

## 14

### Multiple String Comparison – The Holy Grail

---

In this chapter we begin the discussion of *multiple string comparison*, one of the most important methodological issues and most active research areas in current biological sequence analysis. We first discuss some of the reasons for the importance of multiple string comparison in molecular biology. Then we will examine multiple string *alignment*, one common way that multiple string comparison has been formalized. We will precisely define three variants of the multiple alignment problem and consider in depth algorithms for attacking those problems. Other variants will be sketched in this chapter; additional multiple alignment issues will be discussed in Part IV.

#### 14.1. Why multiple string comparison?

For a computer scientist, the multiple string comparison problem may at first seem like a generalization for generalization's sake – “two strings good, four strings better”. But in the context of molecular biology, multiple string comparison (of DNA, RNA, or protein strings) is much more than a technical exercise. It is the most critical cutting-edge tool for *extracting and representing* biologically important, yet faint or widely dispersed, commonalities from a set of strings. These (faint) commonalities may reveal evolutionary history, critical conserved motifs or conserved characters in DNA or protein, common two- and three-dimensional molecular structure, or clues about the common biological function of the strings. Such commonalities are also used to characterize families or superfamilies of proteins. These characterizations are then used in database searches to identify other potential members of a family. Because many important commonalities are faint or widely dispersed, they might not be apparent when comparing two strings alone but may become clear, or even obvious, when comparing a set of related strings. As an example, see Figure 14.1.

One central technique for multiple string comparison involves *multiple alignment*. Although the main algorithmic issues involved in multiple alignment will be discussed later, we need here a clear definition in order to introduce some of its uses.

**Definition** A global multiple alignment of  $k > 2$  strings  $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$  is a natural generalization of alignment for two strings. Chosen spaces are inserted into (or at either end of) each of the  $k$  strings so that the resulting strings have the same length, defined to be  $l$ . Then the strings are arrayed in  $k$  rows of  $l$  columns each, so that each character and space of each string is in a unique column.

For example, see Figure 14.1 and 14.2.

#### Multiple comparison can be inverse to two-string comparison

In one sense, multiple string comparison is a natural extension of two-string comparison (via alignment), when one is looking for weakly conserved patterns rather than strongly

				*vvvvv*	
HUMA	VLSPADKTNVKAAGWKVGAHAGEYGAEALERMFLSFPTTKTYFPHF	DLSH	GS		
HAOR	MLTDAEKKEVTALWGKAAGHGEEYGAEALERLFQAFPTTKTYFSHF	DLSH	GS		
HADK	VLSAADKTNVKGVFSKIGGHAEYGAETLERMFIAYPQTKTYFPHF	DLSH	GS		
HBHU	VHLTPEEKSAVTALWGKV	NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGN			
HBOR	VHLSGGEKSAVTNLWGKV	NINELGGEALGRLLVVYPWTQRFFEFAGDLSSAGAVMGN			
HBDK	VHWTAEKQLITGLWGKV	NVADCGAEALARLLIVYPWTQRFFASFGNLSPTAILGN			
MYHU	GLSDGEWQLVLNVWGVKVEADIPGHGQEVLRIRLFKGHPEPTELEKFDKFKHLKSEDEMKAS				
MYOR	GLSDGEWQLVLNVWGVKVEADIPGHGQEVLRIRLFKTHPETLEKFDKFKGLKTEDEMKAS				
IGLOB	SPLTADAEASLVQSSWK	AVSHNEVEILAAVFAAYPDIQNKFSQFAIGKDLASIKDT			
GPUGNI	ALTEKQEALLKQSWEVLKQNI	PAHSLRFLALIEAAPESKYVFSFLKDSNEIPE	NN		
GPYL	GVLTDVQVALVKSSEEFNANI	PKNTHRFFTLVLEIAPGAKDLFSFLKGSSEVPQ	NN		
GGZLB	MLDQQTINIIKATVPVLKEHGVITITTTFYKNLFAKHPEVRPLF	DMGRQE	SL		
	vvvvv			vvvv*	
HUMA	AQVKGHGKKVADALTNAV	AHVDDM	PNALSALSDLHAHKLVRDPVNFKLLS		
HAOR	AQIKAHGKKVADALSTAA	GHFDDM	DSALSALSDLHAHKLVRDPVNFKLLA		
HADK	AQIKAHGKKVAAALVEAV	NHVDDI	AGALSCLSDLHAQKLVRDPVNFKFLG		
HBHU	PKVKAHGKKVLAGFSDGL	AHLDDL	KGTFAATLSELHCDKLHVDPENFRLLG		
HBOR	PKVKAHGAKVLTSGDAL	KNLDDL	KGTFAKLSELHCDKLHVDPENFNRLG		
HBDK	PMVRAHGKKVLTSGDAV	KNLDNI	KNTFAQLSELHCDKLHVDPENFRLLG		
MYHU	EDLKKHGATVLTALGGIL	KKKGHH	EAEIKPLAQSHATKHKIPVKYLEFIS		
MYOR	ADLKKHGGTVLTALGNIL	KKKGQH	EAEIKPLAQSHATKHKISIKFLEYIS		
IGLOB	GAFATHATRIVSFLSEVIALISGNTSNAAAV	NSLVSKLGGDDHKARGVSAQAQ1FGEFR			
GPUGNI	PKLKAHAIVFKTICESA	TELROKQHAWDNNLTKRLGSIH	LKNKITDPHFVEMK		
GPYL	PDLQAHAGKVFKLTYEAA	IQLEVNGAVASDATLKSLSGVHVSQGVVDA	HFPVVK		
GGZLB	EQPKALAMTVLAAAQNI	ENLPAI	LPAVKKIIVKHC	QAGVAAAHPYPIV	
	vvvvv	vvv			
HUMA	HCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTISKYR				
HAOR	HCILVVLARHCPGEFTPSAHAAMDKFLSKVATVLTISKYR				
HADK	HCFLVVVAIHHPAALTPEVHASLDKFMCVAVLTAKYR				
HBHU	NVLVLCVLAHFGKEFTPPVQAAYQKVVAGVANALAHKYH				
HBOR	NVLIVVLARHFSKDFSPVQAQWKLVSQVAHALGHKYH				
HBDK	DILIIIVLAAHFTKDFTEPCQAQWKLVRVVAHALARKYH				
MYHU	ECIIQVLQSKHPGDFGADAQAGAMNKALELFRKDMASNYKELGFQ				
MYOR	EAIIVHLQSKHSADFGADAQAAMGKALELFRNDMAAKYKEFGFQ				
IGLOB	TALVAYLQANVS	WGDNVAAWNKALIDNTFAIVVPR			
GPUGNI	GALLGTIKEAIKENWSEMGQAWTEAYNQLVATIKAEKME				
GPYL	EAILKTIKEVVGDKWSEELNTAWTIAYDELATIIKKEMKDA				
GGZLB	QELLGAIKEVLGDAATDDILDWGWKAYGVIAVFIQVEADLYAQAVE				

