Algorithms for Molecular Biology **Fall Semester, 2001**

Lecture 4: 6 December, 2001

Lecturer: Ron Shamir *Scribe: Oks Eduard and Shrayber Yevgeny*[1]

4.1 Multiple Alignment

Definition A *multiple alignment* of strings $S_1, S_2, \ldots, S_k$ is a series of strings with spaces $S'_1, S'_2, \ldots, S'_k$ such that

1. $|S'_1| = |S'_2| = \ldots = |S'_k|$.
2. S'_j is an extension of S_j, obtained by insertions of spaces.

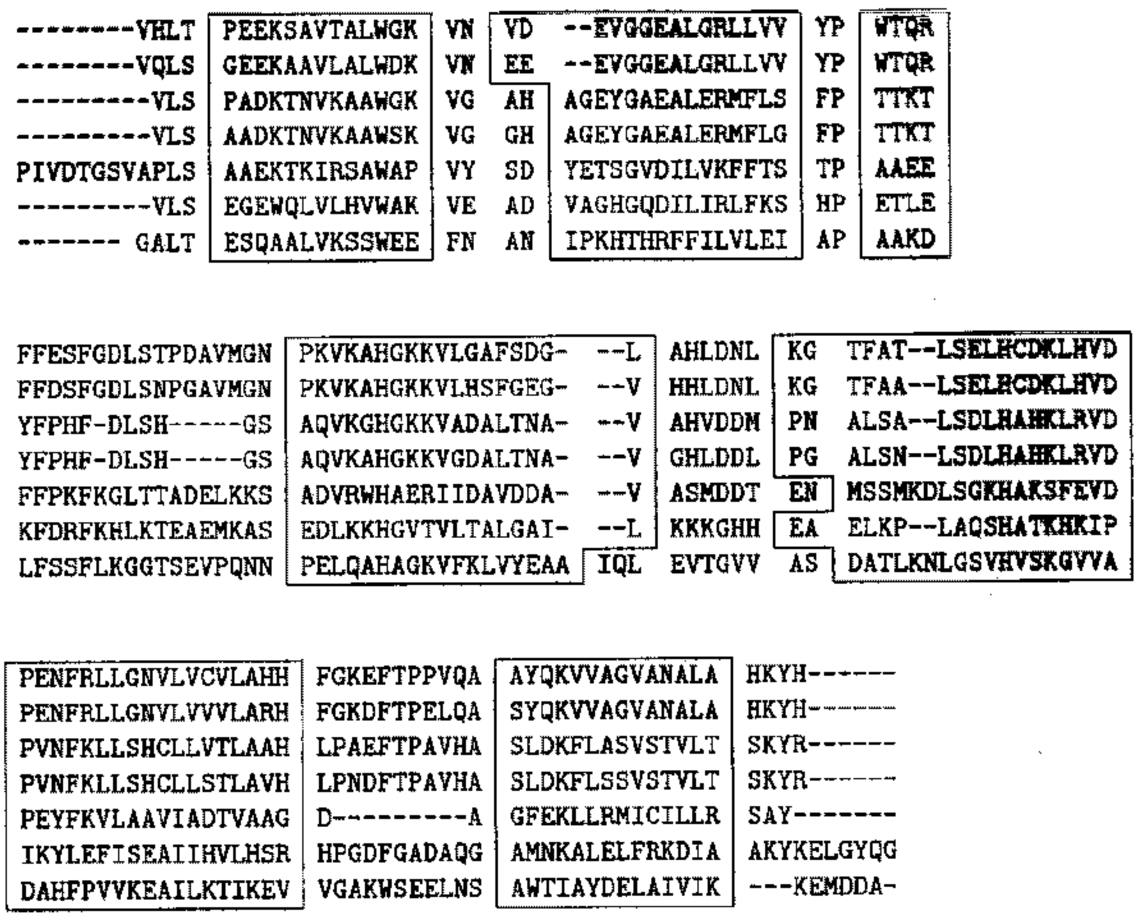
For an example of a multiple alignment, see Figure 4.1.

```
AC..BCDB
.CADB.D.
ACA.BCD.
```

Figure 4.1: A multiple alignment of $ACBCBD$, $CADDB$ and $ACABCD$.

We are interested in finding a common alignment of several sequences, because this multiple similarity suggests a common structure of the protein product, a common function or a common evolutionary source. A multiple alignment carries more information than a pairwise one, as a protein can be matched against a family of proteins instead of only against another one. See Figure 4.2 for example.

[1]Based in part on a scribe by Elery Pfeffer and Orit Kilper, November 20, 2000 and Itay Lotan and Ziv Modai, January 3 ,1999 and a scribe by Sariel Har-Peled, December 18, 1995.



```
---------VHLT  PEEKSAVTALWGK  VN  VD  --EVGGEALGRLLVV  YP  WTQR
---------VQLS  GEEKAAVLALWDK  VN  EE  --EVGGEALGRLLVV  YP  WTQR
----------VLS  PADKTNVKAAWGK  VG  AH  AGEYGAEALERMFLS  FP  TTKT
----------VLS  AADKTNVKAAWSK  VG  GH  AGEYGAEALERMFLG  FP  TTKT
PIVDTGSVAPLS  AAEKTKIRSAWAP  VY  SD  YETSGVDILVKFFTS  TP  AAEE
----------VLS  EGEWQLVLHVWAK  VE  AD  VAGHGQDILIRLFKS  HP  ETLE
-------- GALT  ESQAALVKSSWEE  FN  AN  IPKHTHRFFILVLEI  AP  AAKD

FFESFGDLSTPDAVMGN  PKVKAHGKKVLGAFSDG-  --L  AHLDNL  KG  TFAT--LSELHCDKLHVD
FFDSFGDLSNPGAVMGN  PKVKAHGKKVLHSFGEG-  --V  HHLDNL  KG  TFAA--LSELHCDKLHVD
YFPHF-DLSH-----GS  AQVKGHGKKVADALTNA-  --V  AHVDDM  PN  ALSA--LSDLHAHKLRVD
YFPHF-DLSH-----GS  AQVKAHGKKVGDALTNA-  --V  GHLDDL  PG  ALSN--LSDLHAHKLRVD
FFPKFKGLTTADELKKS  ADVRWHAERIIDAVDDA-  --V  ASMDDT  EN  MSSMKDLSGKHAKSFEVD
KFDRFKHLKTEAEMKAS  EDLKKHGVTVLTALGAI-  --L  KKKGHH  EA  ELKP--LAQSHATKHKIP
LFSSFLKGGTSEVPQNN  PELQAHAGKVFKLVYEAA  IQL  EVTGVV  AS  DATLKNLGSVHVSKGVVA

PENFRLLGNVLVCVLAHH  FGKEFTPPVQA  AYQKVVAGVANALA  HKYH------
PENFRLLGNVLVVVLARH  FGKDFTPELQA  SYQKVVAGVANALA  HKYH------
PVNFKLLSHCLLVTLAAH  LPAEFTPAVHA  SLDKFLASVSTVLT  SKYR------
PVNFKLLSHCLLSTLAVH  LPNDFTPAVHA  SLDKFLSSVSTVLT  SKYR------
PEYFKVLAAVIADTVAAG  D---------A  GFEKLLRMICILLR  SAY-------
IKYLEFISEAIIHVLHSR  HPGDFGADAQG  AMNKALELFRKDIA  AKYKELGYQG
DAHFPVVKEAILKTIKEV  VGAKWSEELNS  AWTIAYDELAIVIK  ---KEMDDA-
```

Figure 4.2: An Alignment between globins produces by an alignment program Clustal. The proteins that appear in the alignment are human beta globin, horse beta globin, human alpha globin, horse alpha globin, cyanohaemoglobin, whale myoglobin and leghaemoglobin in that order. The boxes mark the seven α helices composing each globin.

