Pairwise alignment of biological sequences
Part 2

Peter Sestoft
sestoft@dina.kvl.dk
Department of Natural Sciences, KVL

2004-09-10

Plan:

	Global	Local
Linear	**1:** Linear global (Needleman-Wunsch)	**2:** Linear local (Smith-Waterman)
Affine	**3:** Affine global (Needleman-Wunsch)	**4:** Affine local (Smith-Waterman)

We're are really interested in **4** — this is what Blast does.

But it is easier to explain **1** and the changes in direction **2** and in direction **3** independently.

3. Affine gap costs

Until now we used *linear* gap costs $g(h) = -d \cdot k$, proportional to the length h of the gap (and d was 8).

Thus a gap of length $h = 4$ has 4 times the cost of a gap of length 1.

This is unrealistic; too expensive, biologically speaking.

A gap arises by an evolutionary event, and a long gap is nearly as likely to arise as a short one.

Better use *affine gap costs* of the form $g(h) = -d - e \cdot (h - 1)$ where $d = 10$ and $e = 1$, for instance.

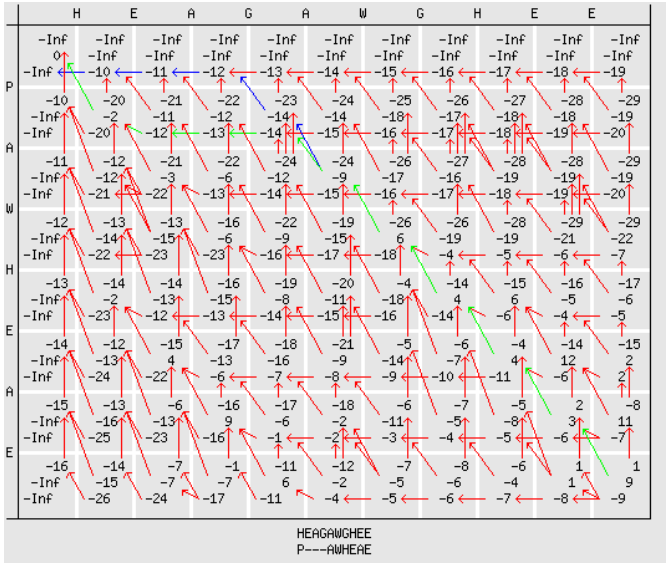
Hence it is expensive to open a gap (-10) but inexpensive to extend it (-1).

Alignment with affine gap costs is done by dynamic programming using three matrices I_y, M, I_x instead of one.

The matrices have the following meanings:

$$
\begin{aligned}
I_y(i,j) &= \text{max score for alignment between } x_{1\ldots i} \text{ and } y_{1\ldots j} \text{ ending with a match between a gap in } x \text{ and } y_j \\
M(i,j) &= \text{max score for alignment between } x_{1\ldots i} \text{ and } y_{1\ldots j} \text{ ending with a match between } x_i \text{ and } y_j \\
I_x(i,j) &= \text{max score for alignment between } x_{1\ldots i} \text{ and } y_{1\ldots j} \text{ ending with a match between } x_i \text{ and a gap in } y
\end{aligned}
$$

Global alignment, affine gap costs



Global alignment, affine gap costs

```
for (int i=1; i<=n; i++)
  for (int j=1; j<=m; j++) {
    int val;
    int s = score[x[i]][y[j]];
    val = Iy[i][j] = max(M[i][j-1]-d, Iy[i][j-1]-e, Ix[i][j-1]-d);
    if (val == M[i][j-1]-d)
      B[0][i][j] = new Traceback3(1, i, j-1);        // Up, to M
    else if (val == Iy[i][j-1]-e)
      B[0][i][j] = new Traceback3(0, i, j-1);        // Up, to Iy
    else if (val == Ix[i][j-1]-d)
      B[0][i][j] = new Traceback3(2, i, j-1);        // Up, to Ix
    val = M[i][j] = max(M[i-1][j-1]+s, Ix[i-1][j-1]+s, Iy[i-1][j-1]+s);
    if (val == M[i-1][j-1]+s)
      B[1][i][j] = new Traceback3(1, i-1, j-1);      // Up left, to M
    else if (val == Ix[i-1][j-1]+s)
      B[1][i][j] = new Traceback3(2, i-1, j-1);      // Up left, to Ix
    else if (val == Iy[i-1][j-1]+s)
      B[1][i][j] = new Traceback3(0, i-1, j-1);      // Up left, to Iy
    val = Ix[i][j] = max(M[i-1][j]-d, Ix[i-1][j]-e, Iy[i-1][j]-d);
    if (val == M[i-1][j]-d)
      B[2][i][j] = new Traceback3(1, i-1, j);        // Left, to M
    else if (val == Ix[i-1][j]-e)
      B[2][i][j] = new Traceback3(2, i-1, j);        // Left, to Ix
    else if (val == Iy[i-1][j]-d)
      B[2][i][j] = new Traceback3(0, i-1, j);        // Left, to Iy
  }
}
```

Matrix number 0 is I_y, matrix number 1 is M, and matrix number 2 is I_x.