**Figure 14.1:** A multiple alignment of several amino acid sequences of globin proteins modified from the paper of McLure, Vasi, and Fitch [316]. The abbreviations on the left indicate the organisms that the globin sequences are from. Because of the length of the sequences, the multiple alignment is displayed using three lines for each sequence. Columns in the alignment containing a high concentration of similar residues in regions of known secondary structure are marked with a "v", and columns with identical residues are marked with a star. Two residues are considered similar if they belong to the same class of the following partition: (F,Y), (M,L,I,V), (A,G), (T,S), (Q,N), (K,R), and (E,D).

conserved patterns. Quoting from Arthur Lesk [233]: "One or two homologous sequences whisper . . . a full multiple alignment shouts out loud".

But in another sense, multiple string comparison is used in a fundamentally different way than two-string comparison. It can be used for biological inference problems that are *inverse* to problems addressed by two-string comparison. In the context of database searching, two-string alignment finds strings that have common subpatterns (substrings or subsequences) but may not have been known to be biologically related. Indeed, the

greatest value of database searching comes from using *apparent* string similarities to identify unsuspected biological relationships. The inverse problem is to move from known biological relationships to unknown substring or subsequence similarities. This direction uses multiple string comparison to deduce unknown conserved subpatterns from a set of strings already known to be biologically related.

#### 14.1.1. Biological basis for multiple string comparison

Recall the *first fact of biological sequence comparison* from Chapter 10 on page 212. This first fact underlies the effectiveness of two-sequence comparison and of biological database searching. But as powerful as the *first fact* is, it does not capture all the important biology forming the basis for biological sequence comparison. There is an additional fact that compels *multiple* string comparison.

**The second fact of biological sequence comparison** Evolutionarily and functionally related molecular strings can *differ significantly* throughout much of the string and yet preserve the same three-dimensional structure(s), or the same two-dimensional substructure(s) (motifs, domains), or the same active sites, or the same or related dispersed residues (DNA or amino acid).

This *second fact* was implicit in the earlier discussion of local alignment (Section 11.7). What is added here is a matter of degree. Two strings specifying the “same” protein in different species may be so different that the few observed similarities may just be due to chance. Typically, secondary or three-dimensional structure is the most well preserved feature of a large set of evolutionarily related proteins. Conservation of function is less common, and conservation of amino-acid sequence is the least common. “One of the most intriguing observations to arise from recent analyses of protein structures is that similar folds recur, even in the absence of sequence similarity” [428]. (This point was also made by the quote from Cohen on page 214.)

For example, hemoglobin is a nearly universal protein containing four chains of about 140 amino acids each. It is contained in organisms as diverse as mammals and insects, and it functions in essentially the same way (it binds and transports oxygen). However, in the time since insects and invertebrates diverged (some 600 million years ago), multiple amino acid substitutions have occurred in insect and invertebrate hemoglobin sequences, so that on average there have been about 100 amino acid mutations in each chain of the two sequences<sup>1</sup>[11]. A pairwise alignment of hemoglobin chains from two mammals will suggest a functional similarity of the protein (human and chimpanzee sequences are in fact identical), but a two-string alignment from a mammal and an insect may reveal very little. Moreover, many proteins mutate faster than hemoglobin, compounding the problem.

