Pairwise alignment of biological sequences
Part 1

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

2004-09-07

Literature:
Durbin et al: *Biological Sequence Analysis*, Cambridge U P 1998, section 2.1, 2.2-2.5.
Claverie and Notredame: *Bioinformatics for Dummies*, Wiley 2003, chapter 8.
Altschul: Significance of alignment scores: http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html
Altschul et al: Gapped BLAST and PSI-BLAST: a new generation of protein database. . . , *Nucleic Acids Research* 25 (1) 1997.

Blast: searching sequence databases by sequence similarity

Given a query sequence such as PAWHEAE, Blast will search a database of sequences:

```
RMYDYDHVGDAPPPLQQYHRDASDHMHMFSNHQDVSTMDQRANKIA
RNFIGGFCLNQFECMCMGYVQYFDMTMIMSVARFSRKMKWYDDEKHQVWC
IHDKFHAIGEFPVLGPITSGEAEWSVQQHYVFKVANPDNWEWI
NMPYIYMQRSNIPTFFEFPHRFLRWNENPAWWWHEAECLICKRVWAIVEWFKAT
IKWLEPRTMYFKVYKSYFPHEAEQMIFDGSSIMYTPQTDYGDNDNSFHTCESMDH
MEHTAQCCSCFMHALNMPSVPFFTNVAAMEVCRTEAMIMQHTDYS
...
```

Blast then returns those database sequences that best match the query sequence.

How? It finds the best *alignment* between the query sequence and each database sequence.

The score of that alignment is the score of the database sequence.

All database similarity search programs work by alignment:

- WU-Blast by W. Gish from Washington University; slightly different from NCBI Blast.
- FastA by W. Pearson; slower but possibly more sensitive than Blast.
- . . . many others

What is an alignment?

A pairwise alignment is what Blast shows in the detailed search results:

```
Query: 10  SNIPTFFEFPHRFLRWNENPAWWWHEAE---CLICKRVWAIVEWFKA 53
           SNI    E+P  F++ +  P++W  +AE   C IC+ ++   E   A
Sbjct: 113 SNIGAVLEYPKDFIKESARPSYWVPDAEAPNCYICELLFGSPEELNA 159
```

Why pairwise alignment is important:

- Pairwise alignment is used in database searches.
  Blast and Fasta are essentially highly optimized versions of local pairwise alignment.
- Pairwise alignment is used to compute evolutionary distances, which are used to build phylogenetic trees.
  This is used in the multiple alignment programs ClustalW and T-COFFEE (next week).
- Pairwise alignment underlies multiple alignment, which is used to find concensus patterns.
- Pairwise alignment is used for sequence assembly in shotgun sequencing.

We describe the theory for amino acid sequences (proteins).

Nucleotide sequences are handled in much the same way.

Global alignment versus local alignment

- Global alignment (Needleman-Wunsch): all of x aligned with all of y:

```
HEAGAWGHE-E
--P-AW-HEAE
```

- Local alignment (Smith-Waterman): a subsequence of x aligned with a subsequence of y:

```
AWGHE
AW-HE
```

Linear gap costs versus affine gap costs

- Linear gap costs: a gap of length h has score $-8 \cdot h$
  So many small gaps are as good (same score) as one large gap.
- Affine gap costs: a gap of length h has score $-10 - 2 \cdot (h - 1)$
  So one large gap is better (higher score) than many small gaps.
  More complicated, but biologically more meaningful.

	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 case **4** — this is what Blast does.

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

Warming up: Ungapped global alignment of two strings; substitution matrices

Consider two strings, e.g. $x =$ HEAGA and $y =$ PAWHE of the same length $n = m = 5$.

Notation: x_i is the i'th letter from x. For instance, x_3 = A.

In an *ungapped alignment*, an amino acid x_i in x must be matched by an amino acid y_i in y:

```
HEAGA
PAWHE
```

The score of a match between amino acids x_i and y_i is $score[x_i][y_i]$, given by a *substitution matrix*.

It describes the likelihood that amino acid x_i was replaced (substituted) by amino acid y_i by an evolutionary event.

A high score means 'likely' and a low one means 'unlikely'.

An amino acid is most likely to remain the same, so the diagonal of the substitution matrix has high numbers.