Overlap matches between $x =$ HEAGAWGHEE and $y =$ PAWHEAE

A prefix or suffix of x must be aligned with a prefix or suffix of y:

```
GAWGHEE
PAW-HEA
```

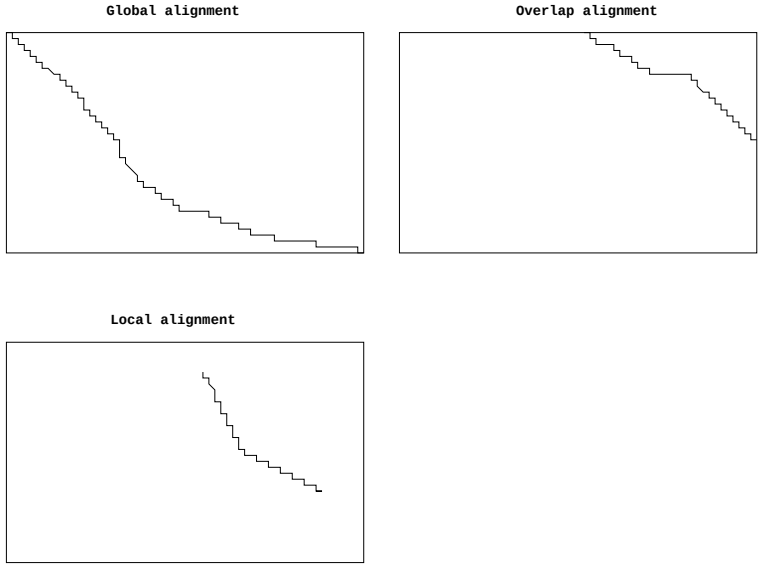
This is like local alignment, with the restrictions that

- an alignment must begin on the left-hand or top border;
- an alignment must end on the right-hand or bottom border.

That is, an alignment cannot begin or end inside the F matrix.

Overlap matches are used when assembling (nucleotide) sequences from shot-gun sequencing.

Bird's eye view of the traceback



Substitution matrices

Where do the numbers $score[a][b]$ in the substitution matrix come from?

Let q_a be the frequency of amino acid a.

Let p_{ab} be the probability of amino acids a and b being derived (mutated) from some common ancestor.

We must have $\sum_a q_a = 1$ and $\sum_b p_{ab} = 1$ for every a.

Then the scoring numbers are logarithms of ratios between such probabilities:

$$score[a][b] = \log(\frac{p_{ab}}{q_a q_b})$$

The absolute values of these numbers are not important, so one can scale and round them to obtain integers.

This is useful because as computing with integers is somewhat faster than with floating-point numbers.

But how do we find the probabilities q_a and p_{ab} in the first place?

The main substitution matrix families

- The PAM matrices (Dayhoff et al. 1978) are created from manual alignments of sequences with $\geq$85% identity, and thus evolutionarily close. The matrix PAM1 represents 1 % accepted mutations, and PAM250 = PAM1^{250} etc. are derived by extrapolation. Higher numbers represent higher evolutionary distance.
- The BLOSUM matrices (Henikoff and Henikoff 1992) are created from BLOCKS, manual multiple alignments of evolutionarily more distant sequences: BLOSUM62 is built from such multiple alignments with $\geq$62 % identity, and BLOSUM45 from multiple alignments with $\geq$45 % identity. Thus lower numbers represent higher evolutionary distance. The BLOSUM matrices perform better than PAM when aligning distant sequences.
- The GONNET matrices (Gonnet et al. 1992) are created like the PAM matrices, but from a broader material, and are better for computing evolutionary distance. The GONNET matrices are used by default in the ClustalW multiple alignment program (since version 1.8).

According to the Blast documentation,

> [...] the BLOSUM62 matrix is among the best for detecting most weak protein similarities. For particularly long and weak alignments, the BLOSUM45 matrix may prove superior. [...] The BLOSUM series does not include any matrices [...] suitable for the shortest queries, so the older PAM matrices may be used instead.

Blast version 2 uses BLOSUM62 with gap opening cost $d = -11$ and gap extension cost $e = -1$ by default.

The choice of gap penalties seems to be based on performance ('it works'), not probability theory.

Dot plots

A dot plot gives a rough comparison of two sequences.

It is a rectangular schema with a marking in element (i, j) if $x_i = y_j$.

$y \backslash x$	H	E	A	G	A	W	G	H	E	E
P										
A			•		•					
W						•				
H	•							•		
E		•							•	•
A			•		•					
E		•							•	•

Use the program at `http://www.isrec.isb-sib.ch/java/dotlet/Dotlet.html` to experiment with dot plots.

Database searches

A sequence database contains a large number of sequences (e.g. 100,000).

When searching a database we are given a short *query string*, e.g. of length $n = 500$.

We then seek the best local, gapped alignment between the query string and each of the database sequences.

So we might use the Smith-Waterman algorithm for every single sequence in the database.

