Lecture 4: 20 November, 2000

Lecturer: Ron Shamir *Scribe: Elery Pfeffer and Orit Kliper*

4.1 Multiple Alignment

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

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

```
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.

4.1.1 Dynamic Programming Solution

The best multiple alignment of r sequences is calculated using an r-dimensional hypercube D, 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. 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 of $O(2^r n^r)$.

4.1.2 Scoring Metrics

There are several known useful possibilities for measuring the divergence of a set of aligned strings, namely the total distance between them.

- *Distance from Consensus* - The consensus of an alignment is a string of the most common character 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.
- *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 the pairs of sequences.

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. 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, we now detail this algorithm.

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 strings $x_1, x_2, \ldots, x_r$. 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. 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. 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^* on 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.

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.

4.2 Approximation Algorithms for Multiple Sequence Alignment

4.2.1 Problem Definition

The Problem definition remaines the same.

Denote $D(S, T)$ as 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 $\sigma(-, -) = 0$, $\sigma(x, y) = \sigma(y, x)$, and that the triangle inequality $\sigma(x, y) \leq \sigma(x, z) + \sigma(z, y)$ holds.

4.2.2 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.

Definitions:

Assume $\sigma(x, y)$ is our scoring function i.e., the price of aligning the character x with the character y. For each x,y and z,

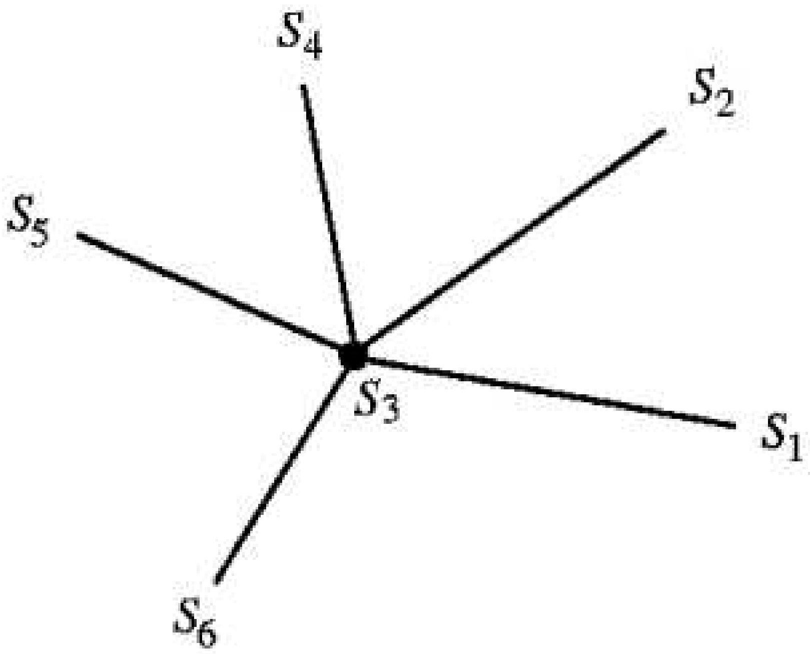

Figure 4.2: A generic center star for six strings, where the center string (S_c) is S_3.

- $\sigma(x, x) = 0$.
- $\sigma(x, y) \leq \sigma(x, z) + \sigma(z, y)$.
- $\sigma(x, y) = \sigma(y, x)$

We denote by $D(S, Y)$ the score of the optimal alignment between sequences S and Y.

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

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'_c after the addition of i strings is $(i+1) \cdot n$)

Approximation analysis:

- Let $\mathcal{M}$ denote the multiple alignment of 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 $\sigma(M) = \sum_{i=1}^{k}\ \sum_{j \neq i, j=1}^{k} d(i,j)$
- Let $\mathcal{M}^*$ denote the optimal alignment of $\mathcal{S}$.
- Let $d^*(i,j)$ denote the value of alignment between S_i and S_j induced by $\mathcal{M}^*$.

We note that $\sigma(M) = 2\times$ *the SP score of* M . We will assume w.l.o.g that S_1 is the center found by the algorithm, so for each $1 \leq l \leq k, d(1,l) = D(S_1, S_l)$. We also note that $\sigma(M^*) = \sum_{i=1}^{k}\ \sum_{j \neq i, j=1}^{k} d^*(i,j)$

Theorem 4.1

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

.

Proof

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

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

The inequality followes 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.

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

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

And the theorem followes. ■

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 $R_3 = \frac{4}{3}$, $R_4 = \frac{3}{2}$.

4.2.3 Multiple Alignment with Consensus

In this section we look at an approximation algorithm for a multiple alignment that optimizes a different score metrics - the consensus error. As before, we assume the existence of a pairwise scoring scheme σ satisfying the triangle inequality.

Definition 4.1 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 $\mathcal{S}$.

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

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 *Let $\mathcal{S}$ have 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 $\frac{E(\bar{S})}{E(S^*)} \leq 2 - \frac{2}{k} < 2$ (see e.g. [4] [pp 349–351]).*

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 the $\bar{S}$ (the string closest to S^*). We get:

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

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

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.

