

# 1 The Smith-Waterman Algorithm for local alignment

The notes you already have on local alignment are written more generally than what you need for lab 4. So here I am streamlining those notes to better suit the conditions of lab 4.

Then  $V(i, 0) = V(0, j) = 0$  for all  $i, j$ , since we can always choose an empty suffix of either string.

For any  $i > 0$  and  $j > 0$ , define  $t(i, j)$  to be 1 if  $S_1(i) = S_2(j)$  and to be -1 otherwise. Then the general recurrence for  $i > 0$  and  $j > 0$  is

$$V(i, j) = \max[0, V(i - 1, j - 1) + t(i, j), V(i - 1, j) - 1, V(i, j - 1) - 1].$$

You fill in the  $(n + 1) \times (m + 1)$  table of  $V$  values – setting all cells in row and column 0 to zero, and then filling in the rest of the cells by applying the general recurrence.

Then to find the value of the optimal local alignment of strings  $S_1$  and  $S_2$ , you search for the largest  $V$  value in the table.

That is all your program needs to do.

To actually find the optimal local alignment (not just its value) one starts at a cell with the maximum value and does a traceback (of the same kind as is done for global alignment) until a cell with value 0 is reached. Let's say that the maximum value is in cell  $(p, q)$  and the traceback ends at cell  $(i, j)$ . Then the two substrings that form the optimal local alignment are the substring of string  $S_1$  from position  $i + 1$  to position  $p$ , and the substring of string  $S_2$  from position  $j + 1$  to position  $q$ . You do not need to implement the traceback in your program. That will come later.

Note that the number of arithmetic and comparison operations is of the same order as that for global alignment. Hence it is practical to compute optimal local alignment for the same size strings that allow a practical global alignment.