As another, more recent example, the three-dimensional structure of part of a *cell adhesion molecule* called *E-Cadherin* was worked out [354, 451]. The deduced structure was unexpectedly found to show “remarkable similarity to the Ig fold (although there is no sequence homology)” [451]. The Ig fold is a common structure first seen in the immunoglobulin (antibody) superfamily but later seen in many other proteins as well. The term “homology” here should be interpreted as “similarity”, and “no homology” should be interpreted as “no more similarity than expected between random strings”. So the

<sup>1</sup> This doesn’t translate immediately into the number of positions where amino acid differences occur between mammal and insect hemoglobin because more than one mutation can occur at a given position.

fold found is a conserved physical structure relating the Ig proteins with the E-Cadherin protein, despite the lack of sequence similarity. A related discussion in [451] about a different protein (the growth hormone receptor) illustrates the same point: “Previously, the growth hormone receptor structure was found to be built of domains related to this (Ig) evolutionarily successful motif; in this case too, no sequence homology exists”.

The ability of many proteins to preserve structure and function in the face of massive amino acid substitution is simply an empirical fact, although it may seem counterintuitive. One might intuitively expect<sup>2</sup> that every detail of an amino acid string or its encoding DNA is essential, so that even small changes to the string would destroy the structure or function of the protein. Indeed, many gene-related diseases are caused by such small mutations. Often even a single change at a *particular* amino acid position, which might be caused by a single DNA nucleotide change, leads to a serious disease such as sickle-cell anemia, cancer (the *ras* oncogene, for example, is activated by a single nucleotide point mutation [403, 110]), or a prion disease such as *Mad Cow Disease* [377]. But on whole, many amino acid positions in a protein string are noncritical and over evolutionary history have mutated to a wide selection of other amino acids without a destructive effect on the protein. So, although the *first fact* of biological sequence analysis, that sequence similarity implies structural or functional similarity, is very powerful and widely exploited, the converse of that fact simply is not true.

The extent of permissive mutations in structurally or functionally conserved molecules may be such that comparing two strings at a time reveals little of the critically conserved patterns or of the critical amino acids. There may be so few conserved amino acids or they may be so widely dispersed that the best alignment between two related protein strings is statistically indistinguishable from the best alignment of two random amino acid strings. The problem occurs at the other extreme as well. Comparison of two proteins from highly related species might not reveal biologically important conserved patterns because the critical similarities are lost in the large number of similarities due only to recent shared evolution. Thus when doing two-string comparison to find critical common patterns, the challenge is to pick species whose level of divergence is “most informative” (a vague and difficult task).

Multiple string comparison (often via multiple alignment) is a natural response to the problems introduced above. With multiple string comparison, it is not as crucial (although it is still helpful) to pick species whose level of divergence is “most informative”. Often, biologically important patterns that cannot be revealed by comparison of two strings alone become clear when many related strings are simultaneously compared. Moreover, with multiple alignments, it is sometimes possible to arrange a set of related strings (often in a tree) to demonstrate *continuous* changes along a path connecting two extreme strings that by themselves show little pairwise similarity. In particular, multiple alignments of this kind are very useful in deducing evolutionary history.

### 14.2. Three “big-picture” biological uses for multiple string comparison

Multiple sequence comparison is a critical task in three kinds of inquiries that address “big-picture” biological questions: the representation of protein families and superfamilies, the identification and representation of conserved sequence features of DNA or protein that

<sup>2</sup> And some early papers on protein evolution even state this expectation as a “fact”. For example, see [121].

the extent that character changes in column  $i$  correlate with character changes in column  $j$  (we will again be vague in the precise definition of this correlation). For example, if half the sequences have an A in column 16 and a U in column 92, while the other half have a C in 16 and a G in 92, then columns 16 and 92 would be considered highly correlated. The explanation for high correlation in this example is that the two nucleotides occupying positions 16 and 92 (defined by the alignment) in the *tRNA* are base-paired with each other, even though the specific nucleotides in those positions may mutate and be different in different sequences. The point is that in order to preserve structural integrity, point mutations in paired nucleotides must generally be correlated.<sup>9</sup> At the other extreme, if changes in column 16 were totally independent of changes in column 92, then one could deduce that the nucleotides in those two positions were probably not base-paired. In the first situation, the weight of edge (16, 92) in the graph should be set high, whereas in the second situation, the edge weight should be set low. Of course, these two extreme cases are for illustration only; most correlation data will be more ambiguous, which is why a weighted matching computation is needed to decide on the overall nucleotide pairing.

We have not been precise on how pairwise column correlation is translated into edge weights [89], but we have presented the essential intuition. In summary, multiple string comparison (alignment) of *tRNA* sequences seems to expose enough information about *tRNA* secondary structure so that nucleotide pairing based on general weighted matching is “fairly” successful in recovering the true secondary structure from sequence information alone.

## 14.5. Introduction to computing multiple string alignments

Having motivated multiple string comparison with the preceding discussions, we now move to the purely technical issues of how to compute *multiple string alignment*. The definition of *global* multiple alignment was given on page 332, and examples are shown in Figures 14.2 and 14.6b. It is also natural to define *local* multiple alignment. Two-string local alignment was defined as a global alignment of substrings, and multiple local alignment can be similarly defined.