The similarity of the strings x and y is just the sum of the scores:

$$sim(x, y) = score[x_1][y_1] + score[x_2][y_2] + score[x_3][y_3] + score[x_4][y_4] + score[x_5][y_5]$$

Often used substitution matrices include BLOSUM62 and PAM250.

So calculating the score of an ungapped alignment is easy. Problems arise when we allow *gaps* in x and y.

The BLOSUM50 substitution matrix (Henikoff and Henikoff 1992)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5

Case 1. Global alignment with linear gap costs (Needleman-Wunsch 1970)

We have two sequences $x =$HEAGAWGHEE and $y =$PAWHEAE with lengths $n = 10$ and $m = 7$.

Notation: $x_{1...i}$ is the prefix of x with i letters. For example, $x_{1...3}$ is HEA.

In a *gapped alignment*, an amino acid x_i is matched either by an amino acid y_j, or by a *gap* written '-':

```
HEAGAWGHE-E
---PAW-HEAE
```

The score of a match between amino acids x_i and y_j is $score[x_i][y_j]$, given by e.g. the BLOSUM50 matrix.

The score of a match between an amino acid and a gap is $-d$, where d may be 8.

We want to find an *optimal* global alignment of x and y: one that has the maximal sum of scores.

Naive attempt to find best gapped alignment

Enumerate all possible alignments of x and y, compute their scores, then choose one with maximal score:

```
HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE
PAWHEAE---     PAWHEA-E--     PAWHEA--E-     PAWHEA---E     PAWHE-A--E

HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE
PAWHE-AE--     PAWHE-A-E-     PAWHE--AE-     PAWHE---AE     PAWH-E--AE

HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE     HEAGAWGHEE
PAWH-EAE--     PAWH-EA-E-     PAWH-E-AE-     PAWH-E--AE     PAWH--E-AE
```

. . . and thousands of others.

Sadly, the number of possible matches for gapped alignment of two sequences of length n is *very* large:

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

For $n = 10$ this is roughly $2^{20} \approx 10^6$, and for $n = 150$ it is roughly $2^{300} \approx 10^{90}$.

Usually n is between 20 and several hundred. The number of atoms in the visible universe is approximately 10^{79}.

Thus this naive approach would be much too slow on even the fastest computers.

Useful observation:

Any prefix of the optimal alignment between x and y

is an optimal alignment between a prefix $x_{1\ldots i}$ of x and a prefix $y_{1\ldots j}$ of y.

So an optimal alignment can be computed by scanning x and y from left to right, recording only the optimal alignments between prefixes of x and y, and forgetting all the non-optimal ones.

More precisely, we can build a table F in which

$$F(i,j) = \text{ the maximal score for an alignment between } x_{1\ldots i} \text{ and } y_{1\ldots j}$$

Then, by definition, $F(n,m)$ is the maximal score for a global alignment between x and y.

This is because $x_{1..n}$ is the entire string x, and $y_{1..m}$ is the entire string y.

Useful observation:

The value $F(i,j)$ depends only on the values

$F(i-1,j-1)$, $F(i-1,j)$, and $F(i,j-1)$.

This is because an optimal alignment between $x_{1\ldots i}$ and $y_{1\ldots j}$ consists of either

- an optimal alignment between $x_{1\ldots(i-1)}$ and $y_{1\ldots(j-1)}$ extended with a match between x_i and y_j; or
- an optimal alignment between $x_{1\ldots(i-1)}$ and $y_{1\ldots j}$ extended with a match between x_i and a gap; or
- an optimal alignment between $x_{1\ldots i}$ and $y_{1\ldots(j-1)}$ extended with a match between a gap and y_j.

So we can fill in the F table from left to right and top to bottom.

This 'filling in the table' is called *dynamic programming* (Bellman 1955).

Table F gives us the maximal *score*. How find a corresponding optimal *alignment*?

When filling in $F(i,j)$, we record the traceback $B(i,j)$ from (i,j):

The traceback points at the cell that led to the maximal score: $(i-1,j-1)$ or $(i-1,j)$ or $(i,j-1)$.

When we are finished we find an optimal alignment just by following the traceback from (n,m) to $(0,0)$.

