14.8. Multiple alignment to a (phylogenetic) tree

As mentioned in the discussion of weighted *SP* alignment, it is often desirable to use known evolutionary history to influence the multiple alignment computed for a set of strings. The method should be encouraged to align more closely those sequences from closely related organisms. Evolutionary history is most often represented by an *evolutionary tree* where known sequences of (extant) organisms are represented at the leaves of the tree, and their unknown ancestors are represented at internal nodes of the tree. When the tree is known (from previous data and deductions) the problem is to deduce sequences for the internal nodes to optimize an objective function defined below. Once these sequences are in hand, one can find a multiple alignment consistent with the labeled tree and then remove the deduced sequences from that multiple alignment. Alternatively, the labeled tree itself gives a deduced picture of the evolutionary history that led to the known (leaf) sequences. The latter use for phylogenetic alignment will be discussed in Section 17.6.

Definition Given an input tree T with a distinct string (from a set of strings $\mathcal{S}$) written at each leaf, a *phylogenetic alignment* for T is an assignment of one string to each internal node of T. Note that the strings assigned to internal nodes need not be distinct and need not be from the input strings $\mathcal{S}$.

The phylogenetic tree, T, is meant to represent the "established" evolutionary history of a set of taxa (read "objects of interest"), with the convention that each extant taxon ("object") is represented at a unique leaf of T. Each edge (u, v) represents some mutational history that transforms the string at u (assuming u is the parent of v) to the string at v. The cost of that transformation is given by the edit distance between those two strings, and so the cost of the phylogenetic alignment is the sum of all those edge costs. This is now stated more formally.

Recall that $D(S, S')$ denotes the edit distance between string S and string S'. All that is assumed about the function D is that it obeys the usual triangle inequality conditions.

Definition If strings S and S' are assigned to the endpoints of an edge (i, j), then (i, j) has *edge distance* $D(S, S')$. The distance along a path is the sum of the distances on the edges in the path. The distance of a phylogenetic alignment is the total of all the edge distances in the tree.

The phylogenetic alignment problem for T Find an assignment of strings to internal nodes of T (one string to each node) that minimizes the distance of the alignment.

The phylogenetic alignment problem, also called the *tree alignment problem* was developed principally by Sankoff [387, 388]. We use the term "phylogenetic alignment" instead of "tree alignment" because the latter term is also used for a very different problem in the string literature [117].

Note that the consensus string problem studied in Section 14.7 is just a special case of the phylogenetic alignment problem (i.e., when tree T is a star). The connection between phylogenetic alignment and the consensus problem was first observed in [20]. In a similar manner, the algorithm to be presented here for the phylogenetic alignment problem will be a natural generalization of the algorithm developed for the *SP* and consensus objective functions.

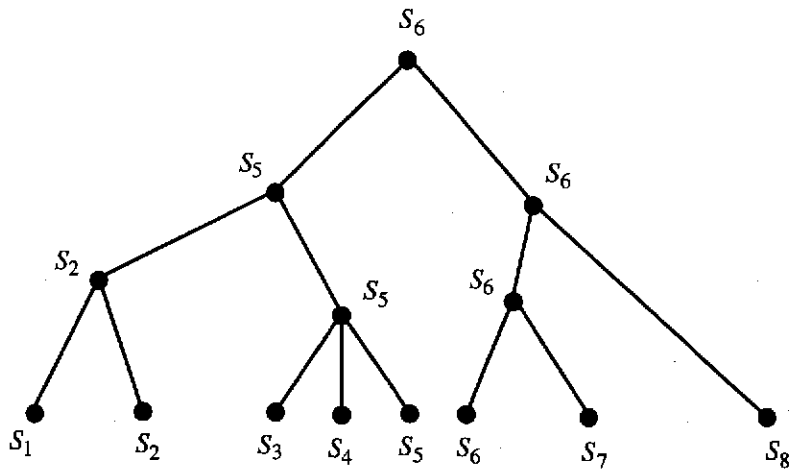

Figure 14.8: An abstract lifted alignment. Each node has the abstract name of a string written by it.