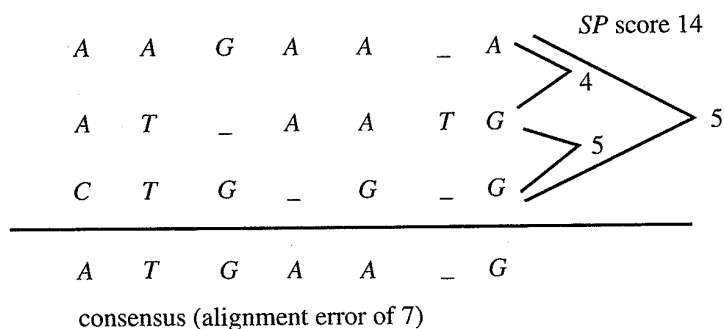
**Definition** Given a set of  $k > 2$  strings  $S = \{S_1, S_2, \dots, S_k\}$ , a *local* multiple alignment of  $S$  is obtained by selecting one substring  $S'_i$  from each string  $S_i \in S$  and then globally aligning those substrings.

All of the biological justifications for preferring local alignment to global alignment of two strings (see Section 11.7) also apply to local versus global multiple alignment. These justifications are further strengthened by the *second fact* of biological sequence analysis, discussed above. However, the best theoretical results (from a computer science perspective) on multiple alignment have been obtained for global alignment, and so our emphasis will be on global alignment. We will briefly discuss local multiple alignment in Section 14.10.3.

### 14.5.1. How to score multiple alignments

Although the notion of a multiple alignment is easily extended from two strings to many strings, the *score* or goodness of a multiple alignment is not as easily generalized. To date,

<sup>9</sup> However, this explanation does not cover all the cases of high correlation, for there are pairs of positions with high correlation that do not correspond to base-paired nucleotides in the *tRNA*. Also, positions at which the nucleotide does not mutate are not well handled by this method.



**Figure 14.5:** Multiple alignment  $\mathcal{M}$  of three strings shown above the horizontal line. Using the pairwise scoring scheme of  $ms + id$ , i.e.,  $\#(\text{mismatches}) + \#(\text{spaces opposite a nonspace})$ , the three induced pairwise alignments have scores of 4, 5, and 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 14.7.2) for alignment  $\mathcal{M}$  is shown below the horizontal line. It has an alignment error of seven.

there is no objective function that has been as well accepted for multiple alignment as (weighted) edit distance or similarity has been for two-string alignment. In fact, as we will see later, some popular methods to find multiple alignments do so without using any explicit objective function. The goodness of those methods is judged by the biological meaning of the alignments that they produce, and so the biological insight of the evaluator is of critical importance.

Not being biologists, we will emphasize multiple alignment methods that *are* guided by explicit objective functions, although we will sketch other methods as well. We will discuss three types of objective functions in this chapter: *sum-of-pairs functions*, *consensus functions*, and *tree functions*. Although these objective functions are quite different, we will explore approximation algorithms for these problems that are highly related. In Part IV of the book we will discuss how multiple alignment problems interact with problems of deducing trees representing evolutionary history.

**Definition** Given a multiple alignment  $\mathcal{M}$ , the *induced pairwise alignment* of two strings  $S_i$  and  $S_j$  is obtained from  $\mathcal{M}$  by removing all rows except the two rows for  $S_i$  and  $S_j$ . That is, the induced alignment is the multiple alignment  $\mathcal{M}$  restricted to  $S_i$  and  $S_j$ . Any two opposing spaces in that induced alignment can be removed if desired.

**Definition** The *score* of an induced pairwise alignment is determined using any chosen scoring scheme for two-string alignment in the standard manner.

For an example, see Figure 14.5.

Notice that the definition of “score” does not specify whether a score is a weighted distance or a similarity. As before, similarity is more natural for treating local alignment. Most of this chapter concerns global alignment, where the algorithms and consequent theorems all require the score to be a weighted distance.

## 14.6. 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 14.5.

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 [88], and was subsequently used in [33, 334], and [187]. A similar score is used in [153]. The *SP* score is also used in a subtask in the multiple alignment package MACAW [398] developed at the National Institutes of Health (NIH), National Center for Biotechnology Information.

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

### 14.6.1. An exact solution to the *SP* alignment problem

As might be expected, the *SP* problem can be solved exactly (optimally) via dynamic programming [334]. 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 [454]. We will therefore develop the dynamic programming recurrences only for the case of three strings. Extension to any larger number of strings is straightforward, although it proves to be 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 that somewhat increases the number of strings that can be optimally aligned.

**Definition** Let  $S_1$ ,  $S_2$ , and  $S_3$  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*, and *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 nonboundary 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 nonboundary 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  $c_{ij} := \text{smatch}$ 
        else  $c_{ij} := \text{smis}$ ;
        if  $(S_1(i) = S_3(k))$  then  $c_{ik} := \text{smatch}$ 
        else  $c_{ik} := \text{smis}$ ;
        if  $(S_2(j) = S_3(k))$  then  $c_{jk} := \text{smatch}$ 
        else  $c_{jk} := \text{smis}$ ;

         $d1 := D(i-1, j-1, k-1) + c_{ij} + c_{ik} + c_{jk}$ ;
         $d2 := D(i-1, j-1, k) + c_{ij} + 2 * \text{sspace}$ ;
         $d3 := D(i-1, j, k-1) + c_{ik} + 2 * \text{sspace}$ ;

```

```

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) := 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  $i = 0$ ,  $j = 0$ , 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,

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

and

$$D(0, 0, 0) = 0.$$

The correctness of the recurrences and the fact that they can be evaluated in  $O(n_1 n_2 n_3)$  time is left to the reader. The recurrences can be generalized to incorporate alphabet-weighted scores, and this is also left to the reader.