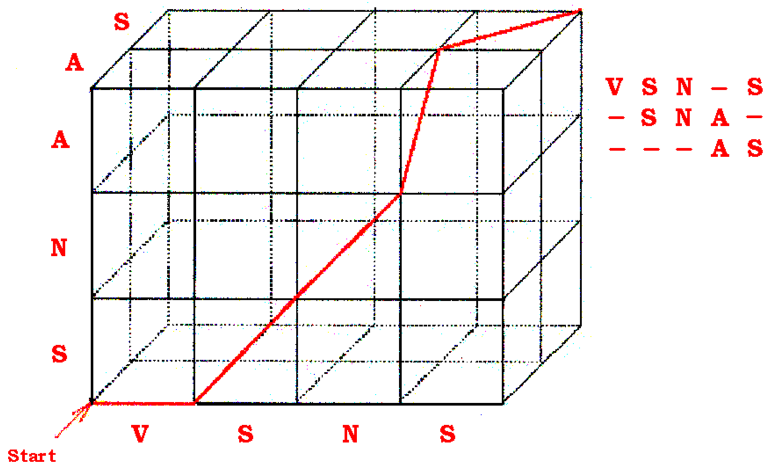


Figure 4.3: Alignment path for 3 sequences [7].

4.1.1 Dynamic Programming Solution

The best multiple alignment of r sequences is calculated using an r-dimensional hyper-cube D (for example see Figure 4.3), defining $D(j_1, j_2, \ldots, j_r)$ to be the best score for aligning the prefixes of lengths $j_1, j_2, \ldots, j_r$ of the sequences $x_1, x_2, \ldots, x_r$, respectively.
We define

$$D(0, 0, \ldots, 0) = 0$$

And we calculate

$$D(j_1, j_2, \ldots, j_r) = min_{\epsilon \in \{0,1\}^n,\, \epsilon \neq 0}\{D(j_1 - \epsilon_1, j_2 - \epsilon_2, \ldots, j_r - \epsilon_r) + \rho(\epsilon_1 x_{j_1}, \ldots, \epsilon_r x_{j_r})\}$$

where ρ is the cost function, and

$$\epsilon = (\epsilon_1, \epsilon_2, ..., \epsilon_n) \in \{0, 1\}^n$$

is a vector that indicates the directions of alignment progress in the hyper-cube. The size of the hyper-cube is $O(\prod_{j=1}^{r} n_j)$, where n_j is the length of x_j, where computation of each of each entry consider $2^r - 1$ others. If $n_1 = n_2 = \ldots = n_r = n$, the space complexity is of $O(n^r)$ and the time complexity is $O(2^r n^r) \cdot O(\text{computation of the } \rho \text{ function})$. Hence the exact solution, using dynamic programming, is practical for only a small number of strings. Moreover, the exact multiple alignment problem, using sum-of-pairs or evolutionary-tree scoring metrics, has be proven to be NP-complete [9].

4.1.2 Scoring Metrics

There are several known useful possibilities for measuring the divergence of a set of aligned sequences, namely the total distance between them (i.e., possible ρ functions):

- *Distance from Consensus* - The consensus of an alignment is a string of the most common characters in each column of the alignment. The total distance between the strings is defined as the number of characters that differ from the consensus character of their column: let C be the consensus sequence, then the total distance is $\sum_i D(S_i, C)$.
- *Evolutionary Tree Alignment* - The weight of the lightest evolutionary tree that can be constructed from the sequences, with the weight of the tree defined as the number of changes between pairs of sequences that correspond to two adjacent nodes in the tree, summed over all such pairs.
- *Sum of Pairs* - The sum of pairwise distances between all pairs of sequences:

$$\sum_{i<j} D(S_i, S_j).$$

4.1.3 Faster DP for Multiple Alignment

Carrillo and *Lipman* [1] found a heuristic method for accelerating the search for the best multiple alignment. An extension of their idea was implemented in the program called MSA [6]. The method is based on the property that if the strings are relatively similar, the alignment path would be close to the main diagonal. Therefore not all the values in the multi-dimensional cube need to be calculated. Assuming an upper bound on cost of the best alignment, we will discard some alignments that are a priori known to be more expensive than the bound on the cost.

Let A be an alignment of sequences $x_1, x_2, \ldots, x_r$. Denote by $A_{i,j}$ the pair of rows in A containing only x_i and x_j, and by $c(A_{i,j})$ the cost of this pairwise alignment. The only difference from usual pairwise alignment is the possibility of alignment of space against space, that can be ignored. Denote by $c(A)$ the total cost of A, and suppose we define $c(A) = \sum_{i<j} c(A_{i,j})$. Let A^* be the optimal alignment (the one with the minimal cost), and suppose we know that $c(A^*) \leq c'$. Therefore,

$$c' \geq c(A^*) = \sum_{i<j} c(A^*_{i,j}) = c(A^*_{u,v}) + \sum_{i<j,(i,j)\neq(u,v)} c(A^*_{i,j}) \geq c(A^*_{u,v}) + \sum_{i<j,(i,j)\neq(u,v)} D(x_i, x_j)$$

where $D(x, y)$ is the optimal score for aligning strings x and y, and u, v are arbitrary chosen indices so that

$$1 \leq u \leq r, 1 \leq v \leq r, u \neq v.$$

It follows that

$$c(A^*_{u,v}) \leq c' - \sum_{i<j,(i,j)\neq(u,v)} D(x_i, x_j)$$

$A^*_{u,v}$ is a projection of A^* to the uv-plain. By calculating $D(x_i, x_j)$ for each i and j, we can find $B(u, v) = c' - \sum_{i<j,(i,j)\neq(u,v)} D(x_i, x_j)$.