14.8.1. A heuristic for phylogenetic alignment

The general phylogenetic alignment problem is *NP-complete* [454], but it has an exponential-time solution via dynamic programming [387]. A limited version of the problem is approached heuristically in [457]. Here we discuss an efficient heuristic method that was developed and analyzed by Wang, Jiang, and Lawler [249]. Their method produces phylogenetic alignments whose distance is never more than twice the distance of the minimum alignment. However, our analysis follows a simplification of the original proof, suggested by Mike Paterson. For this result, we only assume that the edit distance function D satisfies the triangle inequality. The factor of two result also leads to a polynomial approximation scheme allowing a trade-off between computation time and guaranteed accuracy [249]. A newer result along those lines leads to a practical approximation scheme where a deviation of at most 1.58 can be guaranteed for protein-length strings [453].

Definition A phylogenetic alignment is called a *lifted alignment* if for every internal node v, the string assigned to v is also assigned to one of v's children.

For example, the solution in Figure 14.8 is a lifted alignment. Clearly, each node v in a lifted alignment is assigned a string that labels a leaf in the subtree rooted at v.

We will show that the best lifted alignment in T has a total distance less than twice that of the optimal phylogenetic alignment. We do this by constructing a particular lifted alignment T^L with that property. T^L is found by transforming the optimal phylogenetic alignment to the lifted alignment T^L. Later we show that the best lifted alignment can be computed efficiently by dynamic programming. Because the best lifted alignment has total distance at most that of T^L, one can efficiently find a lifted alignment with total distance less than twice that of the optimal phylogenetic alignment.

The transformation creating T^L

Let T^* be the optimal phylogenetic alignment for tree T. We will transform T^* into a lifted alignment T^L by a series of string replacements at internal nodes of T. This transformation is only conceptual since we do not know T^*, but it serves to define the lifted alignment T^L.

We say that a node has been *lifted* after it has been labeled by a string in the leaf set S. By definition, each leaf begins lifted. The lifting process will successively "lift" each of the internal nodes of T in any order, provided that a node is lifted only after all of

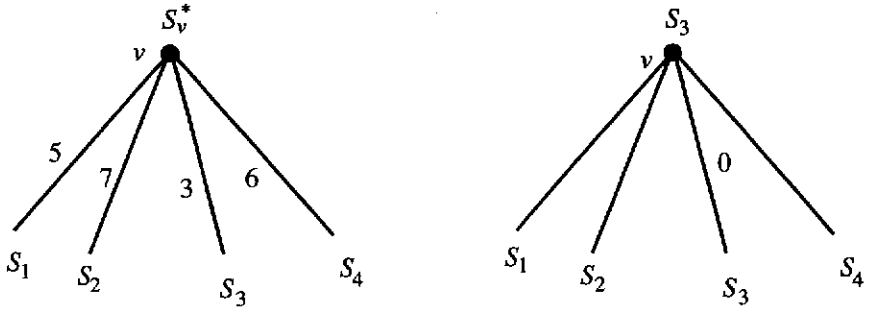

Figure 14.9: The lifting operation at node v. The numbers on the edges are the distances from S_v^* to the lifted strings labeling its children. Note that after the lift, one edge will have zero distance.

its children have been lifted. The tree that results at the end of the process is the lifted alignment T^L.

We lift a node v as follows: Let S_v^* be the string labeling internal node v in T^*. Without loss of generality, assume that v's children have been labeled with the lifted strings $S_1, S_2, \ldots, S_k$ from $\mathcal{S}$, and we refer to v's children by their string labels. Let S_j be the string from among v's children that is closest to S_v^*. That is, $D(S_v^*, S_j) \leq D(S_v^*, S_i)$ for any i from 1 to k. To lift v, replace string S_v^* by S_j (see Figure 14.9). This changes the distance on each edge out of v to $D(S_j, S_i)$. In particular, the distance becomes zero on the edge between v and its child labeled S_j.