### A speedup for the exact solution

Carillo and Lipman [88] suggested a way to reduce some of the work needed in computing the optimal  $SP$  multiple alignment in practice. An extension of their idea was implemented in the program called MSA [303], but that extension was not described in the paper. Here we will explain the idea in the MSA version of the speedup. Additional refinements (particularly space reductions) to MSA are reported in [197]. We again restrict attention to aligning three strings.

The program for multiple alignment shown in the previous section is an example of using recurrences in a *backward* direction. At the time that the algorithm sets  $D(i, j, k)$ , the program looks backward to retrieve the  $D$  values of the seven (at most) cells that influence the value of  $D(i, j, k)$ .

In the alternative approach of forward dynamic programming (see also Section 12.6.1), when  $D(i, j, k)$  is set,  $D(i, j, k)$  is then sent *forward* to the seven (at most) cells whose  $D$  value can be influenced by cell  $(i, j, k)$ . Another way to say this is to view optimal alignment as a shortest path problem in the weighted edit graph corresponding to a multiple alignment table. In that view, when the shortest path from the source  $s$  (cell  $(0, 0, 0)$ ) to a node  $v$  (cell  $(i, j, k)$ ) has been computed, the best-yet distance from  $s$  to any out-neighbor of  $v$  is then updated. In more detail, let  $D(v)$  be the shortest distance from  $s$  to  $v$ , let  $(v, w)$  be a directed edge of weight  $p(v, w)$ , and let  $p(w)$  be the shortest distance yet found from  $s$  to  $w$ . After  $D(v)$  is computed,  $p(w)$  is immediately updated to be  $\min[p(w), D(v) + p(v, w)]$ . Moreover, the true shortest distance from  $s$  to  $w$ ,  $D(w)$ , must be  $p(w)$  after  $p(w)$  has been updated from every node  $v$  having an edge pointing into  $w$ . One more detail is needed. The forward dynamic programming implementation will keep a *queue* of nodes (cells)



whose final  $D$  value has not yet been set. The algorithm will set the  $D$  value of the node  $v$  at the head of the queue and remove that node. When it does, it updates  $p(w)$  for each out-neighbor of  $v$ , and if  $w$  is not yet in the queue, it adds  $w$  to the end of the queue. It is easy to verify that when a node comes to the head of the queue, all the nodes that point to it in the graph have already been removed from the queue.

For the problem of aligning three strings of  $n$  characters each, the edit graph has roughly  $n^3$  nodes and  $7n^3$  edges, and the backward dynamic programming computation will do some work for each of these edges. However, if one can identify a large number of nodes as not possibly being on any optimal (shortest) path from  $(0, 0, 0)$  to  $(n, n, n)$ , then work can be avoided for each edge out of those excluded nodes. The Carillo and Lipman speedup excludes some nodes before the main dynamic programming computation is begun, and forwards or backwards dynamic programming is equally convenient. In the extension used in MSA, nodes are excluded *during* the main algorithm, and forward dynamic programming is critical.

**Definition** Let  $d_{1,2}(i, j)$  be the edit distance between suffixes  $S_1[i..n]$  and  $S_2[j..n]$  of strings  $S_1$  and  $S_2$ . Define  $d_{1,3}(i, k)$  and  $d_{2,3}(j, k)$  analogously.

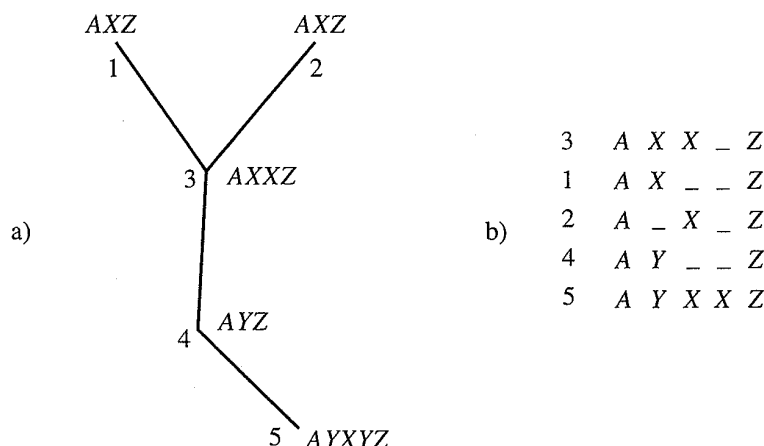
Certainly, all these  $d$  values can be computed in  $O(n^2)$  time by reversing the strings and computing three pairwise distances. Also, the shortest path from node  $(i, j, k)$  to node  $(n, n, n)$  in the edit graph for  $S_1$ ,  $S_2$ , and  $S_3$  must have distance at least  $d_{1,2}(i, j) + d_{1,3}(i, k) + d_{2,3}(j, k)$ .

Now suppose that some multiple alignment of  $S_1$ ,  $S_2$ , and  $S_3$  is known (perhaps from a bounded-error heuristic of the type to be discussed later) and that the alignment has  $SP$  score of  $z$ . The idea of the heuristic speedup is:

**Key idea** Recall that  $D(i, j, k)$  is the optimal  $SP$  score for aligning  $S_1[1..i]$ ,  $S_2[1..j]$ , and  $S_3[1..k]$ . If  $D(i, j, k) + d_{1,2}(i, j) + d_{1,3}(i, k) + d_{2,3}(j, k)$  is greater than  $z$ , then node  $(i, j, k)$  cannot be on any optimal path and so (in a forward computation)  $D(i, j, k)$  need not be sent forward to any cell.