Now, consider a cell $(i_1, i_2, \ldots, i_u = s, \ldots, i_v = t, \ldots, i_r)$ whose projection to the uv-plane is (s, t). If the best alignment A^* passes through this cell, then its projection $A^*_{u,v}$ passes through (s, t), and its cost $c(A^*_{u,v})$ agrees with $best^{(u,v)}_{s,t} \leq c(A^*_{u,v}) \leq B(u, v)$ where $best^{(u,v)}_{s,t}$ is an upper bound on the optimal score for an alignment through (s, t) in the uv-plain. We can compute such an upper bound as:

$$best^{(u,v)}_{i,j} = D(x_{u,1}x_{u,2}\ldots x_{u,i-1}, x_{v,1}x_{v,2}\ldots x_{v,j-1}) + d(x_{u,i}, x_{v,j}) +$$

$$+D(x_{u,i+1}\ldots x_{u,n_u}, x_{v,j+1}\ldots x_{v,n_v})$$

where $d(\kappa_1, \kappa_2)$ is the cost of matching the characters κ_1 and κ_2.
This can be computed by forward dynamic programming, keeping a queue of cells whose final D value has not yet been set. The algorithm will set the D value of the cell w at the head of the queue and remove that cell. When it does, it updates the shortest distance from cell $(0, ..., 0)$ to all neighbors of w, whose D value can be influenced by w, and if any of the neighbors is not yet in the queue, it is added to the end of the queue [4, p 346].

Therefore if $best^{(u,v)}_{s,t} > B(u, v)$, then the best alignment A^* cannot pass through the cell $(i_1, i_2, \ldots, i_u = s, \ldots, i_v = t, \ldots, i_r)$ for any $i_1, i_2, \ldots, i_{u-1}, i_{u+1}, \ldots, i_{v-1}, i_{v+1}, \ldots, i_r$, and these cells can be discarded from the computation of the r-dimentional DP.

Notice that the bound c' is found by using heuristics giving "*promising*" solutions first. In practice, MSA [6] can align ~ 6 sequences of length ~ 200 [4, p 346].

4.2 Approximation Algorithms for Multiple Sequence Alignment

Denote by $D(S, T)$ the best score of aligning S with T. Let $\sigma(x, y)$ be our scoring function, i.e., the price of aligning the character x with the character y, for $x, y \in \Sigma \cup \{-\}$.
We assume that

- $\sigma(-, -) = 0$,
- $\sigma(x, y) = \sigma(y, x)$,
- the triangle inequality $\sigma(x, y) \leq \sigma(x, z) + \sigma(z, y)$ holds.

4.2.1 The Center Star Method for (SP) Alignment

In this section, we present an approximation algorithm for calculating the optimal multiple alignment under the SP metric (see, e.g., [4, pp 348–350]. The algorithm achieves an approximation ratio of two.

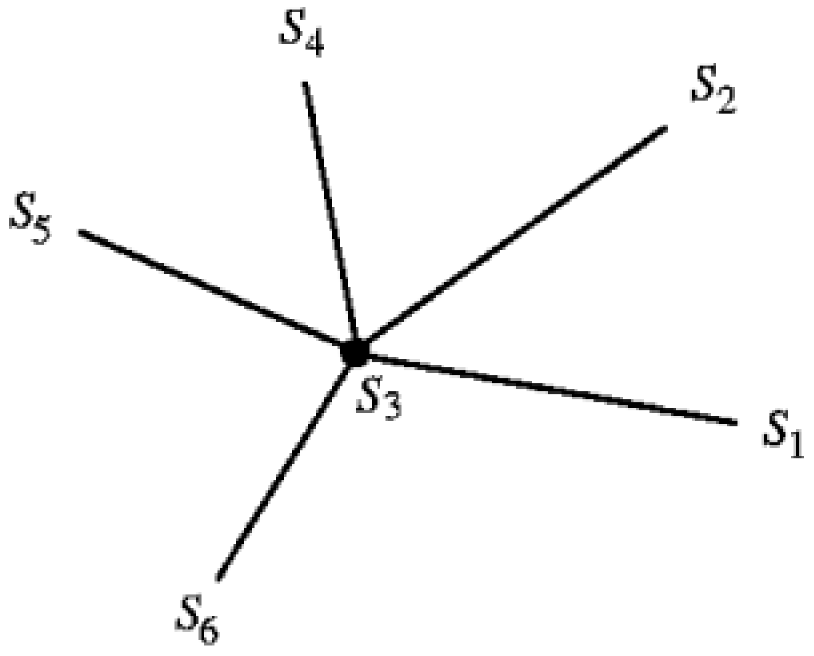


Figure 4.4: A generic center star for six strings, where the center string (S_c) is S_3 [4, p 349].

Problem 4.1 The SP alignment problem.
INPUT: A set of sequences $\mathcal{S} = \{S_1, S_2, ..., S_k\}$.
QUESTION: Compute a global multiple alignment $\mathcal{M}$ with minimum sum-of-pairs score.

Definition 4.1 Given a set of k strings $\mathcal{S}$, define a *center string* $S_c \in \mathcal{S}$ as a string that minimizes $\sum_{S_j \in \mathcal{S}} D(S_c, S_j)$.

Definition 4.2 Define *the center star* to be a star tree of k nodes, with the center node labelled S_c and with each of the $k-1$ remaining nodes labelled by a distinct string in $\mathcal{S} \setminus \{S_c\}$. For an example, see Figure 4.4.

Definition 4.3 Define the multiple alignment $\mathcal{M}_c$ of the set of strings $\mathcal{S}$ to be the multiple alignment consistent with the center star.

The Center Star Algorithm:

1. Find $S_t \in \mathcal{S}$ minimizing $\sum_{i \neq t} D(S_i, S_t)$ and let $\mathcal{M} = \{S_t\}$.

2. Add the sequences in $\mathcal{S} \setminus \{S_t\}$ to $\mathcal{M}$ one by one so that the alignment of every newly added sequence with S_t is optimal. Add spaces, when needed, to all pre-aligned sequences.

Running time analysis:

1. $\binom{k}{2}\, O(n^2)$ for step 1.

2. $\sum_{i=1}^{k-1} O((i \cdot n) \cdot n) = O(k^2 \cdot n^2)$ for step 2 (since the worst-case length of S'_t after the addition of i strings is $(i+1) \cdot n$) [4, p 348].

Approximation analysis:

- Let $\mathcal{M}$ denote the multiple alignment produced by the algorithm.
- Let $d(i,j)$ be the score of the pairwise alignment it induces on S_i, S_j. (Note that $D(S_i, S_j) \leq d(i,j)$).
- Let $\sigma(\mathcal{M}) = \sum_{i=1}^{k}\ \sum_{j \neq i, j=1}^{k} d(i,j)$.
- Let $\mathcal{M}^*$ denote the optimal alignment of $\mathcal{S}$.
- Let $d^*(i,j)$ denote the value of the alignment between S_i and S_j induced by $\mathcal{M}^*$.

We assume w.l.o.g that S_1 is the center found by the algorithm, so for each $1 \leq l \leq k, d(1,l) = D(S_1, S_l)$.