Filling in the F matrix for $x =$ **HEAGAWGHEE** and $y =$ **PAWHEAE**

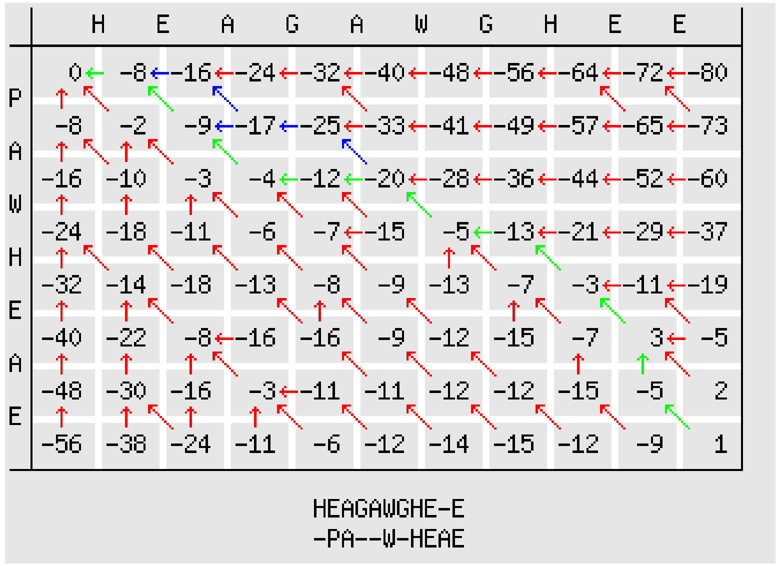
$y \backslash x$	H	E	A	G	A	W	G	H	E	E
P										
A										
W										
H										
E										
A										
E										

The filled-in F matrix for global alignment of $x =$ **HEAGAWGHEE** and $y =$ **PAWHEAE**

$y\backslash x$		H	E	A	G	A	W	G	H	E	E
	0	−8	−16	−24	−32	−40	−48	−56	−64	−72	−80
P	−8	−2	−9	−17	−25	−33	−41	−49	−57	−65	−73
A	−16	−10	−3	−4	−12	−20	−28	−36	−44	−52	−60
W	−24	−18	−11	−6	−7	−15	−5	−13	−21	−29	−37
H	−32	−14	−18	−13	−8	−9	−13	−7	−3	−11	−19
E	−40	−22	−8	−16	−16	−9	−12	−15	−7	3	−5
A	−48	−30	−16	−3	−11	−11	−12	−12	−15	−5	2
E	−56	−38	−24	−11	−6	−12	−14	−15	−12	−9	1

The traceback is recorded in a matrix B with the same shape as F.

Global alignment, linear gap costs



```
HEAGAWGHE-E
-PA--W-HEAE
```

Global alignment with linear gap costs: Initializing the matrix

Upper border: position $(i, 0)$ represents the alignment of $x_{1...i}$ to the empty prefix of y.

That is, the prefix $x_{1...i}$ has been matched with i gaps in y.

With simple linear gap costs, the score is $-d \cdot i$.

The traceback pointer at $(i, 0)$ points left, to $(i-1, 0)$.

The left-hand border is similar.

Hence we initialize the borders as follows:

```
for (int i=1; i<=n; i++) {
  F[i][0] = -d * i;
  B[i][0] = new Traceback2(i-1, 0);
}
for (int j=1; j<=m; j++) {
  F[0][j] = -d * j;
  B[0][j] = new Traceback2(0, j-1);
}
```

Implementing global alignment: Filling in the matrix

Position (i, j) may be reached

- from $(i-1, j-1)$ with a match, adding $score[x_i][y_j]$ to the score;
- from $(i-1, j)$ with a gap in y, subtracting d from the score; or
- from $(i, j-1)$ with a gap in x, subtracting d from the score.