When this heuristic idea applies to a cell  $(i, j, k)$ , it saves the work of not sending a  $D$  value forward to the (seven) out-neighbors of cell  $(i, j, k)$ . But more important than the speedup observed at cell  $(i, j, k)$  is that when the heuristic applies at many cells, additional cells downstream may be automatically excluded from consideration because they never get placed in the queue. In this way, the heuristic may prune off a large part of the alignment graph and dramatically reduce the number of cells whose  $D$  value is set when determining  $D(n, n, n)$ . This is a standard heuristic in shortest path computations. Notice that the computation remains exact and will find the optimal alignment.

If no initial  $z$  value is known, then the heuristic can be implemented as an  $A^*$  heuristic [230, 379] so that the most “promising” alignments are computed first. In that approach, during the running of the algorithm the best alignment computed so far provides the (falling)  $z$  value. It can be proven [230] that this approach prunes out as much of the alignment graph as the Carillo and Lipman method does, but usually it will exclude much more. It is reported in [303] that MSA can align six strings of lengths around 200 characters in a “practical” amount of time. It doesn’t seem likely, however, that this approach will make it practical to optimally align tens or hundreds of strings, unless one begins with an extremely good value for  $z$ . Even then, it is not clear that MSA would be practical.



**Figure 14.6:** a. A tree with its nodes labeled by a (multi)set of strings. b. A multiple alignment of those strings that is consistent with the tree. The pairwise scoring scheme scores a zero for each match and a one for each mismatch or space opposite a character. The reader can verify that each of the four induced alignments specified by an edge of the tree has a score equal to its respective optimal distance. However, the induced alignment of two strings which do not label adjacent nodes may have a score greater than their optimal pairwise distance.

### 14.6.2. A bounded-error approximation method for *SP* alignment

Because the exact solution to the *SP* alignment problem is feasible for only a small number of strings, most practical (heuristic) multiple alignment methods do not insist on finding the optimal *SP* alignment. However, little is usually known about how much the alignments produced by those heuristics can deviate from the optimal *SP* alignment. In contrast, in this section we discuss one of the few methods where such error analysis has been possible. We develop here a *bounded-error approximation* method, from [201], for *SP* alignment. That is, the method is provably fast (runs in polynomial worst-case time) and yet produces alignments whose *SP* score is guaranteed to be less than twice the score of the optimal *SP* alignment. This will be the first of several bounded-error approximation methods presented in this book, which will include additional methods for different multiple alignment problems.

#### An initial key idea: alignments consistent with a tree

The *SP* approximation we present, the improvements of it, and several other methods for other problems all use a key idea that relates multiple alignments to trees.<sup>10</sup> We develop that idea in general before continuing with *SP* alignment in particular. Recall that for two strings,  $D(S_i, S_j)$  is the (optimal) weighted edit distance between  $S_i$  and  $S_j$ .

**Definition** Let  $S$  be a set of strings, and let  $T$  be a tree where each node is labeled with a distinct string from  $S$ . Then, a multiple alignment  $\mathcal{M}$  of  $S$  is called *consistent* with  $T$  if the induced pairwise alignment of  $S_i$  and  $S_j$  has score  $D(S_i, S_j)$  for each pair of strings  $(S_i, S_j)$  that label adjacent nodes in  $T$ . For an example, see Figure 14.6.

**Theorem 14.6.1.** *For any set of strings  $S$  and for any tree  $T$  whose nodes are labeled by distinct strings of  $S$ , we can efficiently find a multiple alignment  $\mathcal{M}(T)$  of  $S$  that is consistent with  $T$ .*

<sup>10</sup> This connection between trees and multiple alignments should not be confused with the connection between evolutionary trees and multiple alignments, to be discussed in Sections 14.8 and 14.10.2.

Note that the role of  $T$  in this theorem is very special and the induced alignment of two strings  $S_i$  and  $S_j$  that do not label adjacent nodes will generally have a score greater than  $D(S_i, S_j)$ .

**PROOF OF THEOREM 14.6.1** We will construct the multiple alignment  $\mathcal{M}(T)$  one string at a time and show inductively that Theorem 14.6.1 holds after each new string is added to the alignment. To start, choose any two strings  $S_i$  and  $S_j$  that label adjacent nodes in  $T$  and form a two-string alignment of  $S_i$  and  $S_j$  with distance  $D(S_i, S_j)$ . The theorem trivially holds at this point. Assume that the theorem holds after some arbitrary number of strings have been added to the multiple alignment. To continue, select any string  $S'$  not yet included in the alignment such that  $S'$  labels a node adjacent to a node whose label,  $S_i$  say, is already in the alignment. In that existing multiple alignment, some spaces may have been inserted into  $S_i$ , and we use  $\bar{S}_i$  to denote the string  $S_i$  with those spaces included. Note that for every string in the existing multiple alignment, each character of that string is in a distinct column with exactly one character of  $\bar{S}_i$ .