The error analysis

Theorem 14.8.1. *The lifted alignment T^L has total distance less or equal to twice that of the optimal phylogenetic alignment T^* of T.*

PROOF Let $e = (v, w)$ be any edge in T, where v is the parent of w. Suppose that in T^L, node v is labeled with string $S_j \in \mathcal{S}$ and node w is labeled with string $S_i \in \mathcal{S}$. If $S_j = S_i$, then edge e has a zero-length edge in T^L and so we will not be concerned with it. But if $S_j \neq S_i$, then the distance of edge e in T^L is $D(S_j, S_i) \leq D(S_j, S_v^*) + D(S_v^*, S_i) \leq 2 \times D(S_v^*, S_i)$. The first inequality is due to the triangle inequality. To see the second inequality, note that at the point in the lifting process that v becomes labeled, S_i labeled a child (w) of v and hence was a candidate to label v. So $D(S_v^*, S_j)$ must be no larger than $D(S_v^*, S_i)$, and so $D(S_j, S_i) \leq 2 \times D(S_v^*, S_i)$. Now consider the path in T^* from v to the leaf labeled S_i. Denote that path by P_e. By the triangle inequality, $D(S_v^*, S_i)$ is at most the *total* of the edge distances on P_e in T^*. Hence the distance of edge e in T^L is at most twice the total path length of P_e. Note that path P_e is only defined for an edge e that has a nonzero distance in T^L.

For a nonzero-length edge $e = (v, w)$, path P_e is the path along which the leaf string S_i was lifted to node w in T^L (see Figure 14.10). Hence, every node on P_e, except v, is labeled by S_i, and no node outside of P_e is labeled with S_i. So if $e' = (v', w')$ is any other nonzero-length edge, and $P_{e'}$ is the path along which a leaf string was lifted to w', then P_e and $P_{e'}$ have no edges in common (again, see Figure 14.10). This defines a mapping from each nonzero-length edge e in T^L to a path P_e in T^* such that: a. the distance in T^L of edge e is at most twice the total distance in T^* of the edges on P_e and b. no edge in T^* is mapped to by more than one edge in T^L. Further, if the root is labeled in T^L with string $S \in \mathcal{S}$, then no edge in T^* on the path from the root to leaf S is mapped to by any edge in T^L. Therefore, the total distance of the lifted alignment T^L is less than or equal to twice the total distance of the optimal phylogenetic alignment T^*. □

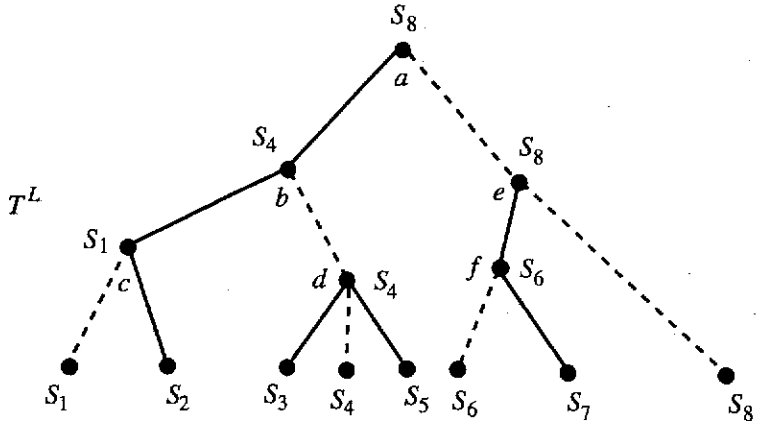

Figure 14.10: Lifted tree T^L. The dashed edges show the paths along which a leaf string was lifted to some internal node. In T^L each of these dashed edges has zero length. Path $P_{(a,b)}$ (as an example) is the path b, d, S_4 along which the string labeling b was lifted. Edge (a, b) has distance in T^L at most twice that of path $P_{(a,b)}$ in T^*.

