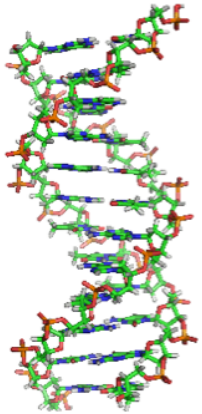
Lecture 2: Sequence Alignment

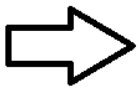
Rob Powell

robert.powell@dur.ac.uk

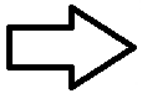
Tree Construction

1. Extract DNA sequence from each species.
2. Align the sequences.
3. Compute the inter-species distances.
4. Build the tree from the distance matrix.





```
-------GGC-CGTGTTCTTGCGGTGGCTG--AAGATGGTCC
       || |||  ||||| | ||||  |   ||||||||||
CGCATGCGGAACGT--TCTTTCCGTGG--GAACAGATGGTCC
|  |||||| |||   ||||  | ||||  ||| ||||||||||
C--ATGCGGCACG---TCTTGCGGTGG--GAAAAGATGGTCC
```



Alignments

Sequence 1: ACGGT
Sequence 2: ACCGT

- What possible alignments are there?
- Each position of the alignment is either:
  - A character from each sequence.
  - A character from sequence 1 against a gap.
  - A character from sequence 2 against a gap.

Alignments

First we must bound the size of the search space:

- We have two sequences of length n and m.
- Each position has three possible states
- There are at most n+m positions.
- Hence there are $3^{(n+m)}$ possible alignments.

Alignments

Let's go back to our earlier example.

Sequence 1: ACGGT
Sequence 2: ACCGT

We can think of alignment as a recursive process:

Alignments

Let's go back to our earlier example.

Sequence 1: ACGGT
Sequence 2: ACCGT

We can think of alignment as a recursive process:

- End with $\begin{matrix}\text{T}\\\text{T}\end{matrix}$ preceded by all alignments of $\begin{matrix}\text{ACGG}\\\text{ACCG}\end{matrix}$

Alignments

Let's go back to our earlier example.

Sequence 1: ACGGT
Sequence 2: ACCGT

We can think of alignment as a recursive process:

- End with $\begin{matrix}\text{T}\\\text{T}\end{matrix}$ preceded by all alignments of $\begin{matrix}\text{ACGG}\\\text{ACCG}\end{matrix}$
- or end with $\begin{matrix}\text{T}\\\text{–}\end{matrix}$ preceded by all alignments of $\begin{matrix}\text{ACGG}\\\text{ACCGT}\end{matrix}$

Alignments

Let's go back to our earlier example.

Sequence 1: ACGGT
Sequence 2: ACCGT

We can think of alignment as a recursive process:

- End with $\begin{matrix}\mathrm{T}\\\mathrm{T}\end{matrix}$ preceded by all alignments of $\begin{matrix}\mathrm{ACGG}\\\mathrm{ACCG}\end{matrix}$
- or end with $\begin{matrix}\mathrm{T}\\\mathrm{-}\end{matrix}$ preceded by all alignments of $\begin{matrix}\mathrm{ACGG}\\\mathrm{ACCGT}\end{matrix}$
- or end with $\begin{matrix}\mathrm{-}\\\mathrm{T}\end{matrix}$ preceded by all alignments of $\begin{matrix}\mathrm{ACGGT}\\\mathrm{ACCG}\end{matrix}$

Alignments

Let's take a really simply example now.

Sequence 1: AG
Sequence 2: AC

There are three possible assignments of the last elements of these sequences:

Alignments

Let's take a really simply example now.

Sequence 1: AG
Sequence 2: AC

There are three possible assignments of the last elements of these sequences:

- First option is to end with $\begin{matrix} G \\ C \end{matrix}$

Alignments

Let's take a really simply example now.

Sequence 1: AG
Sequence 2: AC

There are three possible assignments of the last elements of these sequences:

- First option is to end with $\begin{matrix} G \\ C \end{matrix}$

  AG
  AC

Alignments

Let's take a really simply example now.

Sequence 1: AG
Sequence 2: AC

There are three possible assignments of the last elements of these sequences:

- First option is to end with $\begin{matrix} G \\ C \end{matrix}$

  AG –AG
  AC A–C

Alignments

Let's take a really simply example now.