Next, optimally align string  $S'$  with  $\bar{S}_i$  using a scoring scheme for two-string alignment, with the added rule that two opposing spaces have zero cost (they are considered a match). It is immediately apparent that the score of that resulting pairwise alignment is exactly  $D(S_i, S')$ . Now we will add  $S'$  to the existing multiple alignment so that the induced alignment of  $S_i$  and  $S'$  has score  $D(S_i, S')$  and so that all the induced scores of the strings already in the alignment remain unchanged. Let  $\bar{S}'$  be the string  $S'$  with any spaces inserted by the alignment of  $S'$  and  $\bar{S}_i$ . If the pairwise alignment of  $S'$  and  $\bar{S}_i$  does not insert any new spaces into  $\bar{S}_i$ , then append  $\bar{S}'$  to the existing multiple alignment. The result is a multiple alignment with one more string, where the induced score of  $S_i$  and  $S'$  is  $D(S_i, S')$  and where all the induced scores from the previous multiple alignment remain unchanged.

However, if the pairwise alignment does insert a new space in  $\bar{S}_i$  between characters  $l$  and  $l+1$ , say, then insert a space between characters  $l$  and  $l+1$  in every string in the *existing* multiple alignment. This will create new column(s) in the existing multiple alignment in which every character is a space, but otherwise retains the columns of the existing multiple alignment. (For example, suppose that the first four strings from Figure 14.6 have been multiply aligned and that  $\bar{S}_4$  at that point is  $AY\_Z$ . The alignment of  $S_5$  to  $\bar{S}_4$  inserts an additional space into  $\bar{S}_4$  at the fourth position. The first four rows of Figure 14.6 show the resulting multiple alignment after that space is replicated in strings  $S_1$ ,  $S_2$ , and  $S_3$ .) The result of adding a column containing only spaces is a multiple alignment in which the score of all pairwise induced alignments is the same as before any spaces were inserted. Then appending  $\bar{S}'$  to that multiple alignment creates a multiple alignment of one more string, where the statement of the theorem holds. The existence of  $\mathcal{M}(T)$  then follows by induction.

The time needed to compute  $\mathcal{M}(T)$  is dominated by the time to compute  $k-1$  pairwise alignments. If each string has length  $n$ , then each pairwise alignment takes time  $O(n^2)$  and the time to construct  $\mathcal{M}(T)$  is  $O(kn^2)$ .  $\square$

Although Theorem 14.6.1 has become a part of “folklore”, it may have been first explicitly stated in [153]. We can now return to the approximation method for the *SP* alignment problem.

### The center star method for *SP* alignment

We will describe the method in terms of an alphabet-weighted scoring scheme for two-string alignment, and let  $s(x, y)$  be the score contributed when a character  $x$  (possibly a space) is aligned opposite a character  $y$  (possibly a space).

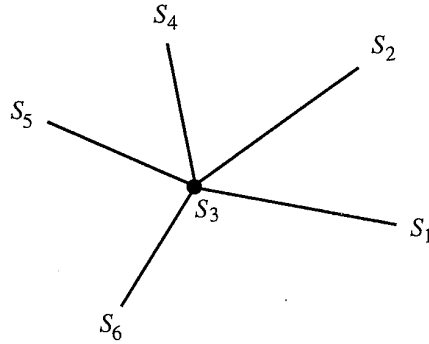


Figure 14.7: A generic center star for six strings, where the center string  $S_c$  is  $S_3$ .

**Definition** A scoring scheme satisfies the *triangle inequality* if for any three characters  $x$ ,  $y$ , and  $z$ ,  $s(x, z) \leq s(x, y) + s(y, z)$ .

Triangle inequality makes good intuitive sense in the context of edit distance. It says that the “cost” of transforming  $x$  to  $z$  directly is no more than the cost of first transforming  $x$  to an intermediate character  $y$ , and then transforming  $y$  to  $z$ . It should be noted, however, that not all scoring matrices in use in computational biology satisfy the triangle inequality.

**Definition** Given a set of  $k$  strings  $\mathcal{S}$ , define a *center string*  $S_c \in \mathcal{S}$  as a string in  $\mathcal{S}$  that minimizes  $\sum_{S_j \in \mathcal{S}} D(S_c, S_j)$ , and let  $M$  denote that minimum sum. Define the *center star* to be a star tree of  $k$  nodes, with the center node labeled  $S_c$  and with each of the  $k - 1$  remaining nodes labeled by a distinct string in  $\mathcal{S} - S_c$ . For an example, see Figure 14.7.

**Definition** Define the multiple alignment  $\mathcal{M}_c$  of the set of strings  $\mathcal{S}$  to be the multiple alignment *consistent* with the center star.

**Definition** Define  $d(S_i, S_j)$  as the score of the pairwise alignment of strings  $S_i$  and  $S_j$  induced by  $\mathcal{M}_c$ . Denote the score of an alignment  $\mathcal{M}$  as  $d(\mathcal{M})$ .

Clearly,  $d(S_i, S_j) \geq D(S_i, S_j)$ , and  $d(\mathcal{M}_c) = \sum_{i < j} d(S_i, S_j)$ . Also, since  $\mathcal{M}_c$  is consistent with the star tree, and string  $S_c$  is at the center of the star,  $d(S_i, S_c) = D(S_i, S_c)$  for each string  $S_i$ . We will show that  $d(\mathcal{M}_c)$  is at most twice the score of the optimal *SP* multiple alignment of  $\mathcal{S}$ , provided that the scoring scheme used for pairwise alignment satisfies the triangle inequality.