Theorem 4.1

$$\frac{\sigma(\mathcal{M})}{\sigma(\mathcal{M}^*)} \leq \frac{2(k-1)}{k} < 2.$$

Proof:

$\sigma(\mathcal{M}) = \sum_{i=1}^{k} \sum_{j \neq i, j=1}^{k} d(i,j) \leq \sum_{i=1}^{k} \sum_{j \neq i, j=1}^{k} [d(i,1) + d(1,j)] =$

$$= 2(k-1) \sum_{m=2}^{k} d(1,m) = 2(k-1) \sum_{m=2}^{k} D(S_1, S_m) \tag{4.1}$$

The inequality follows from the triangle inequality. Since the triangle inequality holds for every single column of the alignment by the definition of the scoring scheme, it also holds for entire strings by the definition of d. Also,

$k \sum_{m=2}^{k} D(S_1, S_m) = \sum_{i=1}^{k} \sum_{j=2}^{k} D(S_1, S_j) \leq$

$$\leq \sum_{i=1}^{k} \sum_{j \neq i, j=1}^{k} D(S_i, S_j) \leq \sum_{i=1}^{k} \sum_{j \neq i, j=1}^{k} d(i,j) = \sigma(M^*) \tag{4.2}$$

The theorem follows. ■

Theorem 4.1 implies that calculating the multiple alignment of the center star produces a multiple alignment with a value which is at most $R_k = \frac{2(k-1)}{k}$ times the value of the optimal alignment. For example $R_3 = \frac{4}{3}$, $R_4 = \frac{3}{2}$.

4.2.2 Multiple Alignment with Consensus

In this section we look at an approximation algorithm for a multiple alignment that optimizes a different score metric - the consensus error. As before, we assume the existence of a pairwise scoring scheme σ satisfying the triangle inequality.
Notice, that Consensus sequence doesn't have to be similar to any of the aligned sequences (see Figure 4.5).

```
A B — C — D
A B C C — D
A B C D E —
___________
A B C D E D
```

Figure 4.5: An example of multiple alignment with consensus string.

Definition 4.4 Given a set of strings $\mathcal{S}$, the *consensus error* of a string $\bar{S}$ with respect to $\mathcal{S}$ is $E(\bar{S}) = \sum_{S_i \in \mathcal{S}} D(\bar{S}, S_i)$. Note that $\bar{S}$ need not be in $\mathcal{S}$.

Definition 4.5 $\mathcal{S}^*$ is an optimal Steiner string for $\mathcal{S}$ if it minimizes $E(S)$.

Problem 4.2 Optimal Steiner string.
INPUT: A set of strings $\mathcal{S}$.
QUESTION: Find a string S^* which minimizes the consensus error $E(\mathcal{S})$.

The Steiner string S^* attempts to capture the common characteristics of the set of strings $\mathcal{S}$ and reflect them in a single string. We will present an approximation algorithm for the optimal Steiner string problem with worst-case approximation ratio of 2.

Lemma 4.2 *[4, pp 349–351] Let $\mathcal{S}$ contain k strings, and assume that the scoring scheme σ satisfies the triangle inequality. Then there exists a string $\bar{S} \in \mathcal{S}$ such that $\frac{E(\bar{S})}{E(S^*)} \leq 2 - \frac{2}{k} < 2$.*

Proof: For any $\bar{S} \in \mathcal{S}$:

$$E(\bar{S}) = \sum_{S_i \in \mathcal{S}} D(\bar{S}, S_i) \leq \sum_{S_i \neq \bar{S}} [D(\bar{S}, S^*) + D(S^*, S_i)] = \tag{4.3}$$

$$(k-2) \cdot D(\bar{S}, S^*) + D(\bar{S}, S^*) + \sum_{S_i \neq \bar{S}} D(S^*, S_i) = (k-2) \cdot D(\bar{S}, S^*) + E(S^*)$$

If we pick $\bar{S} \in \mathcal{S}$ such that $\bar{S}$ is the closest to S^* then:

$$E(S^*) = \sum_{S_i \in \mathcal{S}} D(S^*, S_i) \geq k \cdot D(\bar{S}, S^*) \tag{4.4}$$

The *center string* $S_c \in \mathcal{S}$ minimizes $\sum_{S_i \in \mathcal{S}} D(S_c, S_i)$ and therefore its consensus error is smaller then the consensus error of $\bar{S}$ (the string closest to S^*). Hence:

$$\frac{E(S_c)}{E(S^*)} \leq \frac{E(\bar{S})}{E(S^*)} \leq \frac{(k-2)\cdot D(\bar{S},S^*) + E(S^*)}{E(S^*)} \leq \frac{(k-2)\cdot D(\bar{S},S^*)}{k \cdot D(\bar{S},S^*)} + 1 = 2 - \frac{2}{k} < 2. \quad (4.5)$$

The proof above uses the lemma 4.2 and the fact that $E(S_c) \leq E(\bar{S})$ by definition. ■

It is worthwhile noting that Steiner string was defined without alignment, and the only requirement is the distance function, that satisfies the triangle inequality. We will next start discussing consensus strings that are alignment motivated.

4.2.3 Consensus Strings from Multiple Alignment

Definition 4.6 Given a multiple alignment $\mathcal{M}$ of a set of strings $\mathcal{S}$, the *consensus character* in column i of $\mathcal{M}$ is the character that minimizes the sum of distances to it from all the characters in column i. Let $d(i)$ denote that minimum sum in column i.

Definition 4.7 The *consensus string* $S_{\mathcal{M}}$ derived from the alignment $\mathcal{M}$ is the concatenation of the consensus characters for each column of $\mathcal{M}$.

Definition 4.8 The *alignment error* of $S_{\mathcal{M}}$ equals $\sum_{i=1}^{l} d(i)$ where l is the number of characters in $S_{\mathcal{M}}$.

Definition 4.9 The *optimal consensus multiple alignment* is a multiple alignment $\mathcal{M}$ of an input set $\mathcal{S}$ whose consensus string $S_{\mathcal{M}}$ minimizes the alignment error. It can be shown that the optimal consensus multiple alignment is equal to the optimal Steiner string, as defined in section 4.2.2.

We can use the center string (S_c) for approximating the optimal multiple alignment with an alignment error smaller than $(2 - \frac{2}{k})$ times the optimal alignment error.

Theorem 4.3 *[4, p 353]. Let S denote the consensus string of the optimal consensus multiple alignment. Then, removal of the spaces from S creates the optimal Steiner string S^*. Conversely, removal of the row for S^* from the multiple alignment consistent with S^* creates the optimal consensus multiple alignment of S.*

Proof: Let S have k strings, and let $\mathcal{M}$ be any multiple alignment of S. By definition, each character of the consensus string $S_{\mathcal{M}}$ derived from $\mathcal{M}$ is associated with a distinct column of $\mathcal{M}$, and this association induces a particular pairwise alignment between $S_{\mathcal{M}}$ and each S_i in $\mathcal{S}$. Clearly, the score of that alignment is at least $D(S_i, S_{\mathcal{M}})$. Now the alignment error of $S_{\mathcal{M}}$ is exactly the sum of the scores of those k pairwise induced alignments, and so the alignment error of $S_{\mathcal{M}}$ is at least $\sum_i D(S_i, S_{\mathcal{M}})$, which is the consensus error of $S_{\mathcal{M}}$ for $\mathcal{S}$. But by definition, S^* has the minimum consensus error for $\mathcal{S}$, so the alignment error of $S_{\mathcal{M}}$ is at least the consensus error of S^*.