Sequence 1: AG
Sequence 2: AC

There are three possible assignments of the last elements of these sequences:

- First option is to end with $\begin{matrix} G \\ C \end{matrix}$

AG	–AG	A–G
AC	A–C	–AC

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

  –AG
  AC–

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG ––AG
AC– AC––

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G
AC–	AC––	AC–

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G
AC–	AC––	AC–	A–C–

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

AG–
–AC

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

AG–	AG––
–AC	––AC

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

AG–	AG––	AG–
–AC	––AC	A–C

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

AG–	AG––	AG–	–A G–
–AC	––AC	A–C	A––C

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

AG–	AG––	AG–	–A G–	A–G–
–AC	––AC	A–C	A––C	–A–C

Alignments

- Second option is to end with $\begin{matrix} G \\ - \end{matrix}$

–AG	––AG	A–G	–A–G	A––G
AC–	AC––	AC–	A–C–	–A C–

- Final option is to end with $\begin{matrix} - \\ C \end{matrix}$

AG–	AG––	AG–	–A G–	A–G–
–AC	––AC	A–C	A––C	–A–C

Note that we've only generated 13 possible alignments, yet our bound was $3^{(2+2)} = 81$ possible assignments. Our original bound is nowhere near tight!

A Better Bound

- Let $al(n, m)$ be the number of alignments of two sequences of length n and m.
- Using our idea of recursion we can see that $al(n, m) = al(n-1, m-1) + al(n-1, m) + al(n, m-1)$
- It is clear that $al(i, 0) = 1$ and $al(0, j) = 1$

A Better Bound

- Let $al(n, m)$ be the number of alignments of two sequences of length n and m.
- Using our idea of recursion we can see that
  $al(n, m) = al(n-1, m-1) + al(n-1, m) + al(n, m-1)$
- It is clear that $al(i, 0) = 1$ and $al(0, j) = 1$
- Now we can calculate some simple examples:
  - $al(1, 1) = al(0, 0) + al(0, 1) + al(1, 0) = 3$
  - $al(2, 2) = al(1, 1) + al(1, 2) + al(2, 1) = 13$
  - and so on....

Programming

1. Read sequence data from a file
2. Generate the list of all possible alignments
3. Score each of the alignments, keeping a track of the best
4. Output the best alignment

An Example

input:

Sequence 1 = GATCGGATC
Sequence 2 = GGTCGGGCT

output:

Sequence 1 has length 9.
Sequence 2 has length 9.
Alignments generated: 1,462,563

Best:
Score: 6
Time: 11.7s

Generating Alignments

Sequence Length	# of Alignments	Time to Compute (secs)
0	1	0.000
1	3	0.000
2	13	0.000
3	63	0.000
4	321	0.001
5	1,683	0.008
6	8,989	0.048
7	48,639	0.309
8	265,729	1.858
9	1,462,563	11.783
10	8,097,453	75.735

Roughly 125,000 alignments checked per second.

Generating Alignments

As you can see from the previous slide, the number of alignments, and the time taken to compute them all, grows rapidly with the sequence length.

For two sequences of length 10 there are over 8,000,000 possible alignments.

Generating Alignments

As you can see from the previous slide, the number of alignments, and the time taken to compute them all, grows rapidly with the sequence length.

For two sequences of length 10 there are over 8,000,000 possible alignments.

For two sequences of length 100 there are over 2,053,716,830,872,420,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possible alignments!

Checking them all is computationally infeasible - so we need a better approach!

Dynamic Programming

Lets think back to our original example.

Sequence 1: ACGGT
Sequence 2: ACCGT

How do we obtain the best possible score?

The score depends on each character of the alignment individually, not on the global alignment.

Dynamic Programming

Sequence 1: ACGGT
Sequence 2: ACCGT

$$\begin{matrix} T & T & - \\ T & - & T \end{matrix}$$ are the three choices in the last position.

Dynamic Programming

Sequence 1: ACGGT
Sequence 2: ACCGT

$$\begin{matrix} \mathsf{T} & \mathsf{T} & - \\ \mathsf{T} & - & \mathsf{T} \end{matrix}$$ are the three choices in the last position.

$$al(ACGGT, ACCGT) = al(ACGG, ACCG) + al(ACGGT, ACCG) + al(ACGG, ACCGT)$$

Dynamic Programming

Sequence 1: ACGGT
Sequence 2: ACCGT

$$\begin{matrix} T & T & - \\ T & - & T \end{matrix}$$ are the three choices in the last position.

$$al(ACGGT, ACCGT) = al(ACGG, ACCG) + al(ACGGT, ACCG) + al(ACGG, ACCGT)$$

$$score(ACGGT, ACCGT) = max\{score(ACGG, ACCG) + 2, score(ACGGT, ACCG) - 2, score(ACGG, ACCGT) - 2\}$$

where +2 is the score of a T and T match, and -2 is the penalty for a gap.

Dynamic Programming

Let Seq1[:i] be the first i characters of Sequence 1.
Let Seq2[:j] be the first j characters of Sequence 2.

We want to compute score(Seq1[:i],Seq2[:j]). Either

- Seq1[i] is aligned with Seq2[j], or
- Seq1[i] is aligned with "–", or
- "–" is aligned with Seq2[j].

Let c(i,j) be +2 if Seq1[i] matches Seq2[j], and -1 otherwise.
If we align either Seq1[i] or Seq2[j] with "–" then we incur a penalty of -2.