4.2.4 Consensus Strings from Multiple Alignment

Definition 4.2 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 summed distance to it from all the characters in column i. Let $d(i)$ denote that minimum sum in column i.

Definition 4.3 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 4.4 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 4.5 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 4.2.3.

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.

4.3 Multiple Alignment to a Phylogenetic Tree

Definition 4.6 A tree T with a distinct string label (from a set of string $\mathcal{S}$) assigned to each leaf is called a *phylogenetic tree* on $\mathcal{S}$

Definition 4.7 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 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.

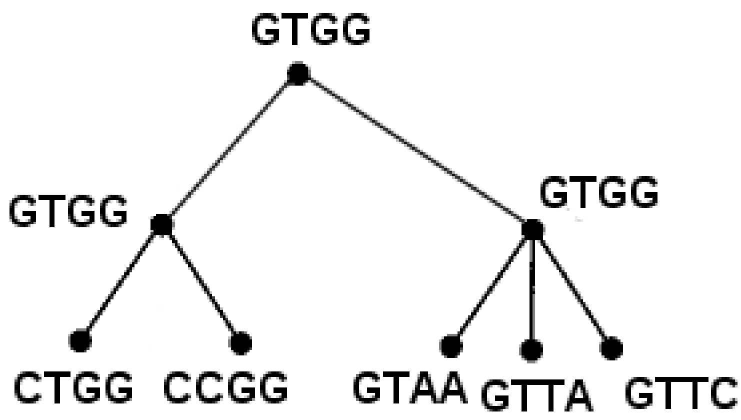

Figure 4.3: A phylogenetic tree alignment

Definition 4.8 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 4.9 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.

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.

4.3.1 Lifted Alignment tree - a Heuristic for Phylogenetic Alignment

Definition 4.10 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 4.4). Trivially, in such a case, all internal nodes are assigned labels from the set of leaf strings.

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

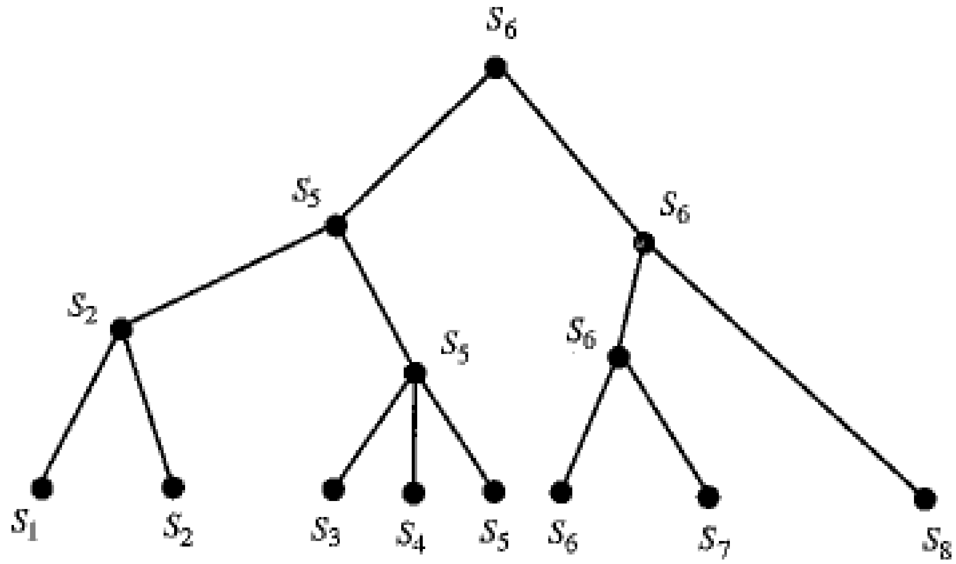

Figure 4.4: A phylogenetic tree with lifted alignment. Each internal node is labeled by one of the strings labeling its children.