Now consider the multiple alignment $\mathcal{M}^*$ of $\mathcal{S} \bigcup S$ consistent with S^*, and for any string α in $\mathcal{S} \bigcup S$, let $\bar{\alpha}$ denote the string in the row of $\mathcal{M}^*$ corresponding to α. By consistency, the score of the induced alignment in $\mathcal{M}^*$ of $\bar{S}^*$ and $\bar{S}_i$ is $D(S^*, S_i)$ for any $S_i \in \mathcal{S}$. Let $\mathcal{M}'$ be $\mathcal{M}^*$ after removing the row for S^*. Then, the alignment error of $\bar{S}^*$ with $\mathcal{M}'$ is exactly $\sum_i D(S^*, S_i)$, which is the consensus error of S^* for $\mathcal{S}$. Hence, using the conclusion from the first paragraph, the alignment error of any other consensus string $S_{\mathcal{M}}$ for any other multiple alignment $\mathcal{M}$ must be at least as large as the alignment error of $\bar{S}^*$ for $\mathcal{M}'$. It follows that $\mathcal{M}'$ is the optimal consensus multiple alignment for $\mathcal{S}$ and that $\bar{S}^*$ is its consensus string. Therefore, since S^* is obtained from $\bar{S}^*$ by removing the spaces in $\bar{S}^*$, the theorem is proved. ■

4.3 Multiple Alignment to a Phylogenetic Tree

Definition 4.10 A tree T with a distinct string label (from a set of strings $\mathcal{S}$) assigned to each leaf is called a *phylogenetic tree* on $\mathcal{S}$.

Definition 4.11 Given a phylogenetic tree T on $\mathcal{S}$, a *phylogenetic alignment* T' for T is an assignment of one string label to each internal node of T. Note that the strings assigned to internal nodes need not be distinct and need not be from the set $\mathcal{S}$. For an example, see Figure 4.6.

For example, when T is a star, choosing the sequence to label its central node is a phylogenetic alignment.

The phylogenetic tree T is meant to represent the "established" evolutionary history of a set of objects of interest, with the convention that each extant object is represented at a unique leaf of the tree. Each edge (u, v) represents some evolutionary history that transforms the string at u (assuming u is the parent of v) to the string at v. For convenience, when denoting an edge by a pair of nodes, we shall hereafter write the parent node first.

Definition 4.12 If strings S and S' are assigned to the endpoints of an edge (i, j), then the *edge distance* of (i, j) is defined to be $D(S, S')$.

Definition 4.13 Let $\mathcal{M}_T$ be a phylogenetic alignment for a phylogenetic tree T. The *distance of* $\mathcal{M}_T$ is given by the sum of all the edge distances over all the edges of T.

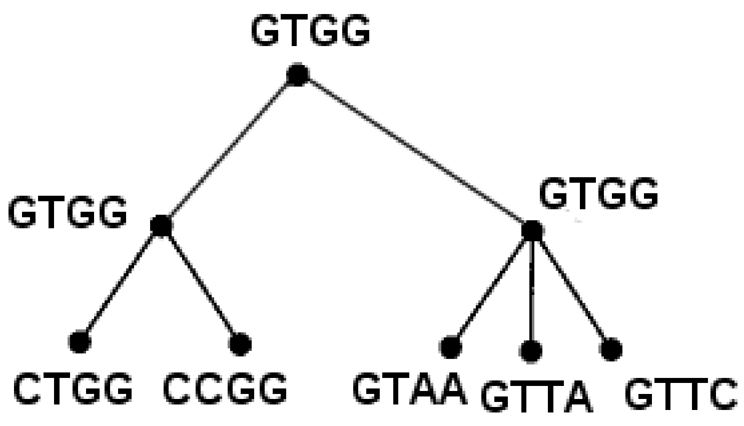


Figure 4.6: A phylogenetic tree alignment.

Problem 4.3 Phylogenetic alignment:
INPUT: A set $\mathcal{S} = (S_1, \ldots, S_k)$ of strings and a phylogenetic tree T on $\mathcal{S}$.
QUESTION: Find a phylogenetic alignment $\mathcal{M}_T$ with minimum distance.

We assume that the structure of the tree T is known to us. This usually happens in practice when T was previously reconstructed using solid evolutionary data.

Notice that this problem is also NP-complete [9], so a heuristic solution must be used.

4.3.1 Lifted Alignment tree - a Heuristic for Phylogenetic Alignment.

Jiang-Wang-Lawler 1996 [5]

Definition 4.14 A phylogenetic alignment is called *lifted alignment* if for every internal node v, the string assigned to v is also assigned to one of v's children (see Figure 4.7). Trivially, in such a case, all internal nodes are assigned labels from the set of leaf strings.

Let T^* be the optimal alignment for a tree T. We will construct a lifted alignment $T^L = Lift(T^*)$, which is based on T^*, and has at most twice the pairwise distance. Note that this construction is only conceptual since we usually do not know T^*.

For each node v let its label in T^* be S_v^*. We shall assign every v a label S_v^L. Initially, only the leaves are labelled, and by definition $S_v^L = S_v^*$ for each leaf v. The labelling process

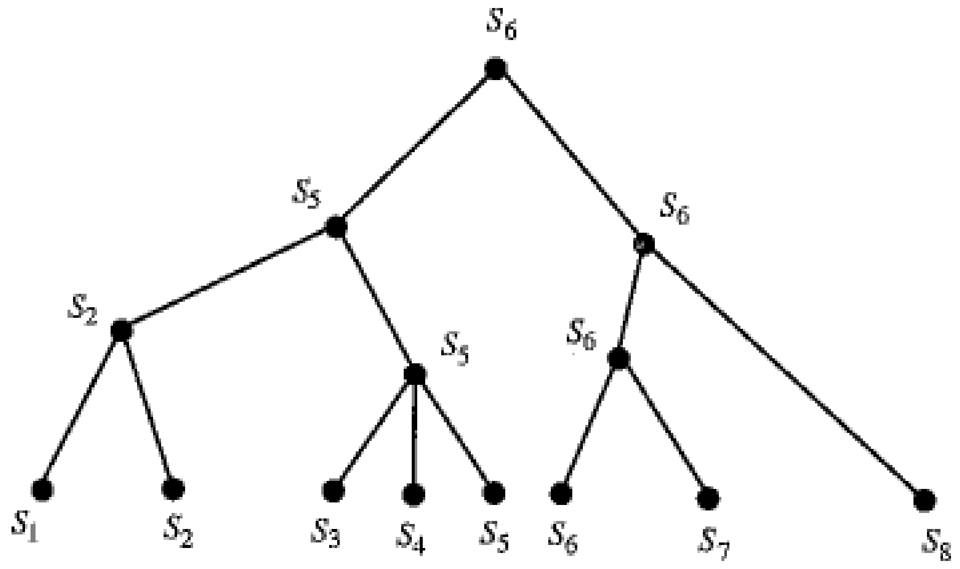


Figure 4.7: A phylogenetic tree with lifted alignment. Each internal node is labelled by one of the strings labelling its children [4, p 355] .

successively traverses the internal nodes in any order, provided that a node is not visited before any of its children. Upon visiting a node, it is *lifted* i.e. labelled by one of the labels of its children. Thus, the resulting phylogenetic alignment is lifted (see Figure 4.8).

