Com S 551 Spring 2015

Comparison of Two DNA Sequences: Local Alignment

1 Local Alignment

We first define a local alignment model. Then we describe a dynamic programming algorithm for computing an optimal local alignment between two sequences.

1.1 A local alignment model

The global alignment algorithm is intended for sequences that are similar over their entire lengths. However, there are often situations where two sequences are not globally similar, but contain similar regions. In this case, a local alignment algorithm is needed to find similar regions between two sequences. A local alignment between two sequences A and B is an alignment of a region of A and a region of B. As in Part 1, a local alignment consists of substitutions, deletion gaps and insertion gaps. A substitution pairs a letter of A with a letter of B. A substitution is a match if the two letters are identical and a mismatch otherwise. A deletion gap is a gap where letters of A correspond to no letter of B, and an insertion gap is a gap where letters of B correspond to no letter of A. The length of a gap is the number of letters involved.

Let numbers q and r be gap-open and gap-extension penalties, respectively. The numbers q and r are nonnegative. The score of a gap of length k is $-(q + k \times r)$. Let $\sigma(a, b)$ be the score of a substitution involving letters a and b. Values for the parameters σ, q and r are specified by the user. A letter-independent substitution table is usually used for comparison of DNA sequences. For example, each match is given a score of 10 and each mismatch a score of -20. Possible values for q and r are 40 and 2, respectively, for DNA sequences. Note that $\sigma(a, b)$ is the match score if a and b are identical and the mismatch score otherwise. The score of a local alignment is the sum of scores of each substitution and each gap.

Below are two DNA sequences along with a local alignment between them. This alignment contains 19 matches, 0 mismatch, an insertion gap of length 2, and a deletion gap of length 1. The score of the local alignment is 104 $(19*10+0*(-20)-(40+2*2)-(40+1*2)=104)$ using the given set of values for DNA sequences. An optimal local alignment between two sequences A and B is one with the maximum score. The local alignment in this example is an optimal local alignment.

```
Sequence A: GATCGTAGAGTGAGACCTAGTGTTTG
    Length: 26
Sequence B: CTCGTAGGTGAGATTCCTAGTGCC
    Length: 24

          Match Score: 10
       Mismatch Score: -20
     Gap-Open Penalty: 40
Gap-Extension Penalty: 2

    Alignment Score: 104
             Length: 22
Start Position in A: 3
Start Position in B: 2
  End Position in A: 22
  End Position in B: 22

     1     .    :    .    :
     3 TCGTAGAGTGAGA  CCTAGTG 22
       ||||||-||||||--|||||||
     2 TCGTAG GTGAGATTCCTAGTG 22
```

1.2 A dynamic programming algorithm

Let $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ be two sequences (called strings in C) of lengths m and n. The sequence A is stored on a computer with a **char** array AR of length $m+1$ with a_i in $AR[i]$ for each $1 \leq i \leq m$ and with $AR[0]$ unused. The sequence B is stored with a **char** array of length $n+1$. A technique called dynamic programming in computer science is used to compute an optimal local alignment between A and B. In this technique, a matrix S is introduced: $S(i,j)$ is the maximum score of all local alignments starting at positions $i+1$ and $j+1$ of A and B. To compute the matrix S efficiently, two additional matrices D and I are introduced. Let $D(i,j)$ (D for deletion) be the maximum score of those local alignments that begin with a deletion gap. Let $I(i,j)$ (I for insertion) be the maximum score of those local alignments that begin with an insertion gap.

An efficient method for computing the matrices S, D and I is developed based on the observation that no optimal local alignment can begin or end with a portion of a negative score. The method performs the computation in a proper order so that for each entry (i,j), the matrix scores $S(i,j)$, $D(i,j)$ and $I(i,j)$ at the entry can be computed in a constant time according to some formulas. We specify how each matrix is computed in each of the three or four conditions for Part 1; each formula is used for the entries that meet the conditions given after the formula. Note that $\sigma(a_{i+1}, b_{j+1})$ is the match score if a_{i+1} is identical to b_{j+1} and the mismatch score otherwise.