For each node v let its T^* label be S_v^*. We shall assign every v a label S_v^L. Initially, only the leaves are labeled, and by definition $S_v^L = S_v^*$ for each leaf v. The labeling process 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. labeled by one of the labels of its children. Thus, the resulting phylogenetic alignment is lifted. (see figure 4.5)

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:

$D(S_v^*, S_j) \leq D(S_v^*, S_i)$

Label S_v^* with S_j

end while

end

Theorem 4.3 *(Jiang, Wang and Lawler, 1996 [6]) The distance of the phylogenetic alignment $T^L = Lift(T^*)$ is at most twice the distance of the optimal phylogenetic alignment T^*.*

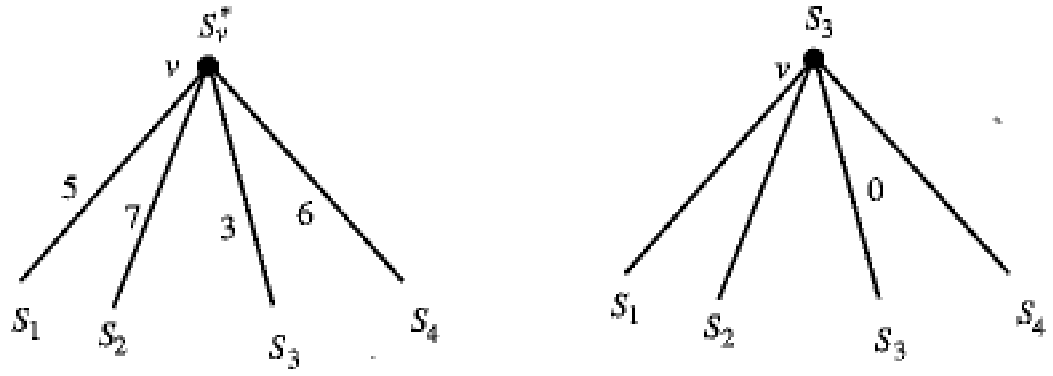

Figure 4.5: The lifting construction at node v. The numbers on the edges are the distances from S_v^* to the lifted strings labeling its children. On the left is the tree before lifting, and on the right the result of the lift. After the lift one edge will have a distance of 0.

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:

$$D(S_i, S_j) \leq D(S_j, S_v^*) + D(S_v^*, S_i) \leq 2 \cdot D(S_v^*, S_i) \tag{4.6}$$

The first inequality is due to the triangle inequality, and the second follows from the labeling 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 labeled S_i. Due to the triangle inequality

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

We say that the edge $e = (v, w)$ is *blue* in T^L if $S_i \neq S_j$. 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 labeled S_i, and no node outside P_e is labeled 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 = the total distance on blue edges $\leq 2\cdot$ the sum of all total distances of P_e paths in $T^* \leq 2\cdot$ the total distance of T^*. (see figure 4.6) ■

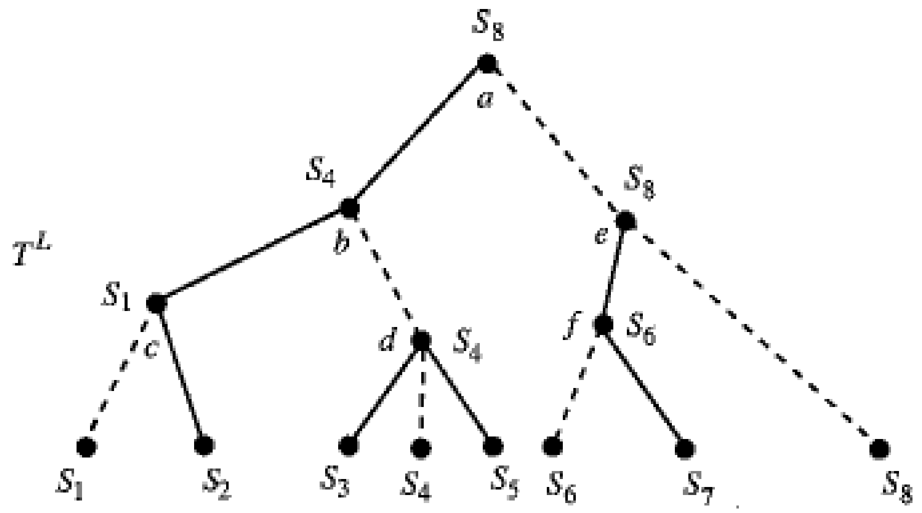

Figure 4.6: The lifted tree T_L^*. The dashed edges show the paths along which a leaf string has been lifted to some internal node, thus their distance is 0. Solid edges are blue edges in T_L^*. The path $P_{(a,b)}$ for example, is the path b, d, S_4 along which the string labeling b was lifted. Edge (a, b) has distance in T_L^* at most twice the distance of path $P_{(a,b)}$ in T^*.

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

Definition 4.11 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 it's 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 $T_{v'}$.

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 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 searching with a single sequence, we can only look for the sequences in the database with the best alignments with the given sequence, while when using a profile we may test for membership to the family.

Definition 4.12 For an alignment S' of length l, a *profile* is a $l \times |\Sigma \cup \{-\}\,|$ 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 it's occurences 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 a usual pairwise dynamic programming alignment to compare the given sequence to the family profile.

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

In this algorithm (Feng and Doolittle, 1987 [2]), the key idea is that the pair of strings with minimum distance is almost likely to having 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 (Fitch and Margoliash, 1967 [3]) to construct a tree from the distances.
3. Traverse the nodes in their order of addition to the tree, repeatedly align the child nodes (sequences or alignments).

Features of this heuristic:

- Highest scoring pairwise alignment determines the alignment to two groups.
- "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 [5]). The basic idea is the same as in the Feng-Doolittle algorithm:

1. Calculate the $\binom{k}{2}$ pairwise alignment scores, and convert to distances.
2. Use a clustering algorithm (Neighbor-Joining) 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.

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.*, 60:351–360, 1987.

[3] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. 155:279–284, 1967.

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

[5] D. G. Higgins J. D. Thompson 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.

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