A Class Note on Local Alignment

Kun-Mao Chao[1,2,3]

[1]Graduate Institute of Biomedical Electronics and Bioinformatics
[2]Department of Computer Science and Information Engineering
[3]Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106

September 30, 2008

In many applications, a global (i.e., end-to-end) alignment of the two given sequences is inappropriate; instead, a local alignment (i.e., involving only a part of each sequence) is desired. In other words, one seeks a high-scoring local path that need not terminate at the corners of the dynamic-programming matrix.

Let $S[i,j]$ denote the score of the highest-scoring local path ending at (i,j) between $a_1a_2\ldots a_i$, and $b_1b_2\ldots b_j$. $S[i,j]$ can be computed as follows.

$$S[i,j] = \max \begin{cases} 0, \\ S[i-1,j]-\beta, \\ S[i,j-1]-\beta, \\ S[i-1,j-1]+\sigma(a_i,b_j). \end{cases}$$

The recurrence is quite similar to that for global alignment except the first entry "zero." For local alignment, we are not required to start from the source $(0,0)$. Therefore, if the scores of all possible paths ending at the current position are all negative, they are reset to zero. The largest value of $S[i,j]$ is the score of the best local alignment between sequences A and B.

Figure 1 gives the pseudo-code for computing the score of an optimal local alignment. Whenever there is a tie, any one of them will work. Since there are $O(mn)$ entries and the time spent for each entry is $O(1)$, the total running time of algorithm LOCAL_ALIGNMENT_SCORE is $O(mn)$.

```
Algorithm LOCAL_ALIGNMENT_SCORE($A = a_1a_2 \ldots a_m$, $B = b_1b_2 \ldots b_n$)
begin
    $S[0,0] \leftarrow 0$
    $Best \leftarrow 0$
    $End_i \leftarrow 0$
    $End_j \leftarrow 0$
    for $j \leftarrow 1$ to $n$ do $S[0,j] \leftarrow 0$
    for $i \leftarrow 1$ to $m$ do
        $S[i,0] \leftarrow 0$
        for $j \leftarrow 1$ to $n$ do
            $S[i,j] \leftarrow \max \begin{cases} 0 \\ S[i-1,j]-\beta \\ S[i,j-1]-\beta \\ S[i-1,j-1]+\sigma(a_i,b_j) \end{cases}$
            if $S[i,j] > Best$ then
                $Best \leftarrow S[i,j]$
                $End_i \leftarrow i$
                $End_j \leftarrow j$
    Output $Best$ as the score of an optimal local alignment.
end
```

Figure 1: Computation of the score of an optimal local alignment.

Now let us use an example to illustrate the tabular computation. Figure 2 computes the score of an optimal local alignment of the two sequences `ATACATGTCT` and `GTACGTCGG`, where a match is given a bonus score 8, a mismatch is penalized by a score -5, and the gap penalty for each gap symbol is -3. The first row and column of the table are initialized with zero's. Other entries are computed in order. Take the entry $(5,5)$ for example. Upon computing the value of this entry, the following values are ready: $S[4,4] = 24$, $S[4,5] = 21$, and $S[5,4] = 21$. Since the edge weight of $(4,4) \rightarrow (5,5)$ is -5 (a mismatch), the maximum score from $(4,4)$ to $(5,5)$ is $24-5 = 19$. The maximum score from $(4,5)$ is $21-3 = 18$, and the maximum score from $(5,4)$ is $21-3 = 18$. Taking the maximum of them, we have $S[5,5] = 19$. Once the table has been computed, the maximum value, *i.e.*, $S[9,7] = 42$, is the score of an optimal local alignment.

		G	T	A	C	G	T	C	G	G
	0	0	0	0	0	0	0	0	0	0
A	0	0	0	8	5	2	0	0	0	0
T	0	0	8	5	3	0	10	7	4	1
A	0	0	5	16	13	10	7	5	2	0
C	0	0	2	13	24	21	18	15	12	9
A	0	0	0	10	21	19	16	13	10	7
T	0	0	8	7	18	16	27	24	21	18
G	0	8	5	3	15	26	24	22	32	29
T	0	5	16	13	12	23	34	31	29	27
C	0	2	13	11	21	20	31	42	39	36
T	0	0	10	8	18	17	28	39	37	34

Figure 2: Computation of the score of an optimal local alignment of the sequences `ATACATGTCT` and `GTACGTCGG`.

Figure 3 lists the pseudo-code for delivering an optimal local alignment, where an initial call LOCAL_ALIGNMENT_OUTPUT(A, B, S, End_i, End_j) is made to deliver an optimal local alignment. Specifically, we trace back the dynamic-programming matrix from the maximum-score entry (End_i, End_j) recursively according to the following rules. Let (i, j) be the entry under consideration. If $S[i, j] = 0$, we have reached the beginning of the optimal local alignment. Otherwise, consider the following three cases. If $S[i, j] = S[i-1, j-1] + \sigma(a_i, b_j)$, we make a diagonal move and output a substitution pair $\begin{pmatrix} a_i \\ b_j \end{pmatrix}$. If $S[i, j] = S[i-1, j] - \beta$, then we make a vertical move and output a deletion pair $\begin{pmatrix} a_i \\ - \end{pmatrix}$. Otherwise, it must be the case where $S[i, j] = S[i, j-1] - \beta$. We simply make a horizontal move and output an insertion pair $\begin{pmatrix} - \\ b_j \end{pmatrix}$. Algorithm LOCAL_ALIGNMENT_OUTPUT takes $O(m+n)$ time in total since each recursive call reduces i and/or j by one. The space complexity is $O(mn)$ since the size of the dynamic-programming matrix is $O(mn)$. Later we shall show that an optimal local alignment can be recovered even if we don't save the whole matrix.

