach



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.

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...

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)										

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)										

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)										

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)										

GTAT (*S*)
||||
GTAT (*T*)

		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)										

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.

A bit more formally..

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

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

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

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

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

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

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)										

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)										

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)										


```
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)										

A bit more formally..

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

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

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

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

Summary so far...

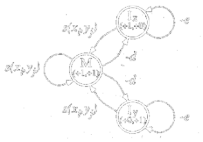
- Dynamic programming algorithms can solve global, local and ends-free alignment
- They give the optimum score and alignment using the parameters given
- Divide and conquer approaches make the space complexity manageable for small-medium sized sequences

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

Tracking the gap state

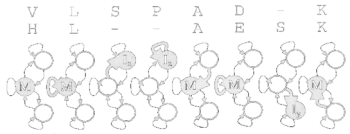
- We can model the matches and gap insertions as a finite state machine:



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

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:

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

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