sestoft@dina.kvl.dk

Algorithms for pairwise alignment

Consider two amino acid sequences `HEAGAWGHEE` and `PAWHEAE`.

Call them x and y, and denote their lengths by $n = 10$ and $m = 7$.

Chapter 2 presents four different alignment problems:

- global alignment: all of x must be aligned with all of y (Needleman-Wunsch):

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

- local alignment: a subsequence of x must be aligned with a subsequence of y (Smith-Waterman):

```
AWGHE
AW-HE
```

- repeated matches: all of x must be aligned with some (possibly repeated) subsequences of y:

```
HEAGAWGHEE
HEA.AW-HE.
```

- overlap matches: a prefix or suffix of x must be aligned with a prefix or suffix of y:

```
GAWGHEE
PAW-HEA
```


Global alignment (Needleman-Wunsch)

In an alignment, an amino acid x_i is matched either by an amino acid y_j, or by a gap.

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:

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

But ... the number of possible matches for two sequences of length $n = 10$ and $m = 7$ is

$$\binom{n+m}{n} = \frac{(n+m)!}{m!\,n!} = 19448$$

For $n = 100$ and $m = 7$, that would be $6491922168400302104990847829084871669228062213140$.

Clearly infeasible to enumerate all possible alignments.

Observation 1:

> 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.

Observation 2:

> 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 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 $x = \texttt{HEAGAWGHEE}$ and $y = \texttt{PAWHEAE}$

$x \backslash y$	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 $x = \texttt{HEAGAWGHEE}$ and $y = \texttt{PAWHEAE}$

$x \backslash y$		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.

Implementing global alignment: Initialization

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

That is, the prefix $x_{1\ldots i}$ 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 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 $F(i, j)$. 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(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);
    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);
  }
B0 = new Traceback2(n, m);
```

The start `B0` of the traceback is cell (n, m).

Local alignment of $x =$ **HEAGAWGHEE** and $y =$ **PAWHEAE** (Smith-Waterman)

A subsequence of x must be aligned with a subsequence of y:

```
AWGHE
AW-HE
```

Requirement: the expected score of a random match must be negative.

If the score of a random match extension were positive, then any local alignment could be profitably extended to a 'better' (but probably biologically meaningless) one.

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

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

Implementing local alignment: Initialization

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

An empty match, with score 0, is the best we can do (provided 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 `null` (this requires no action in Java).

Implementing local alignment: Filling in the matrix

Position (i,j) 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 some cell (i,j) in F with maximal score.

Repeated matches of $y =$ **PAWHEAE** in $x =$ **HEAGAWGHEE**

All of x must be aligned with some (possibly repeated) subsequences from y:

```
HEAGAWGHEE
HEA.AW-HE.
```

A dash (`-`) indicates that the corresponding x_i is matched by a gap in a y subsequence.

A dot (`.`) indicates that the corresponding x_i is matched by no subsequence of y.

Every two matched subsequences of x are separated by one or more unmatched subsequences.

Assume we are interested only in matches scoring higher than some threshold T, e.g. 20.

Otherwise we might find many (low-score) random matches.

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

$$\begin{aligned} F(i,0) &= \text{the best sum of match scores up to } x_{1\ldots i} \text{ provided } i \text{ is in an unmatched region of } x \\ F(i,j) &= \text{the best sum of match scores up to } x_{1\ldots i} \text{ provided } i \text{ is in a matched region of } x \end{aligned}$$

Implementing repeated matches: Initialization

Left-hand border:

Position $(0, j)$ represents the best alignment of an empty subsequence of x to a subsequence of y.

This must have score 0.

The traceback pointer at $(0, j)$ points nowhere.

Implementing repeated matches: Filling in the matrix

Position $(i, 0)$ may be reached

- from $(i-1, 0)$ by letting x_i be unmatched by any part of y, keeping the old score; or
- from $(i-1, j)$ by completing a match whose score is at least T, subtracting T from that score.

Position (i, j) for $j > 0$ may be reached

- from $(i, 0)$, because we start a new local alignment, keeping the old score;
- 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.

As always, the traceback $B(i, j)$ points to the source of the maximal resulting score $F(i, j)$.

Let `maxj(i-1)` be $j > 0$ if $F(i-1, j) - T$ is greater than $F(i-1, 0)$ and maximal; otherwise 0.

This gives:

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

That is, (n, j) if there is a last match with score $> T$; otherwise $(n, 0)$, if some suffix of x is unmatched.

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.

Reducing the space consumption of global alignment

All algorithms require time $O(nm)$ to fill in the tables and space $O(nm)$ to store the tables.

However, column i of F depends only on column $i-1$.

So only two columns of F (and the traceback) need to be stored at the same time.

Hence we can compute the best score using only space $O(n+m)$.

How reconstruct the optimal global alignment?

When $n \leq 1$ or $m \leq 1$, use the standard algorithm (in this case it uses little space anyway).

Otherwise, let $u = n/2$ and assume the optimal alignment passes through (u, v).

(We can determine v while filling in F).

Recursively determine

- the optimal global alignment z_1 between $x_{1 \ldots u}$ and $y_{1 \ldots v}$; and
- the optimal global alignment z_2 between $x_{(u+1) \ldots n}$ and $y_{(v+1) \ldots m}$

Then the optimal alignment between x and y is the concatenation of z_1 and z_2.

Reducing the space consumption of local alignment

Fill in F using only space $O(n+m)$ as above.

Keep track of the starting point (s_1, s_2) and the ending point (e_1, e_2) of the local alignment with highest score.

Compute the optimal *global* alignment between the subsequences $x_{s_1 \ldots e_1}$ and $y_{s_2 \ldots e_2}$ in space $O(n+m)$.

The result is also the optimal *local* alignment between x and y.