$S(m,n) = 0$,
$S(i,n) = 0$ for $m > i \geq 0$,
$S(m,j) = 0$ for $n > j \geq 0$,
$S(i,j) = \max\{0, S(i+1,j+1) + \sigma(a_{i+1}, b_{j+1}), D(i,j), I(i,j)\}$
for $m > i \geq 0$ and $n > j \geq 0$.

$D(m,j) = -(q+r)$ for $n \geq j \geq 0$,
$D(i,n) = -(q+r)$ for $m > i \geq 0$,
$D(i,j) = \max\{D(i+1,j) - r, S(i+1,j) - q - r\}$ for $m > i \geq 0$ and $n > j \geq 0$.

$I(i,n) = -(q+r)$ for $m \geq i \geq 0$,
$I(m,j) = -(q+r)$ for $n > j \geq 0$,

$I(i,j) = \max\{I(i,j+1) - r, S(i,j+1) - q - r\}$ for $m > i \geq 0$ and $n > j \geq 0$.

The zero in the formulas for S is the score of the empty local alignment, an alignment of two regions of length 0. The zero serves two purposes. First, an optimal local alignment can end at any positions in A and B. There is no penalty for not including, in the optimal local alignment, terminal regions of A and B after the positions. Second, any local alignment of a negative score is ignored because it cannot be a terminal portion of any optimal local alignment.

The matrices can be computed in decreasing order of rows and then in decreasing order of columns. For entry (m,n), compute, in any order, $S(m,n)$, $D(m,n)$ and $I(m,n)$. Then for each j from $n-1$ downto 0, compute, in any order, $S(m,j)$, $D(m,j)$ and $I(m,j)$. Next, for each i from $m-1$ downto 0, first compute $S(i,n)$, $D(i,n)$ and $I(i,n)$, and then for each j from $n-1$ downto 0, compute $I(i,j)$, $D(i,j)$ and $S(i,j)$ in this order. An entry $(rowfirst, colfirst)$ with the maximum value in the matrix S is the end point of an optimal local alignment between A and B. The maximum value $S(rowfirst, colfirst)$ is the score of an optimal local alignment. The complete computation is given in pseudocode in Figure 1.

```
$score = 0$;
$rowfirst = m$;
$colfirst = n$;
$S(m, n) = 0$;
$D(m, n) = -(q + r)$;
$I(m, n) = -(q + r)$;
for $j$ going from $n - 1$ downto $0$
    {
        $S(m, j) = 0$;
        $I(m, j) = -(q + r)$;
        $D(m, j) = -(q + r)$;
    }
for $i$ going from $m - 1$ downto $0$
    {
        $S(i, n) = 0$;
        $D(i, n) = -(q + r)$;
        $I(i, n) = -(q + r)$;
        for $j$ going from $n - 1$ downto $0$
            {
                $D(i, j) = max(D(i + 1, j) - r, S(i + 1, j) - q - r)$;
                $I(i, j) = max(I(i, j + 1) - r, S(i, j + 1) - q - r)$;
                $S(i, j) = max(0, S(i + 1, j + 1) + \sigma(a_{i+1}, b_{j+1}), D(i, j), I(i, j))$;
                if ( $score < S(i, j)$ )
                    {
                        $score = S(i, j)$;
                        $rowfirst = i$;
                        $colfirst = j$;
                    }
            } // inner loop
    } // outer loop
```

Figure 1. Computation of the three matrices in pseudocode.