The traceback $B(i, j)$ points to the source of the maximal resulting score $F(i, j)$, that is, 'where we came from':

```
for (int i=1; i<=n; i++)
  for (int j=1; j<=m; j++) {
    int s = score[x[i]][y[j]];
    int val = max(F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d);
    F[i][j] = val;
    if (val == F[i-1][j-1]+s)
      B[i][j] = new Traceback2(i-1, j-1);   // Up left
    else if (val == F[i-1][j]-d)
      B[i][j] = new Traceback2(i-1, j);     // Left
    else if (val == F[i][j-1]-d)
      B[i][j] = new Traceback2(i, j-1);     // Up
  }
B0 = new Traceback2(n, m);
```

The start `B0` of the traceback is cell (n, m) in the lower righthand corner.

Case 2. Local alignment with linear gap costs (Smith-Waterman 1981)

Align a subsequence of $x =$ `HEAGAWGHEE` with a subsequence of $y =$ `PAWHEAE`:

```
AWGHE
AW-HE
```

A local alignment can ignore a prefix and a suffix of each sequences.

New interpretation of $F(i, j)$:

$$F(i, j) = \text{ the maximal score for an alignment between a suffix of } x_{1...i} \text{ and a suffix of } y_{1...j}$$

Local alignment with linear gap costs: Initializing the matrix

Upper border: position $(i, 0)$ represents the alignment of a suffix of $x_{1...i}$ to an empty sequence.

An empty match, with score 0, is the best we can do (because gaps have negative scores).

Then $(i, 0)$ is the start of a new local alignment, and the traceback pointer at $(i, 0)$ points nowhere.

The left-hand border is similar.

Hence we initialize the border cells to 0 and the traceback to 'STOP'.

Implementing local alignment: Filling in the matrix

Position (i, j) in the F matrix may be reached

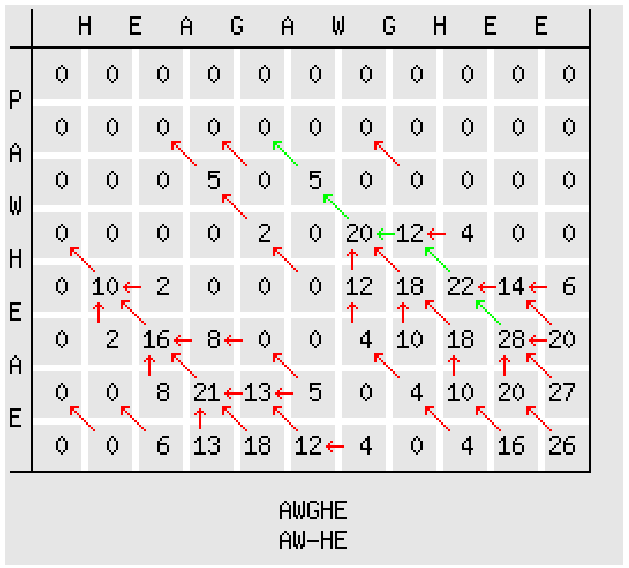
- from nowhere, with score 0, because we can always start a new local alignment;
- from $(i - 1, j - 1)$ with a match, adding $score[x_i][y_j]$ to the score;
- from $(i - 1, j)$ with a gap in y, subtracting d from the score; or
- from $(i, j - 1)$ with a gap in x, subtracting d from the score.

The traceback $B(i, j)$ points to the source of the maximal resulting score $F(i, j)$, if any. Thus:

```
for (int i=1; i<=n; i++)
  for (int j=1; j<=m; j++) {
    int s = score[seq1.charAt(i-1)][seq2.charAt(j-1)];
    int val = max(0, F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d);
    F[i][j] = val;
    if (val == 0)
      B[i][j] = null;
    else if (val == F[i-1][j-1]+s)
      B[i][j] = new Traceback2(i-1, j-1);
    else if (val == F[i-1][j]-d)
      B[i][j] = new Traceback2(i-1, j);
    else if (val == F[i][j-1]-d)
      B[i][j] = new Traceback2(i, j-1);
} }
```

The start `B0` of the traceback must be set to a cell (i, j) in F that has maximal score (there may be several).

Local alignment, linear gap costs



		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	5	0	0	0	0	0
W	0	0	0	0	2	0	20	12	4	0	0
H	0	10	2	0	0	0	12	18	22	14	6
E	0	2	16	8	0	0	4	10	18	28	20
A	0	0	8	21	13	5	0	4	10	20	27
E	0	0	6	13	18	12	4	0	4	16	26

```
AWGHE
AW-HE
```