The proof can be extended to show strict inequality. We can now state the main result.

Corollary 14.8.1. The best lifted alignment has total distance less than twice that of the optimal phylogenetic alignment T^*.

Computing the minimum distance lifted alignment

The best lifted alignment is computed by dynamic programming.

Definition Let T_v be the subtree of T rooted at node v. 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 (assuming of course that S is a string at a leaf of T_v).

After the computation is finished, if r represents the root node, then the value of the best lifted alignment is the minimum of $d(r, S)$, where S ranges over all strings written at the leaves of T. The dynamic programming algorithm computes the values of $d(v, S)$ from the leaves up. A node will be processed only after all of its children have been processed. The algorithm starts with the assumption that all the leaves have already been processed. If v is an internal node all of whose children are leaves, then the algorithm sets $d(v, S) = \sum_{S'} D(S, S')$, where S' is the string written at child of v.

For a general internal node v, the dynamic programming recurrence is

$$d(v, S) = \sum_{v'} \min_{S'}[D(S, S') + d(v', S')],$$

where v' is a child of v and S' is a string at a leaf in tree $T_{v'}$.

It should be apparent that these recurrences are correct and that the algorithm correctly finds the distance of the best lifted phylogenetic alignment. To actually assign the strings at the internal node, the algorithm must do a traceback through the recurrences (or pointers) in the usual dynamic programming way.

For the time analysis, let k be the number of leaves of T and assume that all the $\binom{k}{2}$ distances between pairs of strings are computed in a preprocessing stage. That preprocessing takes $O(N^2)$ time, where N is the total length of all the k strings. Then the work at any internal node is $O(k^2)$, and the overall work of the algorithm is $O(N^2 + k^3)$. In the exercises, we suggest an improvement that reduces the running time to $O(N^2 + k^2)$. In summary, we have

Theorem 14.8.2. *The optimal lifted alignment can be computed in polynomial time as a function of size of the tree and the lengths of the input strings.*

Hence a phylogenetic alignment whose distance is within a factor of two of the optimal can be computed in polynomial time. This phylogenetic alignment will be the center star alignment discussed earlier for the *SP* and consensus objective functions.

14.9. Comments on bounded-error approximations

Three bounded-error approximation methods were presented in the previous sections. Additional bounded-error methods will be presented in Part IV. For the reader who is not familiar with bounded-error approximations, it may be worthwhile emphasizing a few points.

First, Theorem 14.6.2 says that the alignment $\mathcal{M}_c$ will have a score that is never more than twice that of the optimal *SP* score. It does *not* say that the score of $\mathcal{M}_c$ *will* be twice the optimal (either always or typically) or even that it *could* be twice that of the optimal score. In fact, in limited tests of the center star method, the score of $\mathcal{M}_c$ deviated from the bound $\sum_{i<j} D(S_i, S_j)$ (and hence from the optimal score) by amounts between 2 and 16% [201].

Second, bounded-error approximation methods are not always the most effective or practical methods, nor are they necessarily intended to be used as "stand-alone, off-the-shelf" methods. Rather, they are often theoretical devices used to establish *provable* claims about the behavior of algorithms on hard problems. However, bounded-error methods can lead to, or be a part of, very effective methods whose behavior cannot be easily proven. One way is in combination with local improvement methods, which take a solution and iteratively look for small changes that improve it. Another way is in combination with branch-and-bound methods, where the result of the approximation method can be used both as an upper bound and to create a lower bound on the score of the optimal solution.[11] Similarly, approximation methods can sometimes be used to establish guaranteed bounds in practice for methods that lack guaranteed bounds in general but are claimed to be superior. Suppose an approximation method truly is "bad", meaning that its results are often close to its guaranteed worst-case bound, and some other heuristic is truly "good", meaning that its results are often close to the optimal. Then, on specific problem instances, the ratio of the bad value to the good value will be close to the guaranteed approximation bound, *proving* that the good solution truly deviates from the optimal by a small amount. In this manner, an approximation method with a guaranteed error bound will always be a winner – if it generally produces solutions better than other methods, then it wins the upper bound race; if it generally produces solutions much worse than other methods, then it wins too by providing a *proven* bound on the goodness of the better solution. An example, in the case of phylogenetic alignment, is developed in the exercises and is studied more deeply in [206].

