

1 Multiple alignment with the sum-of-pairs (SP) objective function

The sum-of-pairs (SP) score

Definition The *sum of pairs (SP)* score of a multiple alignment \mathcal{M} is the *sum* of the scores of pairwise global alignments *induced* by \mathcal{M} . See Figure 1.

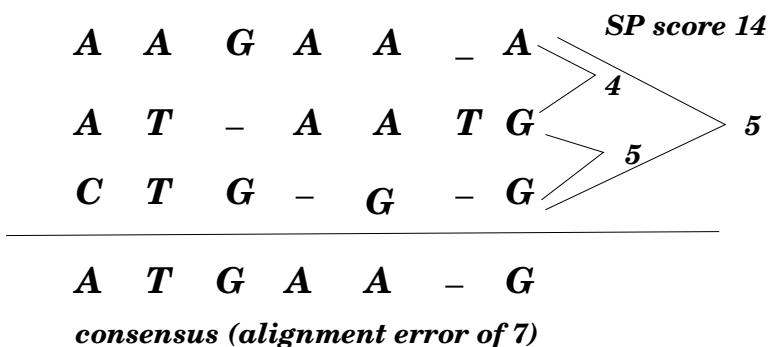


Figure 1: Multiple alignment \mathcal{M} of three strings shown above the horizontal line. Using the pairwise scoring scheme of $ms + id$, i.e., $\#(\text{mismatches}) + \#(\text{spaces opposite a non-space})$, the three induced pairwise alignments have scores of 4, 5, 5 for a total *SP* score of 14. Note that a space opposite a space contributes a zero. Using the plurality rule, the consensus string $S_{\mathcal{M}}$ (defined in Section ??) for alignment \mathcal{M} is shown below the horizontal line. It has an alignment error of seven.

Although one can give “handwaving” arguments for the significance of the *SP* score, it is difficult to give a theoretical justification for it (or any other multiple alignment scoring scheme). However, the *SP* score is easy to work with and has been used by many people studying multiple alignment. The *SP* score was first introduced in [?], and is also used in [?, ?, ?]. A similar score is used in [?]. The *SP* score is also used in a subtask in the multiple alignment package MACAW [?] developed at the National Institutes of Health, National Center for Biotechnology Information.

The SP alignment problem: Compute a global multiple alignment \mathcal{M} with *minimum* sum-of-pairs score.

1.1 An exact solution to the *SP* alignment problem

As might be expected, the *SP* problem can be solved exactly (optimally) via dynamic programming [?]. Unfortunately, if there are k strings and each is of length n say, dynamic programming takes $\Theta(n^k)$ time and hence is practical for only a small number of strings. Moreover, the exact *SP* alignment problem has been proven to be NP-complete [?]. So we will develop the dynamic programming recurrences only for the

case of three strings. Extension to any larger number of strings is straightforward, although impractical for even five strings whose lengths are in the low hundreds (typical of proteins). We will also develop an accelerant to the basic dynamic programming solution, somewhat increasing the number of strings that can be optimally aligned.

Definition Let S_1, S_2, S_3 denote three strings of lengths n_1, n_2 and n_3 respectively, and let $D(i, j, k)$ be the optimal *SP* score for aligning $S_1[1..i], S_2[1..j]$ and $S_3[1..k]$. The score for a match, mismatch or space is specified by the variables *smatch*, *smis*, *sspace* respectively.

The dynamic programming table D used to align three strings forms a three dimensional cube. Each cell (i, j, k) that is not on a boundary of the table (i.e. that has no index equal to zero) has seven neighbors that must be consulted to determine $D(i, j, k)$. The general recurrences for computing the cost of a non-boundary cell are similar in spirit to the recurrences for two strings, but are a bit more involved. The recurrences are encoded in the following pseudocode.

Recurrences for a non-boundary cell (i, j)

```

for  $i := 1$  to  $n_1$  do
  for  $j := 1$  to  $n_2$  do
    for  $k := 1$  to  $n_3$  do
      begin
        if  $(S_1(i) = S_2(j))$  then  $cij := smatch$ 
        else  $cij := smis$ ;
        if  $(S_1(i) = S_3(k))$  then  $cik := smatch$ 
        else  $cik := smis$ ;
        if  $(S_2(j) = S_3(k))$  then  $cjk := smatch$ 
        else  $cjk := smis$ ;

         $d1 := D(i-1, j-1, k-1) + cij + cik + cjk$ ;
         $d2 := D(i-1, j-1, k) + cij + 2*sspace$ ;
         $d3 := D(i-1, j, k-1) + cik + 2*sspace$ ;
         $d4 := D(i, j-1, k-1) + cjk + 2*sspace$ ;
         $d5 := D(i-1, j, k) + 2*sspace$ ;
         $d6 := D(i, j-1, k) + 2*sspace$ ;
         $d7 := D(i, j, k-1) + 2*sspace$ ;

         $D(i, j, k) := \text{Min}[d1, d2, d3, d4, d5, d6, d7]$ ;
      end;

```

What remains is to specify how to compute the D values for the boundary cells on the three initial faces of the table, i.e., when $i = 0$, or $j = 0$, or $k = 0$. To do

this, let $D_{1,2}(i, j)$ denote the familiar *pairwise* distance between substrings $S_1[1..i]$ and $S_2[1..j]$; and let $D_{1,3}(i, k)$ and $D_{2,3}(j, k)$ denote the analogous pairwise distances involving pairs S_1, S_3 and S_2, S_3 . These distances are computed in the standard way. Then,

$$\begin{aligned} D(i, j, 0) &= D_{1,2}(i, j) + (i + j)*sspace, \\ D(i, 0, k) &= D_{1,3}(i, k) + (i + k)*sspace, \\ D(0, j, k) &= D_{2,3}(j, k) + (j + k)*sspace, \\ \text{and } D(0, 0, 0) &= 0. \end{aligned}$$

The correctness of the recurrences and the fact that they can be evaluated in $O(n_1n_2n_3)$ time is left to the reader. The recurrences can be generalized to incorporate alphabet-weighted scores, and this is also left to the reader.