An optimal local alignment is found by a traceback procedure on the matrices S, D, and I. An optimal local alignment corresponds to a path from entry $(rowlast, collast)$ in the matrix S to entry $(rowfirst, colfirst)$ in the matrix S, where entry $(rowfirst, colfirst)$ has the maximum value in the matrix S and $S(rowlast, collast) = 0$. Entry (i, j) in the matrix S is referred to as position S of entry (i, j), entry (i, j) in D as position D of entry (i, j), and entry (i, j) in I as position I of entry (i, j). These positions are shown in a grid graph in Figure 2. Let the current position be a newly determined position. An optimal path is recovered by repeatedly determining a position that is immediately after the current position on the path. Thus the pairs of an optimal alignment are generated in a forward order with the first pair produced first. Initially, the current position is position S of entry $(rowfirst, colfirst)$.

First consider the case where the current position is position S of entry (i, j). The recurrences for S are used to determine a new position. If $i = m$ or $j = n$ or $S(i, j) = 0$, then the traceback procedure terminates with $rowlast = i$ and $collast = j$. Otherwise, if $S(i, j) = D(i, j)$, then the new position is position D of entry (i, j). Otherwise, if $S(i, j) = I(i, j)$, then the new position is position I of entry (i, j). Otherwise, the new position is position S of entry $(i + 1, j + 1)$ and a new pair for the optimal alignment is a substitution pair (a_{i+1}, b_{j+1}).

Next consider the case where the current position is position D of entry (i, j). The recurrences for D are used to determine a new position. If $i = m - 1$ or $D(i, j) = S(i + 1, j) - q - r$, then the new position is position S of entry $(i + 1, j)$. Otherwise, then the new position is position D of entry $(i + 1, j)$. In each situation, a new pair for the optimal alignment is a deletion pair $(a_{i+1}, -)$. The case where the current position is position I of entry (i, j) is similarly handled. The traceback method is given in pseudocode in Figure 3.

The start and end positions of the optimal local alignment in A are $rowfirst + 1$ and $rowlast$, and those in B are $colfirst + 1$ and $collast$. The score of the alignment is $S(rowfirst, colfirst)$.