**Lemma 14.6.1.** Assume that the two-string scoring scheme satisfies the triangle inequality. Then, for any strings  $S_i$  and  $S_j$  in  $\mathcal{S}$ ,  $d(S_i, S_j) \leq d(S_i, S_c) + d(S_c, S_j) = D(S_i, S_c) + D(S_c, S_j)$ .

**PROOF** Consider any single column in the multiple alignment  $\mathcal{M}_c$ , and let  $x$ ,  $y$ , and  $z$  be the three characters in this column from the three strings  $S_i$ ,  $S_c$ , and  $S_j$ , respectively. By the triangle inequality,  $s(x, z) \leq s(x, y) + s(y, z)$ , and so the claimed inequality follows by the definition of  $d$ . The claimed equality follows because the pairwise alignment of  $S_i$  and  $S_c$  induced by  $\mathcal{M}_c$  is an optimal alignment of  $S_i$  and  $S_c$ , and this is true also for the alignment of  $S_c$  and  $S_j$ .  $\square$

**Definition** Let  $\mathcal{M}^*$  be the optimal multiple alignment of the  $k$  strings of  $\mathcal{S}$ . Let  $d^*(S_i, S_j)$  be the score of the pairwise alignment of strings  $S_i$  and  $S_j$  induced by  $\mathcal{M}^*$ . Then  $d(\mathcal{M}^*) = \sum_{i < j} d^*(S_i, S_j)$ .

We can now state and prove the main theorem of this section.

**Theorem 14.6.2.**  $d(\mathcal{M}_c)/d(\mathcal{M}^*) \leq 2(k - 1)/k < 2$ .

**PROOF** First, define  $v(\mathcal{M}_c) \equiv \sum_{(i,j)} d(S_i, S_j)$  and  $v(\mathcal{M}^*) \equiv \sum_{(i,j)} d^*(S_i, S_j)$ , where the pair  $(i, j)$  is an *ordered* pair in each case. Clearly,  $v(\mathcal{M}_c) = 2d(\mathcal{M}_c)$  and  $v(\mathcal{M}^*) = 2d(\mathcal{M}^*)$ , and so the ratios  $d(\mathcal{M}_c)/d(\mathcal{M}^*)$  and  $v(\mathcal{M}_c)/v(\mathcal{M}^*)$  are equal. It is more convenient to work with the second ratio. Recall that the minimum sum of distances,  $M$ , is defined as  $\sum_j D(S_c, S_j)$ . Now  $v(\mathcal{M}_c) = \sum_{(i,j)} d(S_i, S_j) \leq \sum_{(i,j)} [D(S_i, S_c) + D(S_c, S_j)]$ , by Lemma 14.6.1. For any fixed  $j$ ,  $D(S_c, S_j)$  (which equals  $D(S_j, S_c)$ ) shows up in this expression exactly  $2(k-1)$  times. So  $v(\mathcal{M}_c) \leq 2(k-1) \times \sum_j D(S_c, S_j) = 2(k-1)M$ .

From the other side,  $v(\mathcal{M}^*) = \sum_{(i,j)} d^*(S_i, S_j) \geq \sum_{(i,j)} D(S_i, S_j) = \sum_i \sum_j D(S_i, S_j) \geq k \times \sum_j D(S_c, S_j) = kM$  (by the choice of  $S_c$ ). So  $d(\mathcal{M}_c)/d(\mathcal{M}^*) = v(\mathcal{M}_c)/v(\mathcal{M}^*) \leq 2(k-1)M/kM = 2(k-1)/k = 2 - 2/k < 2$ .  $\square$

Note that for  $k = 3$  the guaranteed upper bound is  $4/3$ . That is, for three strings, the multiple alignment produced by the center star method will never be more than 34% more than the optimal *SP* score. Translated into lower bounds this says that for  $k = 3$ ,  $d(\mathcal{M}^*) \geq .75d(\mathcal{M}_c)$ . For  $k = 4$  the upper bound is only 1.5, and for  $k = 6$  (a problem size considered to be too large for efficient exact solution with strings of length 200) the bound is still only 1.67.

**Corollary 14.6.1.**

$$kM \leq \sum_{i < j} D(S_i, S_j) \leq d(\mathcal{M}^*) \leq d(\mathcal{M}_c) \leq [2(k-1)/k] \sum_{i < j} D(S_i, S_j).$$

In practice one can better measure the goodness of  $\mathcal{M}_c$  by the ratio  $d(\mathcal{M}_c)/\sum_{i < j} D(S_i, S_j)$ . By Corollary 14.6.1 this ratio is always less than two, but the analysis used there is worst case, so one can expect the ratio to often be considerably less than two. Similarly, one should expect that  $d(\mathcal{M}_c)/d(\mathcal{M}^*)$  will often be considerably less than two, since typically  $\sum_{(i,j)} D(S_i, S_j)$  will be considerably larger than  $kM$ ; that  $d(\mathcal{M}^*)$  will not generally be close to  $\sum_{i < j} D(S_i, S_j)$  for any strings except those that are very similar; and that  $D(S_i, S_j)$  will be less than  $D(S_i, S_c) + D(S_c, S_j)$  for most typical strings.

Corollary 14.6.1 is also useful in the MSA speedup discussed in Section 14.6.1 for the exact solution to *SP* alignment, since that method requires knowing an efficiently computed upper bound  $z$  on the optimal *SP* alignment score.