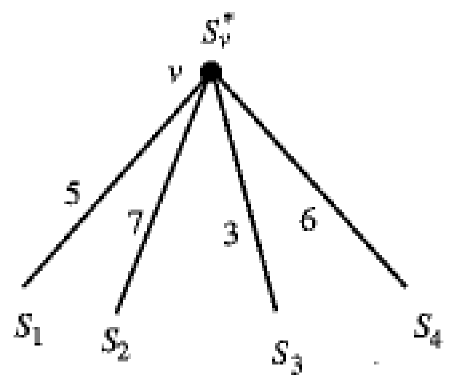
```
Procedure Lift(T : Tree)
  begin
    while there exists an unlifted node $v$, all of whose children have been lifted, do :
      Find a child $j$ whose label $S_j$ is the closest to $S_v^*$. Namely,
        for every child $i$ of $v$: $D(S_v^*, S_j) \le D(S_v^*, S_i)$
      Label $S_v^*$ with $S_j$
    end while
  end
```

Theorem 4.4 *[5] The distance of the phylogenetic alignment $T^L = Lift(T^*)$ is at most twice the distance of the optimal phylogenetic alignment T^*.*

Proof: Let $e = (v, w)$ be an edge in T. Suppose that in T^L, S_j is the label of v and S_i is the label of w. If $i = j$ then $D(S_j, S_i) = 0$. Otherwise:

$$D(S_i, S_j) \le D(S_j, S_v^*) + D(S_v^*, S_i) \le 2 \cdot D(S_v^*, S_j) \qquad (4.6)$$

The first inequality is due to the triangle inequality, and the second follows from the labelling algorithm. For an edge $e = (v, w)$ with $S_w = S_i$, let P_e be the path in T from v to the leaf



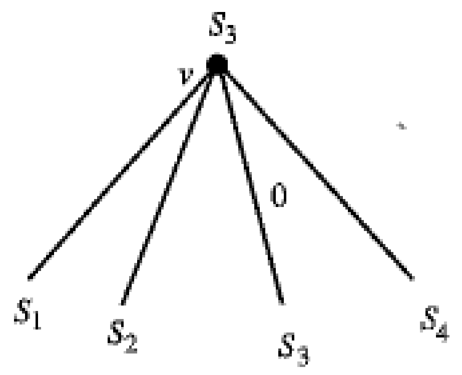


Figure 4.8: The lifting construction at node v. The numbers on the edges are the distances from S_v^* to the lifted strings labelling its children. On the left is the tree before lifting, and on the right the result of the lift. After the lift one edge will have a distance of 0 [4, p 356].

labelled S_i. Due to the triangle inequality

$$D(S_v^*, S_i) \leq \text{the total length of } P_e \text{ in } T^* \tag{4.7}$$

Define a path as a collection of nodes in T^* starting from a leaf and continuing while its sequence is lifted. Then T^* can be partitioned into such pathes.

We say that the edge $e = (v, w)$ is *blue* in T^L if v is labelled with string S_j and w is labelled with S_i, $S_i \neq S_j$. The distance of a lifted alignment T^L is equal to the sum of edge distances on all the blue edges in the tree. For a blue edge $e = (v, w)$, observe that the definition of lifted alignment implies that along the path P_e every node except v is labelled S_i, and no node outside P_e is labelled S_i. Hence, if $e' = (v', w')$ is any other blue edge, then P_e and $P_{e'}$ have no edges in common. This defines a mapping from every blue edge e in T^L to a path P_e in T^* such that:

- The distance in T^L of the edge e is at most twice the total distance in T^* of the edges on P_e (follows from equations 4.6 and 4.7).
- No edge in T^* is mapped to by more than one edge in T^L.

Therefore the total distance of T^L equals the total distance on blue edges, which is at most twice the sum of all total distances of paths in T^* and therefore at most twice the total distance of T^* (see Figure 4.9). ■

We now describe how to find the optimal lifted alignment using a dynamic programming algorithm as listed below. But first we define:

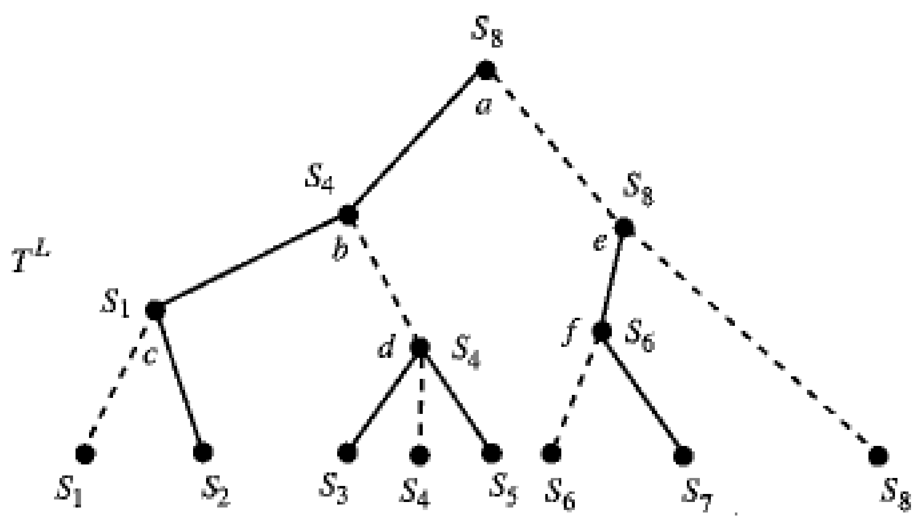


Figure 4.9: The lifted tree T_L^*. The dashed edges show the paths along which a leaf string has been lifted to some internal node, thus their distance is 0. Solid edges are blue edges in T_L^*. The path $P_{(a,b)}$ for example, is the path b, d, S_4 along which the string labelling b was lifted. Edge (a, b) has distance in T_L^* at most twice the distance of path $P_{(a,b)}$ in T^* [4, p 357].

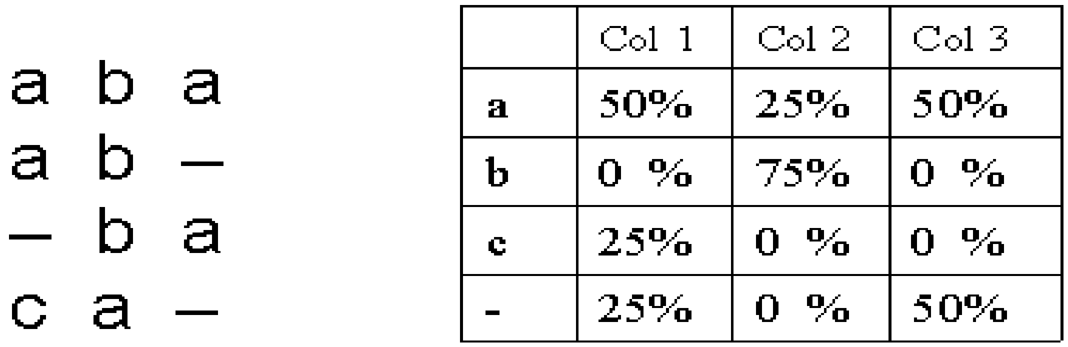
Definition 4.15 Let T_v be the subtree of T rooted at node v and $S \in \mathcal{S}$. Let $d(v, S)$ denote the distance of the best lifted alignment of T_v under the requirement that string S is assigned to node v.