But this is too slow in practice.

Database search programs (Blast and Fasta) do use dynamic programming, but only after some preliminary work.

Observation about the F matrix in dynamic programming

- When the strings x and y are *identical*, the traceback will follow a diagonal of the F matrix.
  In that case, only the diagonal needs to be filled in (which takes much less time).
- When the strings x and y are *very similar* (few gaps), the traceback will follow a band along a diagonal.
  In that case, only the elements of that band of F need to be filled in.
- We may speed up dynamic programming by filling in only a *fixed-width* band along a diagonal of F.
  We then consider all elements outside the band to have an infinitely low score, $-\infty$.
  But this may overlook a good (high-scoring) alignment whose traceback would have gone outside the band.
  This was the approach taken by Blast version 1 (old Blast) and Fasta. Blast version 2 (1997) is more subtle.

The Blast version 2 database search algorithm (Altschul et al. 1997)

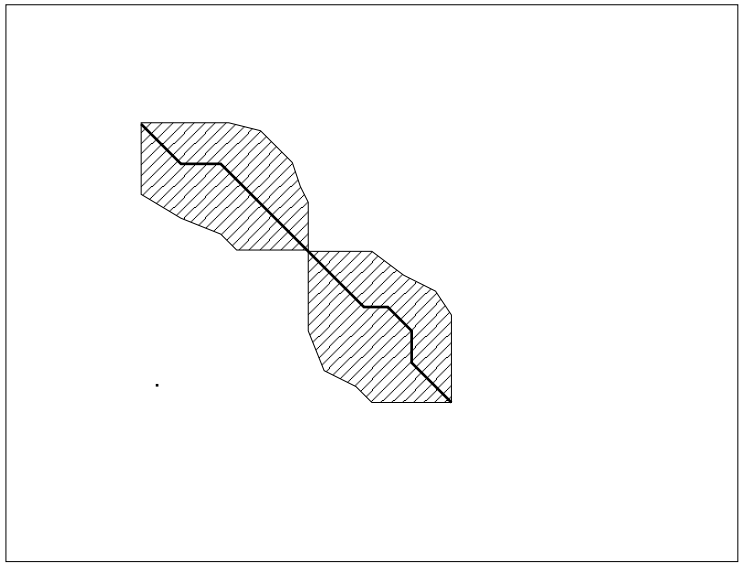
Let a query string be given.

- Find *hits* of length W with score $\geq T$ between substrings of the query string and the database strings.
  In protein Blast, $W = 3$ and $T = 11$. A database of potential hits can be created ahead of time.
- If a pair of hits on the same diagonal are close to each other (distance less than A) then they are probably part of a good ungapped local alignment.
- Extend such pairs in both directions to get an *ungapped* local alignment (just add up scores; this is fast).
  Stop the extension when the score of the alignment has fallen more than X below the maximum attained.
- If the resulting ungapped local alignment is good (score $\geq S_g$), then try to extend it to a *gapped* alignment.
  Choose two residues to force into alignment, and use dynamic programming to the left and right separately.
  Do not fill in an F matrix element if its score would fall more than X_g below the maximum attained.
  This procedure will fill in a *variable-width* band along the diagonal.

The word size W, the substitution matrix, and the gap (opening and extension) costs can be set under Options.

The X, W and X_g options can be set under Program Advanced Options.

Blast explores only part of the F matrix



The mysterious numbers at the end of the Blast output

In Blast output, X1 is X, X2 and (at traceback) X3 are X_g, S1 is S_g (and S2 is related to the expect value limit):

```
Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 2,361,291
Number of Sequences: 134328
Number of extensions: 84567
Number of successful extensions: 194
Number of sequences better than 10.0: 6
Number of HSP's better than 10.0 without gapping: 4
Number of HSP's successfully gapped in prelim test: 2
Number of HSP's that attempted gapping in prelim test: 185
Number of HSP's gapped (non-prelim): 8
...
T: 11
A: 40
X1: 16 ( 7.3 bits)
X2: 38 (14.6 bits)
X3: 64 (24.7 bits)
S1: 42 (22.0 bits)
S2: 60 (27.7 bits)
```


The speed of Blast and other database searches

Blast 2 is 100 times faster than Smith-Waterman, and nearly as sensitive and selective, according to experiments.

Still, the time to search for a sequence of length m in a database of length n is $O(mn)$.

That is, proportional to the size n of the database.

Thus for a given search sequence, doubling the size of the database will double the search time.

The size of nucleotide databases doubles every 9 months, but computer speed doubles only every 18 months.

Biosequence database searches get slower and slower . . . or more and more expensive.

Search in *business databases* or *web indexes* (e.g. Google) takes time $O(\log(n))$.

That is, proportional to the logarithm of the size n of the database.

Thus doubling the size of the database will only increase search time by a single unit.

Any record in a database with 1 000 000 000 records can be located in only 30 steps.

This is possible because the match is required to be exact (the correct account number, the correct CPR number).

Biosequence database search requires finding the *best inexact* match, which is much harder.