Dynamic Programming

For convenience we will abbreviate score(Seq1[:i],Seq2[:j]) to simply s(i,j).

$$s(i,j) = max\{c(i,j) + s(i-1,j-1), s(i-1,j) - 2, s(i,j-1) - 2\}$$

We know that $s(i,0) = -2i$ and $s(0,j) = -2j$ as we are just aligning characters against gaps.

Following through the recursive function calls we eventually find the best alignment score.

Dynamic Programming

Seq1 = AC
Seq2 = AG

		A	G
	0	-2	-4
A	-2		
C	-4		

s(0,1): Best score for aligning a blank against A

s(0,2): Best score for aligning blanks against AG

s(1,0): Best score for aligning a blank against A

s(2,0): Best score for aligning blanks against AC

Dynamic Programming

Seq1 = AC
Seq2 = AG

		A	G
	0	-2	-4
A	-2	2	0
C	-4	0	1

$$s(1,1) = \max\{c(i,j) + s(0,0), s(0,1)-2, s(1,0)-2\}$$
$$= \max\{2+0, -2-2, -2-2\}$$
$$= 2$$

$$s(1,2) = \max\{c(i,j) + s(0,1), s(0,2)-2, s(1,1)-2\}$$
$$= \max\{-1-2, -4-2, 2-2\}$$
$$= 0$$

$$s(2,1) = \max\{c(i,j) + s(1,0), s(1,1)-2, s(2,0)-2\}$$
$$= \max\{-1-2, 2-2, -4-2\}$$
$$= 0$$

$$s(2,2) = \max\{c(i,j) + s(1,1), s(1,2)-2, s(2,1)-2\}$$
$$= \max\{-1+2, 0-2, 0-2\}$$
$$= 1$$

Dynamic Programming

- Optimal substructure - A problem is said to have optimal substructure if the optimal solution can be found by breaking the problem into subproblems, and recursively finding the solution to the subproblems.
- Overlapping subproblems - If the recursion creates the same subproblem multiple times, then we only need to solve it once, rather than multiple times. This helps bound the number of subcases to be solved.
- Dynamic Programming can be applied to problems that have both optimal substructure and overlapping subproblems.

Problems

Seq1 = TTCCAATCTCGAGATCGGATC
Seq2 = GATCGGATCTTGGCCTAGCTA

```
TTCCAATCTCGAGATCG–GATC––––––––––––––
                |||| |  |||| |
––––––––––––––GATC–GGATCTTGGCCTAGCTA
```

- This alignment would score very badly under our standard scoring system.
- However this is a realistic type of match to search for.
- Can we adjust our algorithm to give a better score to alignments such as this?

Ends-free scoring

```
TTCCAATCTCGAGATCG–GATC––––––––––––––
                |||| |   |||| |
––––––––––––––GATC–GGATCTTGGCCTAGCTA
```

- We can change our scoring matrix to make gaps at the beginning or end of sequences free.
- Initialise the top row, and leftmost column to 0. This is equivalent to letting $s(0,i) = 0$ and $s(j,0) = 0$.
- In the bottom row and rightmost column we also allow gaps with 0 cost.

$$s(m,j) = max\{c(m,j)+s(m-1,j-1), s(m,j-1), s(m-1,j)-2\}$$

$$s(i,n) = max\{c(i,n) + s(i-1,n-1), s(i,n-1) - 2, s(i-1,n)\}$$

Local Alignment

Seq1 = TTCCAATCTCGAGATCGGATCTGACTAATCGA
Seq2 = AAGCCTTAGGCGGATGATCGGATCTTGGCCTAGCTA

```
      TTCCAATCTCGAGATCGGATCTGACTAATCGA
                  | | | | | | | | | |
AAGCCTTAGGCGGATGATCGGATCTTGGCCTAGCTA
```

- Perhaps a match such as this is some important protein that is encoded in two otherwise different genes.
- How do we adjust our algorithm to find such local alignments?

Local Alignment

TTCCAATCTCGAGATCGGATCTGACTAATCGA
| | | | | | | | | | |
AAGCCTTAGGCGGATGATCGGATCTTGGCCTAGCTA

- In the top row and left column set all scores to 0.
- Don't allow negative entries:

$$s(i,j) = max\{c(i,j)+s(i-1,j-1), s(i-1,j)-2, s(i,j-1)-2, 0\}$$

- Look for highest score in the matrix, instead of the value in the bottom right.

Complexity

- We can align two sequences of length n and m in O(nm) steps.
- For k sequences of length n, the score matrix is k-dimensional.
- Dynamic Programming approach takes O(n^k) steps.
- This is fine for small examples, but trying to align thousands of sequences would be very slow.
- For this we must use heuristic algorithms (not guarenteed to be optimal).

BLAST

- BLAST - Basic Local Alignment Search Tool.
- The "standard" algorithm for sequence alignment.
- Takes sub-sequences of 11 characters from the query sequence and looks for close matches in a database of target sequences.
- Close matches are then extended to high scoring segments and finally to local alignments.