The algorithm will compute $d(v, S)$ for any $S \in \mathcal{S}$ working its way from the leaves up using the following recursion:

- If v is an internal node with all its children being leaves, then $d(v, S) = \sum_{(v,w)} D(S, S_w)$ where S_w is the label of w.
- Else, $d(v, S) = \sum_{(v,v')} min_{S' \in S}[D(S, S') + d(v', S')]$ where v' is a child of v and S' is a label of one of the leaves of $T_{v'}$.

After the computation is finished, the value of the best lifted alignment is the minimum of $d(root, S)$, where S ranges over all strings written at the leaves of T. The best lifted alignment has total distance less than twice that of the optimal phylogenetic alignment.

Time analysis: We perform a preprocessing stage, in which we compute all the $\binom{k}{2}$ pairwise distances between the k input strings. This takes $O(N^2)$ time, where N is the total length of



```
a b a
a b —
— b a
c a —
```

	Col 1	Col 2	Col 3
a	50%	25%	50%
b	0 %	75%	0 %
c	25%	0 %	0 %
-	25%	0 %	50%

Figure 4.10: Example of profile multiple alignment.

all the strings. The work at any internal node is $O(k^2)$, and the overall work of the algorithm is $O(N^2 + k^3)$.

4.4 Common Multiple Alignments Methods

4.4.1 Aligning a String to a Profile

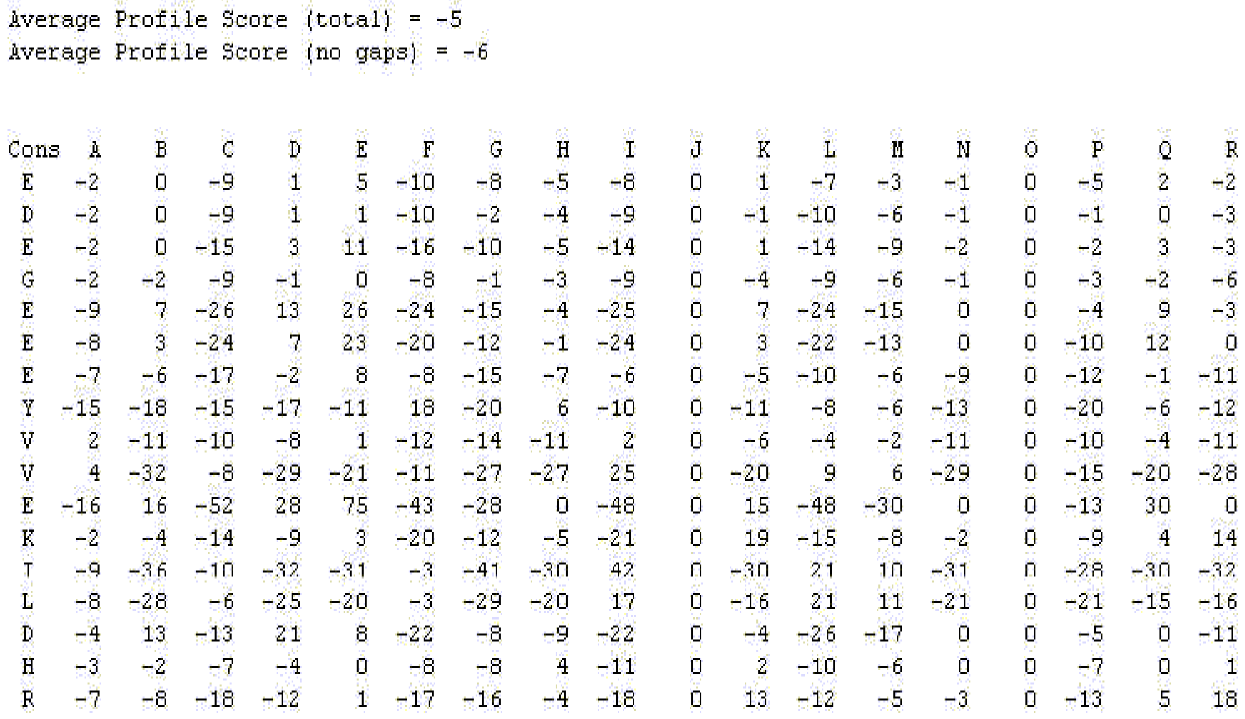
Given a database of sequences, we would like to partition it into families of "similar" sequences. For this purpose we would like to encompass our knowledge on the common properties of the sequences in a family *profile* (formal definition to follow). Constructing a profile of a family enables us to identify its members and test whether or not a new sequence belongs to the family. Moreover, searching the database with a profile is more sensitive than searching using a single sequence of the family: when considering a string for membership in an established family, or when searching a database for new candidate members of a family, it is usually mush more effective to align the candidate string to a representation of the known family members, rather than aligning to single members of the family. When successful, identifying the family of a new newly identified protein is very useful, as the researcher gains tremendous clues about the physical structure or biological function of the protein.

A profile of a multiple alignment gives letter frequencies per column (for example, see Figure 4.10). Alternatively, log likelihood ratios can be used: $\log p_i(a)/p(a)$, where $p_i(a)$ is the fraction of a's in column i and p(a) is the fraction of a's overall (see Figure 4.11).

Definition 4.16 For an alignment S' of length l, a *profile* is an $l \times |\Sigma \cup \{-\}\,|$ matrix, whose columns are probability vectors denoting the frequencies of each symbol in the corresponding alignment column.

Any alignment between a sequence B and a profile P (i.e. both have the same length) can be evaluated by $\sum_{j=1}^{m} \sigma(p_j, b_j)$. Clearly, using dynamic programming, we can find the best alignment of a sequence against a profile.

The key in pairwise alignment is scoring two positions x and y : $\sigma(x, y)$. For a letter x and a column y of a profile, let $\sigma(x, y)$ be the probability of x being in column y. The value for x depends on the frequency of its occurrences in the column y. We also need to devise a score for $\sigma(x, -)$. In order to find whether a given sequence is a member of certain family, we use the usual pairwise dynamic programming alignment to compare the given sequence to the family profile.