Third, bounded-error methods with *fixed* error bounds are often improvable to methods that have a *provable trade-off* between bounded-error and running time. With such methods, called *polynomial time approximation schemes*, the user knows the amount of

[11] Some bounded-error methods have the desirable feature that for any specific instance of the problem, the algorithm computes a nonobvious *lower bound* (as well as an upper bound) on the value of the optimal solution. This is not true of the center-star method, where the lower bound used is immediate and unrelated to the upper bound computation.

running time that will be sufficient to achieve any particular level of confidence (i.e., any chosen limit on the maximum possible deviation from the optimal). Such a method has been developed for the *SP* alignment problem by Bafna, Lawler, and Pevzner [37]. That method aligns k strings and produces a multiple alignment whose *SP* score can never exceed $2 - q/k$ times the optimal *SP* score, for any chosen q. For any fixed value of q, the running time of the method is polynomial in n (the assumed string length of each string) and k. As q increases, so does the *provable accuracy* of the result, but the worst-case running time also increases. Approximation schemes have also been developed for the phylogenetic alignment problem [249, 453].

Finally, bounded-error approximation results sometimes evoke an "ignorance-is-bliss" reaction. That reaction is characterized by the attitude that a proven bound on the maximum possible error is a strike against an algorithm (rather than a point in its favor), unless the proven bound is very small. In its extreme form, the attitude is that a heuristic method whose maximum possible deviation from the optimal is unknown is better than one whose maximum possible deviation is known, unless the proven bound is very small. This attitude may again be a result of confusing a limit on the maximum possible error for a claim that the *typical* error is at that limit. Stated in this extreme cartoon form, the ignorance-is-bliss attitude seems unlikely to occur, but in fact it does (even in the computer science community). In contrast, the viewpoint here is that an algorithm with a proven error bound is a good thing to obtain, even though the algorithms with the best proven bounds are not always the most effective algorithms in practice.

14.10. Common multiple alignment methods

The center-star alignment algorithm was developed as a bounded-error heuristic and is not widely used by practitioners. In this section we focus on ideas embodied in methods that are widely used in practice for multiple alignment.

There are numerous multiple alignment methods (and programs) that have been published in the computational biology literature, and hundreds (if not thousands) of papers report the results of multiple alignments computed from molecular data of interest. Two papers, [92] and [316], give a starting point for examining some of the specific methods in use. We will not attempt a survey here. But virtually all of the methods used in practice are variants of one of two ideas (or a mixture of them), and so it is worth discussing those two ideas in general terms, with some details provided by specific illustrations. After that, we will also briefly mention two different, more recent ideas.

14.10.1. Iterative pairwise alignment

Most of the efforts in global multiple alignment follow a general strategy of iteratively merging two multiple alignments of two subsets of strings into a single multiple alignment of the union of those subsets. In the simplest form, this approach uses pairwise alignment scores to iteratively add one additional string to a growing multiple alignment. What differs between the methods are the criteria for selecting the pair of alignments to merge (and hence the order in which the merges are done) and the style of alignment used to create the new merged alignment.

The simplest example of this approach starts by aligning the two strings whose edit distance is minimum over all pairs of strings. Then the method successively merges in the string with smallest edit distance from any of the strings already in the multiple alignment.