1 Multiple String Alignment

Efficient methods for multiple sequence alignment with guaranteed error bounds

Dan Gusfield [1]
Computer Science Division
University of California, Davis

July, 1991

Abstract

Multiple string (sequence) alignment is a difficult and important problem in computational biology, where it is central in two related tasks: finding highly conserved subregions or embedded patterns of a set of biological sequences (strings of DNA, RNA or amino acids), and inferring the evolutionary history of a set of taxa from their associated biological sequences. Several precise measures have been proposed for evaluating the goodness of a multiple alignment, but no efficient methods are known which compute the optimal alignment for any of these measures in any but small cases. In this paper, we consider two previously proposed measures, and give two computationaly efficient multiple alignment methods (one for each measure) whose deviation from the optimal value is *guaranteed* to be less than a factor of two. This is the novel feature of these methods, but the methods have additional virtues as well. For both methods, the guaranteed bounds are much smaller than two when the number of strings is small (1.33 for three strings of any length); for one of the methods we give a related randomized method which is much faster and which gives, with high probability, multiple alignments with fairly small error bounds; and for the other measure, the method given yields a non-obvious *lower bound* on the value of the optimal alignment.

2 Introduction

Multiple string (sequence) alignment is a difficult problem of great value in computational biology, where it is central in two related tasks: finding highly conserved subregions or embedded patterns of a set of biological sequences (strings of DNA, RNA or amino acids); and inferring the evolutionary history of a set of taxa from their associated biological sequences. In the first case, a conserved pattern may be so dissimilar or dispersed in the strings that it cannot be detected by statistical tests when just two strings of the set are aligned, but the pattern becomes clear and compelling when many strings are simultaneously aligned. Scores of papers have been written on methods for multiple string alignment, and hundreds

[1]Research partially supported by grant DE-FG03-90ER60999 from the Department of Energy, and grant CCR-8803704 from the National Science Foundation.

of papers have used various multiple alignment methods to find patterns or build evolutionary trees from biological sequence data. The following few papers illustrate this broad literature: [6, **?**, 2, 4, **?**, 11, 15, **?**, 1, 10, 3].

Many of the suggested methods build a multiple alignment by attempting to optimize some explicitly or implicitly stated measure of goodness of the alignment. However, no single measure or objective function has yet been proposed that is widely agreed upon (unlike the case of aligning just two strings), and some proposed methods build alignments without relying (even implicitly) on any measure of goodness.

In this paper we consider two previously proposed ways to measure the goodness of a multiple alignment, which correspond to the two general uses of multiple alignments introduced above. For both measures, no efficient methods are known to find the optimal alignment, and so the known methods are either heuristic (not guaranteed to find an optimal alignment) or are usable only for a small number of short strings. For one measure, the best method known to compute the optimal alignment has a worst case (and typical) running time on the order of the product of the lengths of the strings to be aligned, although some ideas have been developed which reduce the typical running times by a constant factor [4]. For the the second measure, a method which solves a special case of the problem runs in exponential time [11], although a more efficient algorithm has been developed for an extremely restricted version of the problem [1].

A common approach in the computer science literature to computationaly hard optimization problems is to develop fast heuristic algorithms whose maximum possible deviation from the optimal solution can be *proven* to be bounded by a small multiplicative factor. Generally, any factor of two or less has been of interest. For the multiple string alignment problem, no bounded deviation heuristics have been reported. In this paper we adapt known heuristics and bounds from related graph theoretic problems [**?**, 9, 7] to provide the first such methods and bounds for the multiple string alignment problem.

2.1 Main results

We discuss two computationaly efficient multiple alignment heuristics (one for each objective function) whose deviation from the optimal value is guaranteed to be less than a factor of two. That is, the heuristics give alignments whose value is guaranteed never to be more than twice the value of the optimal multiple alignment. For both objective functions the guaranteed bounds are even smaller when the number of strings is small (1.33 for three strings of any length), and for one of the objective functions the method yields a non-obvious *lower bound* on the value of the optimal solution. For one of the methods we give a much faster randomized method whose likely deviation from the optimal is surprisingly small.

The error bound of two may at first seem too large to be of use, but the reader should remember that the bound is a *worst case guarantee* and the actual deviation from optimal for any particular set of strings can be expected to be much less. This is illustrated by initial tests run on the method. In fact, the analysis used to obtain the bounds makes several worst

case assumptions that are unlikely to occur naturally. Further, the bounded error methods have several indirect uses other than the direct use of producing "good" alignments, and can often form the basis of more ad hoc methods to improve the solution.

Although we can expect the method to obtain solutions that are better than twice the optimal on typical data, the main thrust of the present paper is theoretical – to establish provable bounds and to introduce the bounded error line of reasoning into this area. No comprehensive tests of the methods given here have been made against other existing methods.

2.2 Basic Definitions

An alignment of two strings X and Y is obtained by first inserting chosen spaces into, or at either end of, X and Y and then placing the two resulting strings one above the other so that every character or space in either string is opposite a unique character or a unique space in the other string. Two opposing identical characters form a *match*, and two opposing nonidentical characters from a *mismatch*. A space in one string opposite a character x in the second string can also be thought of as a *deletion* of x from the second string, or an *insertion* of x into the first string.