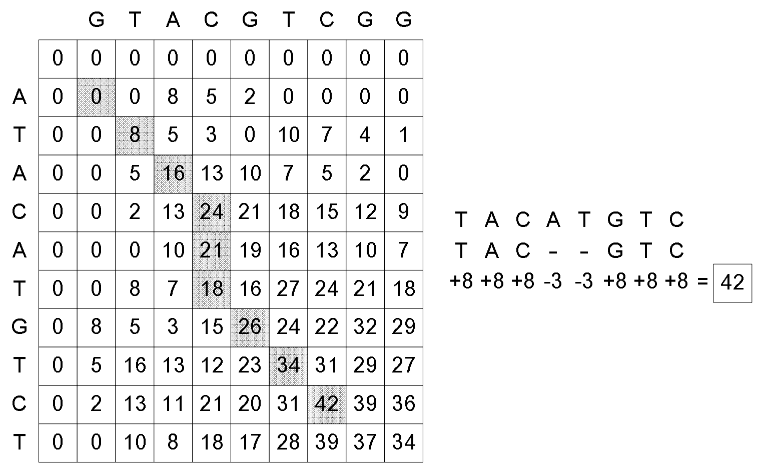
```
**Algorithm** LOCAL_ALIGNMENT_OUTPUT($A = a_1 a_2 \ldots a_m$, $B = b_1 b_2 \ldots b_n$, $S$, $i$, $j$)
**begin**
    **if** $S[i, j] = 0$ **then return**
    **if** $S[i, j] = S[i-1, j-1] + \sigma(a_i, b_j)$ **then**
        LOCAL_ALIGNMENT_OUTPUT($A$, $B$, $S$, $i-1$, $j-1$)
        **print** $\begin{pmatrix} a_i \\ b_j \end{pmatrix}$
    **else if** $S[i, j] = S[i-1, j] - \beta$ **then**
        LOCAL_ALIGNMENT_OUTPUT($A$, $B$, $S$, $i-1$, $j$)
        **print** $\begin{pmatrix} a_i \\ - \end{pmatrix}$
    **else**
        LOCAL_ALIGNMENT_OUTPUT($A$, $B$, $S$, $i$, $j-1$)
        **print** $\begin{pmatrix} - \\ b_j \end{pmatrix}$
**end**
```

Figure 3: Computation of an optimal local alignment.

Figure 4 delivers an optimal local alignment by backtracking from the maximum scoring entry of the dynamic-programming matrix computed in Figure **??**. We start from the entry $(9,7)$ where $S[9,7] = 42$. Since $S[8,6] + 8 = 34 + 8 = 42 = S[9,7]$, we make a diagonal move back to the entry $(8,6)$. Continue this process until an entry with zero value is reached. The shaded area depicts the backtracking path whose corresponding alignment is given on the right-hand side of the figure.

Further complications arise when one seeks k best alignments, where $k > 1$. For computing an arbitrary number of non-intersecting and high-scoring local alignments, Waterman and Eggert developed a very time-efficient method in 1987. It records those high-scoring candidate regions of the dynamic-programming matrix in the first pass. Each time a best alignment is reported, it recomputes only those entries in the affected area rather than recompute the whole matrix. Its linear-space implementation was developed by Huang and Miller in 1991.

On the other hand, to attain greater speed, the strategy of building alignments from alignment fragments is often used. For example, one could specify some fragment length w and work with fragments consisting of a segment of length at least *w* that occurs exactly or approximately in both sequences. In general, algorithms that optimize the score over alignments constructed from fragments can run faster than algorithms that optimize over all possible alignments. Moreover, alignments constructed from fragments have been very successful in initial filtering criteria within programs that search a sequence database for matches to a query sequence. Database sequences whose alignment score with the query sequence falls below a threshold are ignored, and the remaining sequences are subjected to a slower but higher-resolution alignment process. The high-resolution process can be made more efficient by restricting the search to a "neighborhood" of the alignment-from-fragments. Later we will introduce four such homology search programs: FASTA, BLAST, BLAT, and PatternHunter.



		G	T	A	C	G	T	C	G	G
	0	0	0	0	0	0	0	0	0	0
A	0	**0**	0	8	5	2	0	0	0	0
T	0	0	**8**	5	3	0	10	7	4	1
A	0	0	5	**16**	13	10	7	5	2	0
C	0	0	2	13	**24**	21	18	15	12	9
A	0	0	0	10	**21**	19	16	13	10	7
T	0	0	8	7	**18**	16	27	24	21	18
G	0	8	5	3	15	**26**	24	22	32	29
T	0	5	16	13	12	23	**34**	31	29	27
C	0	2	13	11	21	20	31	**42**	39	36
T	0	0	10	8	18	17	28	39	37	34

```
 T  A  C  A  T  G  T  C
 T  A  C  -  -  G  T  C
+8 +8 +8 -3 -3 +8 +8 +8 = 42
```

Figure 4: Computation of an optimal local alignment of the two sequences `ATACATGTCT` and `GTACGTCGG`.