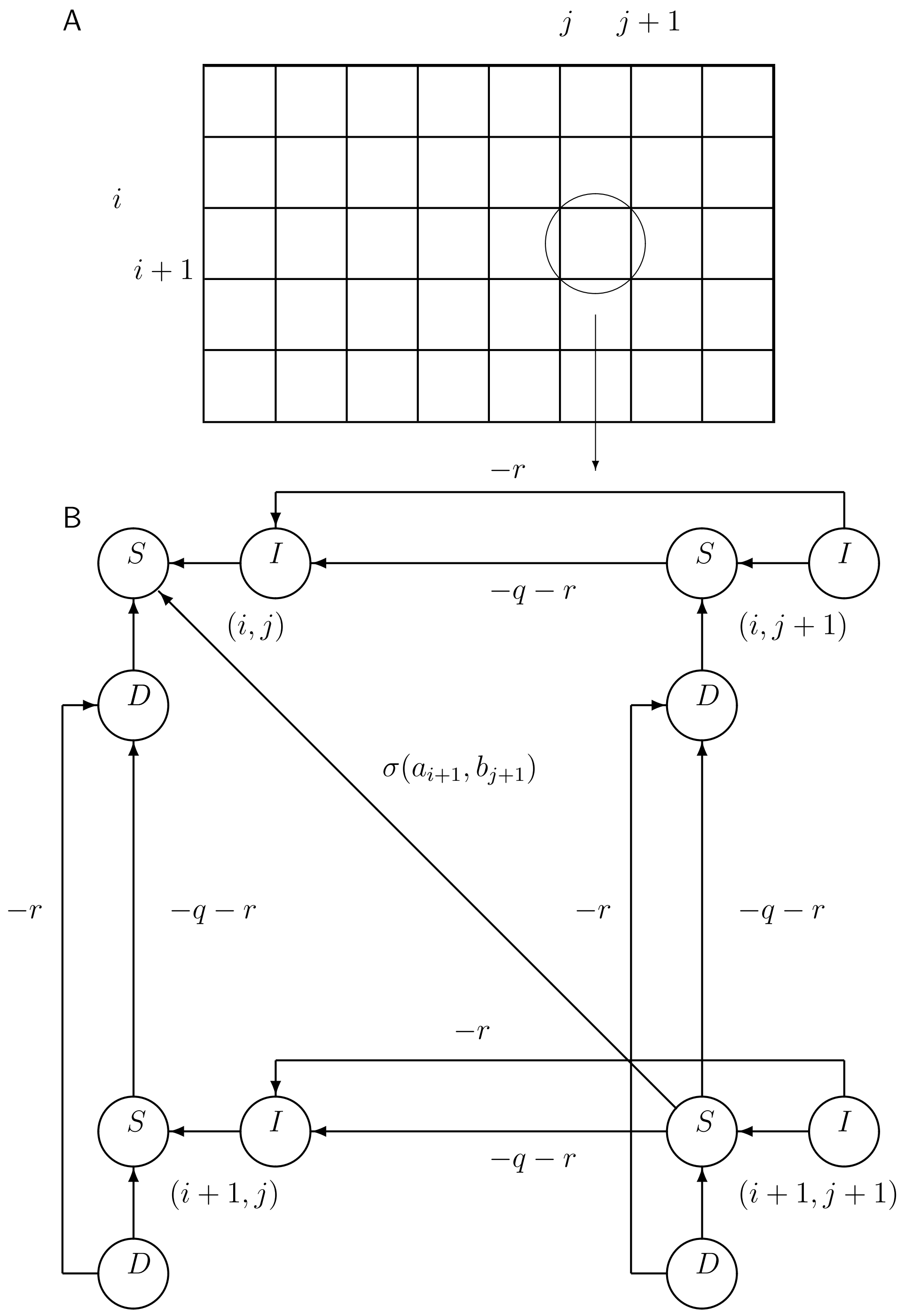

Figure 2. A grid graph with no cycle. (A) An overview of the graph.
(B) A detailed view of four adjacent entries in the graph.
The edges from D to S and from I to S have a score of 0.
The score of each remaining edge is shown next to the edge.

```
let $OA$ be empty;
$i = rowfirst$; $j = colfirst$; $mat$ = 'S';
while ( $i \le m$ and $j \le n$ )
   {
      if ( $mat ==$ 'S' )
         {
            if ( $i == m$ or $j == n$ or $S(i,j) == 0$ )
               break;
            if ( $S(i,j) == D(i,j)$ )
               { $mat$ = 'D'; continue; }
            if ( $S(i,j) == I(i,j)$ )
               { $mat$ = 'I'; continue; }
            append pair $(a_{i+1}, b_{j+1})$ to $OA$;
            $i++$; $j++$;
            continue;
         }
      if ( $mat ==$ 'D' )
         {
            append pair $(a_{i+1}, -)$ to $OA$;
            if ( $i == m-1$ or $D(i,j) == S(i+1,j) - q - r$ )
               $mat$ = 'S';
            $i++$; continue;
         }
      if ( $mat ==$ 'I' )
         {
            append pair $(-, b_{j+1})$ to $OA$;
            if ( $j == n-1$ or $I(i,j) == S(i,j+1) - q - r$ )
               $mat$ = 'S';
            $j++$; continue;
         }
   }
$rowlast = i$; $collast = j$;
```

Figure 3. Generation of an optimal local alignment.

The traceback procedure requires that the complete matrices be saved or additional information be saved to indicate how the value at each matrix entry is generated, which takes computer memory proportional to the product $m \times n$. Thus for two sequences of length 10,000, the algorithm takes computer memory in the order of 100,000,000 words. Because of the high computer memory requirement, only an optimal alignment of two sequences of at most a few thousand letters can be constructed on an ordinary computer using this algorithm. The time requirement of the algorithm is also proportional to the product $m \times n$. For two sequences of length 10,000, it takes less than a minute to compute the matrix S on an ordinary workstation. Thus the space requirement of this algorithm is much more limiting than the time requirement.