Lecture 5

Local Alignment, and Gap Penalties

January 18, 2000
Notes: Martin Tompa

5.1. Computing an Optimal Local Alignment by Dynamic Programming

BASIS : For simplicity, we will make the reasonable assumption that $\sigma(x,-) \le 0$ and $\sigma(-,x) \le 0$. Then

$$\begin{aligned} v(i,0) &= 0, \text{ and} \\ v(0,j) &= 0, \end{aligned}$$

since the optimal suffix to align with a string of length 0 is the empty suffix.

RECURRENCE : for $i > 0$ and $j > 0$,

$$v(i,j) = \max\begin{pmatrix} 0 & & & , \\ v(i-1,j-1) & + & \sigma(S[i],T[j]) & , \\ v(i-1,j) & + & \sigma(S[i],-) & , \\ v(i,j-1) & + & \sigma(-,T[j]) & \end{pmatrix}$$

The formula looks very similar to the recurrence for the optimal global alignment in Section 4.1. Of course, the meaning is somewhat different and we have an additional term in the $\max$ function. The recurrence is explained as follows. Consider an optimal alignment A of a suffix α of $S[1]\cdots S[i]$ and a suffix β of $T[1]\cdots T[j]$. There are four possible cases:

1. $\alpha = \lambda$ and $\beta = \lambda$, in which case the alignment has value 0.
2. $\alpha \ne \lambda$ and $\beta \ne \lambda$, and the last matched pair in A is $(S[i],T[j])$, in which case the remainder of A has value $v(i-1,j-1)$.
3. $\alpha \ne \lambda$, and the last matched pair in A is $(S[i],-)$, in which case the remainder of A has value $v(i-1,j)$.
4. $\beta \ne \lambda$, and the last matched pair in A is $(-,T[j])$, in which case the remainder of A has value $v(i,j-1)$.

The optimal alignment chooses whichever of these cases has greatest value.

5.1.1. Example

For example, let S = `abcxdex` and T = `xxxcde`, and suppose a match scores $+2$, and a mismatch or a space scores -1. The dynamic programming algorithm fills in the table of $v(i,j)$ values from top to bottom and left to right, as follows:

	j	0	1	2	3	4	5	6
i			x	x	x	c	d	e
0		0	0	0	0	0	0	0
1	a	0	0	0	0	0	0	0
2	b	0	0	0	0	0	0	0
3	c	0	0	0	0	2	1	0
4	x	0	2	2	2	1	1	0
5	d	0	1	1	1	1	3	2
6	e	0	0	0	0	0	2	5
7	x	0	2	2	2	1	1	4

The value of the optimal local alignment is $v(6,6) = 5$. We can reconstruct optimal alignments as in Section 4.1.2, by retracing from any maximum entry to any zero entry:

	j	0	1	2	3	4	5	6
i			x	x	x	c	d	e
0		0	0	0	0	0	0	0
1	a	0	0	0	0	0	0	0
2	b	0	0	0	0	0	0	0
3	c	0	0	0	0	$\nwarrow$ 2	1	0
4	x	0	2	2	$\nwarrow$ 2	$\hookleftarrow\uparrow$ 1	1	0
5	d	0	1	1	1	1	$\nwarrow$ 3	2
6	e	0	0	0	0	0	2	$\nwarrow$ 5
7	x	0	2	2	2	1	1	4

The optimal local alignments corresponding to these paths are

```
c x d e          x - d e
c - d e    and   x c d e   .
```

Both alignments have three matches and one space, for a value of $3 \cdot (2) + 1 \cdot (-1) = 5$. You can also see from this diagram how the value was derived in the example following Definition 4.5, which said that for the same strings S and T, $v(5,5) = 3$.

5.1.2. Time Analysis

Theorem 5.1: The dynamic programming algorithm computes an optimal local alignment in time $O(nm)$.

Proof: Computing the value for each of the $(n+1)(m+1)$ entries requires at most 6 table lookups, 3 additions, and 1 max calculation. Reconstructing a single alignment can then be done in time $O(n+m)$. □

5.2. Space Analysis

The space required for either the global or local optimal alignment algorithm is also quadratic in the length of the strings being compared. This could be prohibitive for comparing long DNA sequences. There is a modification of the dynamic programming algorithm that computes an optimal alignment in $O(n+m)$ space and still runs in $O(nm)$ time. If one were interested only in the value of an optimal alignment, this could be done simply by retaining only two consecutive rows of the dynamic programming table at any time. Reconstructing an alignment is somewhat more complicated, but can be done in $O(n+m)$ space and $O(nm)$ time with a divide and conquer approach (Hirschberg [4], Myers and Miller [7]).

5.3. Optimal Alignment with Gaps

Definition 5.2: A *gap* in an alignment of S and T is a maximal substring of either S' or T' consisting only of spaces. (Recall from Definition 3.3 that S' and T' are S and T with spaces inserted as dictated by the alignment.)

5.3.1. Motivations

In certain applications, we may not want to have a penalty proportional to the length of a gap.

1. Mutations causing insertion or deletion of large substrings may be considered a single evolutionary event, and may be nearly as likely as insertion or deletion of a single residue.
2. cDNA matching: Biologists are very interested in learning which genes are expressed in which types of specialized cells, and where those genes are located in the chromosomal DNA. Recall from Section 2.5 that eukaryotic genes often consist of alternating exons and introns. The mature mRNA that leaves the nucleus after transcription has the introns spliced out. To study gene expression within specialized cells, one procedure is as follows:
   (a) Capture the mature mRNA as it leaves the nucleus.
   (b) Make *complementary DNA* (abbreviated *cDNA*) from the mRNA using an enzyme called *reverse transcriptase*. The cDNA is thus a concatenation of the gene's exons.
   (c) Sequence the cDNA.
   (d) Match the sequenced cDNA against sequenced chromosomal DNA to find the region of chromosomal DNA from which the cDNA derives. In this process we do not want to penalize heavily for the introns, which will match gaps in the cDNA.