Average Profile Score (total) = -5
Average Profile Score (no gaps) = -6

Cons	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
E	-2	0	-9	1	5	-10	-8	-5	-8	0	1	-7	-3	-1	0	-5	2	-2
D	-2	0	-9	1	1	-10	-2	-4	-9	0	-1	-10	-6	-1	0	-1	0	-3
E	-2	0	-15	3	11	-16	-10	-5	-14	0	1	-14	-9	-2	0	-2	3	-3
G	-2	-2	-9	-1	0	-8	-1	-3	-9	0	-4	-9	-6	-1	0	-3	-2	-6
E	-9	7	-26	13	26	-24	-15	-4	-25	0	7	-24	-15	0	0	-4	9	-3
E	-8	3	-24	7	23	-20	-12	-1	-24	0	3	-22	-13	0	0	-10	12	0
E	-7	-6	-17	-2	8	-8	-15	-7	-6	0	-5	-10	-6	-9	0	-12	-1	-11
Y	-15	-18	-15	-17	-11	18	-20	6	-10	0	-11	-8	-6	-13	0	-20	-6	-12
V	2	-11	-10	-8	1	-12	-14	-11	2	0	-6	-4	-2	-11	0	-10	-4	-11
V	4	-32	-8	-29	-21	-11	-27	-27	25	0	-20	9	6	-29	0	-15	-20	-28
E	-16	16	-52	28	75	-43	-28	0	-48	0	15	-48	-30	0	0	-13	30	0
K	-2	-4	-14	-9	3	-20	-12	-5	-21	0	19	-15	-8	-2	0	-9	4	14
T	-9	-36	-10	-32	-31	-3	-41	-30	42	0	-30	21	10	-31	0	-28	-30	-32
L	-8	-28	-6	-25	-20	-3	-29	-20	17	0	-16	21	11	-21	0	-21	-15	-16
D	-4	13	-13	21	8	-22	-8	-9	-22	0	-4	-26	-17	0	0	-5	0	-11
H	-3	-2	-7	-4	0	-8	-8	4	-11	0	2	-10	-6	0	0	-7	0	1
R	-7	-8	-18	-12	1	-17	-16	-4	-18	0	13	-12	-5	-3	0	-13	5	18

Figure 4.11: Profile for Classical Chromo Domains [10].

4.4.2 Iterative pairwise alignment

This approach uses pairwise alignment scores to iteratively add one additional string to a growing multiple alignment. We start by aligning the two strings whose edit distance is the minimum over all pairs of strings. Then we iteratively consider the string with the smallest distance to any of the strings already in the multiple alignment.

More generally, the algorithm works as follows:

1. Align some pair.
2. while (not done)
   (a) Pick an unaligned string which is "near" some aligned one(s).
   (b) Align with the *profile* of the previously aligned group.
   Resulting new spaces are inserted into all strings in the group.

4.4.3 Progressive alignment

Feng-Doolittle 1987 [2]

In this algorithm, the key idea is that the pair of strings with minimum distance is almost likely to have been obtained from the pair of objects that had most recently diverged, and that the pairwise alignment of these two specific strings provides the most "reliable" information that can be extracted from the input strings. Therefore any spaces (gaps) that appear in the optimal pairwise alignment of those two strings should be preserved in the overall multiple alignment.

The algorithm is as follows:

1. Calculate the $\binom{k}{2}$ pairwise alignment scores, and convert them to distances.
2. Use an incremental clustering algorithm [3] to construct a tree from the distances. (Clustering algorithms will be described later in the course.)
3. Traverse the nodes in their order of addition to the tree, progressively aligning the sequences. This way, the most similar pair is aligned first, followed by the addition of the next most similar sequence or set of sequences.

Features of this heuristic:

- The order of aligning of sequences, or sets of sequences, is determined by their highest scoring pairwise alignment.
- "Once a gap, always a gap": replace gaps in alignments by a neutral character.

CLUSTALW

`ClustalW` is a software package for multiple alignment (implementing an algorithm of Thompson, Higgins, Gibson 1994 [8]). The basic idea is the same as in the Feng-Doolittle algorithm:

1. Calculate the $\binom{k}{2}$ pairwise alignment scores, and convert them to distances.
2. Use a neighbor-joining algorithm to build a tree from the distances.
3. Align sequence - sequence, sequence - profile, profile - profile in decreasing similarity order.

This algorithm makes use of many ad-hoc rules such as weighting, different matrix scores and special gap scores.
For an example of a alignment tree built by ClustalW, see Figure 4.12.

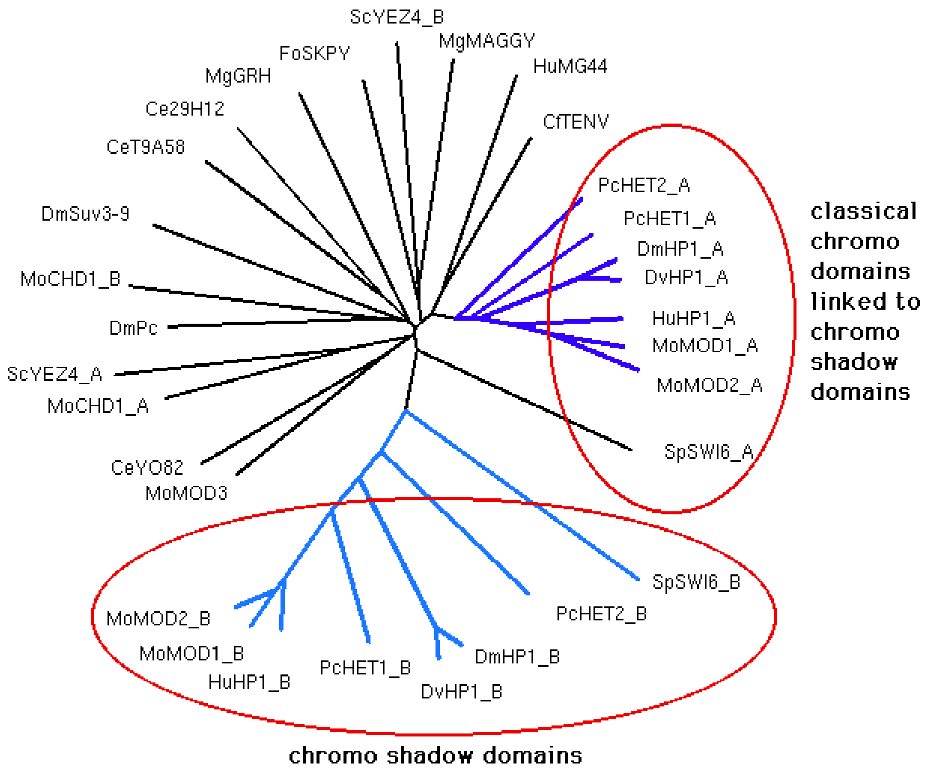


Figure 4.12: Alignment tree built by ClustalW [11].

Bibliography

[1] H. Carrillo and D. Lipmann. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math*, 48:1073–1082, 1988.

[2] D. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, 25:351–360, 1987.

[3] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *science*, 15:279–284, 1967.

[4] D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, New York, 1997.

[5] T. Jiang, L. Wang, and E. L. Lawler. Approximation algorithms for tree alignment with a given phylogeny. *Algorithmica*, 16:302–315, 1996.

[6] D. J. Lipman, S. Altshul, and J. Kececiogly. A tool for multiple sequence alignment. *Proc. Natl. Academy Science*, 86:4412–4415, 1989.

[7] M. Murata, J.S. Richardson, and J.L. Sussman. Three protein alignment. *Medical Information Sciences*, 231:9, 1999.

[8] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22:4673–80, 1994.

[9] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Computational Biology*, 1:337–348, 1994.

[10] `http://www.uib.no/aasland/chromo/chromoCC.html`.

[11] `http://www.uib.no/aasland/chromo/chromo-tree.gif`.