COMPSCI 369: Alignment with Affine Gap Scores

Michael J. Dinneen

Email: mjd@cs.auckland.ac.nz
Room: 303S.579; Ext: 87868

http://www.cs.auckland.ac.nz/∼mjd

March 2007

How related are two sequences/strings?

Some key issues in deciding how two sequences compare:

1. What sorts of alignments should be considered.
2. The scoring system used to rank alignments.
3. The algorithm used to find optimal (or good) alignments.
4. The statistacal methods used to evaluate the score(s).

The scoring model

From the mutational process of *substitutions*, *insertions* and *deletions* we want to find the minimum total cost to transform one string to another.

A *substition matrix* tells biologist how much it costs to align pairs of characters.

- A positive score is given for matches.
- A negative score is usually given for substitutions.
- We expect to penalize gaps of length g:
  - linear score model: $\gamma(g) = -gd$
  - affine score model: $\gamma(g) = -d - (g-1)e$

  where d is the *gap-open* penalty and e is the *gap-extension* penalty.

Global alignment using linear (gap) scores

We have already seen the **Needleman–Wuncsh** dynamic programming algorithm.

Let $s(x_i, y_i)$ be the substitution score for characters x_i and y_i. If $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$ are two strings then we can compute the best alignment $F(n, m)$ using:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

$$F(k, 0) = F(0, k) = -kd$$

Local alignment using linear (gap) scores

The highest scoring alignment of subsequences of two strings is called the best *local alignment*. The **Smith–Waterman** dynamic programming algorithm is a slight modified version of the global alignment algorithm.

Let $s(x_i, y_i)$ be the substitution score for characters x_i and y_i. If $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$ are two strings then we can compute the best alignment $F(n, m)$ using:

$$F(i,j) = \max \begin{cases} 0 \quad \text{— ignore accumulated bad matches} \\ F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

$$F(k,0) = F(0,k) = 0 \quad \text{— no penalty at the ends}$$

Global alignment using affine (gap) scores

We can also easily modify the Needleman–Wunsch algorithm to compute optimal alignments with affine scoring.
Let $s(x_i, y_i)$ be the substitution score for characters x_i and y_i. If $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$ are two strings then we can compute the best alignment $F(n, m)$ using:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j), & \\ F(k,j) + \gamma(i-k), & k = 0, \ldots, i-1 \\ F(i,k) + \gamma(j-k), & k = 0, \ldots, j-1 \end{cases}$$

$$F(k,0) = F(0,k) = \gamma(k) = -d - (k-1)e$$

But, the running time is now $\Theta((n+m)^3)$!!!

Global alignment using affine (gap) scores

1. Let $M(i, j)$ be the best score with x_i aligned to y_j.
2. Let $I_x(i, j)$ be the best score with x_i aligned to a gap.
3. Let $I_y(i, j)$ be the best score with y_j aligned to a gap.

$$\begin{array}{cccccccc} x_1 & x_2 & \dots & x_a & \dots & x_i & & \\ ? & ? & & \| & & \| & & M(i,j) \\ y_1 & y_2 & \dots & y_b & \dots & y_j & & \end{array}$$

$$\begin{array}{cccccccc} x_1 & x_2 & \dots & x_a & \dots & & x_i & \\ ? & ? & & \| & \| & & \| & I_x(i,j) \\ y_1 & y_2 & \dots & y_b & \dots & y_j & \boxed{\quad} & \end{array}$$

$$\begin{array}{cccccccc} x_1 & x_2 & \dots & x_a & \dots & x_i & \boxed{\quad} & \\ ? & ? & & \| & & \| & \| & I_y(i,j) \\ y_1 & y_2 & \dots & y_b & & \dots & y_j & \end{array}$$

We get a quadradic-time algorithm by introducing a constant factor more subproblems.
I.e, 3 tables of size $O(mn)$
$0 \leq i \leq n$ and $0 \leq j \leq m$

Global alignment using affine (gap) scores

An efficient programming relations to compute affine gap scores for strings $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$ is now $\max\{M(n,m), I_x(n,m), I_y(n,m)\}$ where

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i,y_j) \\ I_x(i-1,j-1) + s(x_i,y_j) \\ I_y(i-1,j-1) + s(x_i,y_j) \end{cases}$$

$$I_x(i,j) = \max \begin{cases} M(i-1,j) - d \\ I_x(i-1,j) - e \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i,j-1) - d \\ I_y(i,j-1) - e \end{cases}$$

$$I_x(k,0) = I_y(0,k) = \gamma(k) = -d - (k-1)e, \quad k > 0$$

Illustrating affine gap algorithm

$s(,)$	A	B	C	D	E
A	8	0	-3	-2	-4
B	0	9	-5	0	-7
C	-3	-5	5	-3	-4
D	-2	0	-3	6	-4
E	-4	-7	-4	-4	8

Substituion costs with gap penalties $d = 6$ and $e = 1$ consider strings:

$$X = ACBAE$$

$$Y = DACABE$$

Illustrating affine gap algorithm (cont)

Best score for aligning last characters of (prefixes of) X and Y:

$M(,)$		D	A	C	A	B	E
	0	-6	-7	-8	-9	-10	-11
A	-6	-2	**2**	-10	0	-9	-14
C	-7	-9	-5	**7**	-7	-5	-10
B	-8	-7	-8	-9	**7**	10	-7
A	-9	-10	1	-8	9	**7**	6
E	-10	-13	-14	-3	-4	2	**15**

Illustrating affine gap algorithm (cont)

Best score for aligning last character of (a prefix of) X to a gap:

$I_X(,)$		D	A	C	A	B	E
	0	-6	-7	-8	-9	-10	-11
A	-6	-7	-8	-9	-10	-11	-12
C	-7	-8	-4	-10	-6	-12	-13
A	-8	-9	-5	1	-7	-11	-14
B	-9	-10	-6	0	1	4	-13
E	–	–	–	–	–	–	0

Illustrating affine gap algorithm (cont)

Best score for aligning last character of (a prefix of) Y to a gap:

$I_Y(,)$		D	A	C	A	B	E
	0	-6	-7	-8	-9	-10	–
A	-6	-7	-8	-4	-5	-6	–
C	-7	-8	-9	-10	1	0	–
A	-8	-9	-10	-11	-12	1	–
B	-9	-10	-11	-5	-6	3	–
E	-10	-11	-12	-13	-9	-10	-4

Illustrating affine gap algorithm (cont)

Best alignment is:

	A	C	B	A	E	
D	A	C	A	B	E	
-6	8	5	0	0	8	**=15**

For comparison here are two other non-optimal alignments:

	A	C		B	A	E	
D	A	C	A	B		E	
-6	8	5	-6	9	-6	8	**=12**

	A	C	B	A		E	
D	A	C		A	B	E	
-6	8	5	-6	8	-6	8	**=11**