In general, the gap penalty may be some arbitrary function $g(q)$ of the gap length q. The best choice of this function, like the best choice of a scoring function, depends on the application. In the cDNA matching application, we would like the penalty to reflect what is known about the common lengths of introns. In the next section we will see an $O(nm)$ time algorithm for the case when $g(q)$ is an arbitrary linear affine function, and this is adequate for many applications. There are programs that use piecewise linear functions as gap penalties, and these may be more suitable in the cDNA matching application. There are $O(nm \log m)$ time algorithms for the case when $g(q)$ is concave downward (Galil and Giancarlo [3], Miller and Myers [6]). We could even implement an arbitrary function as a gap penalty function, but the known algorithm for this requires cubic time (Needleman and Wunsch [8]), and such an algorithm is probably not useful in practice.

5.3.2. Affine Gap Model

We will study a model in which the penalty for a gap has two parts: a penalty for initiating a gap, and another penalty that depends linearly on the length of a gap. That is, the gap penalty is $W_g + qW_s$ where W_g and W_s are both constants, $W_g \geq 0$, $W_s \geq 0$, and $q \geq 1$ is the length of the gap. (Note that the model with a constant penalty regardless of gap length is the special case with $W_s = 0$.)

For simplicity, assume we are modifying the global alignment algorithm of Section 4.1 to accommodate an affine gap penalty. Similar ideas would work for local alignment as well.

We will assume $\sigma(x, -) = \sigma(-, x) = 0$, since the spaces will be penalized as part of the gap. Our goal then is to maximize

$$\sum_{i=1}^{\ell} \sigma(S'[i], T'[i]) - W_g(\text{\# gaps}) - W_s(\text{\# spaces}),$$

where S' and T' are S and T with spaces inserted, and $|S'| = |T'| = l$.

5.3.3. Dynamic Programming Algorithm

Once again, the algorithm proceeds by aligning $S[1] \cdots S[i]$ with $T[1] \cdots T[j]$. For these prefixes of S and T, define the following variables:

1. $V(i,j)$ is the value of an optimal alignment of $S[1] \cdots S[i]$ and $T[1] \cdots T[j]$.
2. $G(i,j)$ is the value of an optimal alignment of $S[1] \cdots S[i]$ and $T[1] \cdots T[j]$ whose last pair matches $S[i]$ with $T[j]$.
3. $F(i,j)$ is the value of an optimal alignment of $S[1] \cdots S[i]$ and $T[1] \cdots T[j]$ whose last pair matches $S[i]$ with a space.
4. $E(i,j)$ is the value of an optimal alignment of $S[1] \cdots S[i]$ and $T[1] \cdots T[j]$ whose last pair matches a space with $T[j]$.

BASIS :

$$\begin{aligned} V(0,0) &= 0, \\ V(i,0) &= -W_g - iW_s, \text{ for } i > 0, \\ V(0,j) &= -W_g - jW_s, \text{ for } j > 0, \\ E(i,0) &= -\infty, \text{ for } i > 0, \\ F(0,j) &= -\infty, \text{ for } j > 0. \end{aligned}$$

RECURRENCE : For $i > 0$ and $j > 0$,

$$V(i,j) = \max(G(i,j),\ F(i,j),\ E(i,j)),$$

$$\begin{aligned}
G(i,j) &= V(i-1,j-1)+\sigma(S[i],T[j]), \\
F(i,j) &= \max(F(i-1,j)-W_s,\, V(i-1,j)-W_g-W_s), \\
E(i,j) &= \max(E(i,j-1)-W_s,\, V(i,j-1)-W_g-W_s).
\end{aligned}$$

The equation for $F(i,j)$ (and analogously $E(i,j)$) can be understood as taking the maximum of two cases: adding another space to an existing gap, and starting a new gap. To understand why starting a new gap can use $V(i-1,j)$, which includes the possibility of an alignment ending in a gap, consider that $V(i-1,j) = \max(G(i-1,j),\ F(i-1,j),\ E(i-1,j))$, so that $F(i-1,j)-W_g-W_s$ is always dominated by $F(i-1,j)-W_s$, so will never be chosen by the max.

5.3.4. Time Analysis

Theorem 5.3: An optimal global alignment with affine gap penalty can be computed in time $O(nm)$.

Proof: The algorithm proceeds as those we have studied before, but in this case there are three or four matrices to fill in simultaneously, depending on whether you store the values of $V(i,j)$ or calculate them from the other three matrices when needed. □

5.4. Bibliographic Notes on Alignments

Bellman [2] began the systematic study of dynamic programming. The original paper on global alignment is that of Needleman and Wunsch [8]. Smith and Waterman [9] introduced the local alignment problem, and the $O(nm)$ algorithm to solve it. A number of authors have studied the question of how to construct a good scoring function for sequence comparison, including Karlin and Altschul [5] and Altschul [1].

References

[1] S. F. Altschul. A protein alignment scoring system sensitive at all evolutionary distances. *Journal of Molecular Evolution*, 36(3):290–300, Mar. 1993.

[2] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[3] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theoretical Computer Science*, 64:107–118, 1989.

[4] D. S. Hirschberg. A linear-space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, June 1975.

[5] S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Science USA*, 87(6):2264–2268, Mar. 1990.

[6] W. Miller and E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50(2):97–120, 1988.

[7] E. W. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4:11–17, 1988.

[8] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[9] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, Mar. 1981.