For example, in the alignment

$$\begin{array}{cccccccc} c & a & c & \_ & d & b & d \\ c & a & w & x & \_ & b & \_ \end{array}$$

of strings *cacdbd* and *cawxb*, character c is mismatched with w, both d's and the x are opposite spaces, and all other characters are in matches.

For a given alignment $\mathcal{A}$, let l denote the (equal) length of the two strings in $\mathcal{A}$, i.e., after spaces have been inserted. The *value* of alignment $\mathcal{A}$, denoted $V(\mathcal{A})$, is defined as $\sum_{i=1}^{l} s(X(i), Y(i))$, where $s(X(i), Y(i))$ is the value contributed by the two opposing characters (either of which could be a space) in position i of $\mathcal{A}$. This definition allows $s(X(i), Y(i))$ to depend on exactly what the two characters are, and there are several such character-pair weighting schemes for amino acids and for DNA [14, 8]. A simple, but common, scheme is to score a zero for a match or for two opposing spaces[2], and score a one for either a mismatch or for a character opposite a space. With this scheme, the above alignment has value four.

In this paper we don't assume any particular scoring scheme, but assume only that two opposing spaces have a zero value, and that the other values satisfy triangle inequality. That is, for any three characters x, y, z $s(x, z) \leq s(x, y) + s(y, z)$. This is the standard assumption, and reasonable because we interpret $s(x, z)$ as the "cost" to transform character x to character z.

Given a scoring scheme, the *optimal* alignment of two strings is an alignment $\mathcal{A}$ which *minimizes* $V(\mathcal{A})$ over all alignments of the two strings. The optimal $V(\mathcal{A})$ is also referred

[2]It is more common in defining pairwise alignment to simply forbid opposing spaces, but they occur in multiple alignments, so we allow them here for consistency.

to as the *(weighted) edit distance* between the two strings. We define $D(X, Y)$ to be the value of the optimal alignment between strings X and Y. For strings of length n and m, $D(X, Y)$ can be computed in $O(nm)$ worst case time by dynamic programming [12], a fact discovered independently many times.

2.3 Definition of multiple alignment

A multiple alignment of $k > 2$ strings $\mathcal{X} = \{X_1, X_2, \ldots, X_k\}$ is a natural generalization of the pairwise alignment defined above. Chosen spaces are inserted into (or at either end of) each string so that the resulting strings have the same length, defined to be l, and 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.

The value of a multiple alignment is not so easily generalized. Corresponding to the two general uses for multiple alignment mentioned above, we consider two rather different objective functions, called *SP* and *TA* respectively, that have been proposed to evaluate the value of a multiple alignment. The rational for these two approaches has been discussed in some depth in [1], [**?**], and [4]. We will define the two objective functions in separate sections below.

3 Multiple alignment with objective function SP

For the purpose of finding highly conserved subpatterns and for the purpose of clustering strings by similarity as a first step in constructing an evolutionary tree, the value of a multiple alignment $\mathcal{A}$ has been taken, for example in [4], to be the *sum* of the values of pairwise alignments induced by $\mathcal{A}$. This is called the *SP* value. The induced pairwise alignment of two strings is exactly their alignment given in $\mathcal{A}$ (although any two opposing spaces can be removed if desired). The *SP* measure is used in the multiple alignment package MACAW [13] developed at the National Institutes of Health, National Center for Biotechnology Information. It was also used in [10] and [3] and a similar measure was used in [6].

In this section we discuss an efficiently computed alignment that is guaranteed to have no more than twice the optimal *SP* value.

The center star method

Given a set of k strings $\mathcal{X}$, we define the *center* string $X_c \in \mathcal{X}$ as that string which minimizes $\sum_{j \neq c} D(X_c, X_j)$, and let M denote that minimum sum. We define the *center star* to be a star tree of k nodes, with the center node labeled X_c and with each of the $k - 1$ remaining nodes labeled by a distinct string in $\mathcal{X} - X_c$.

It is folklore, and used for example in [6], that given any tree T where each node is labeled with a distinct string, there is a multiple alignment $\mathcal{A}(T)$ of these strings which is "consistent" with the optimal pairwise string alignments corresponding to the edges of T. That is, if X_i and X_j are strings that label any two adjacent nodes of T, then the pairwise

alignment of X_i and X_j induced by $\mathcal{A}(T)$ has value exactly $D(X_i, X_j)$. This is clearly not necessarily true for the induced alignment of two strings that are not adjacent in T.

We define the multiple alignment $\mathcal{A}_c$ of the set of strings $\mathcal{X}$ to be the alignment derived from and consistent with the center star. For completeness of this report, details of how to construct $\mathcal{A}_c$ are given in the appendix.

We define $d(X_i, X_j) \geq D(X_i, X_j)$ as the value of the pairwise alignment of strings X_i and X_j induced by $\mathcal{A}_c$, so that value of the alignment is $V(\mathcal{A}_c) = \sum_{i<j} d(X_i, X_j)$. We will show that $V(\mathcal{A}_c)$ is at most twice the value of the optimal multiple alignment of $\mathcal{X}$.

Lemma 3.1 *For any strings X_i and X_j, $d(X_i, X_j) \leq d(X_i, X_c) + d(X_c, X_j) = D(X_i, X_c) + D(X_c, X_j)$.*

Proof Consider any single column in the multiple alignment and let x, y and z be the three characters in this column from strings X, Y and Z. By 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 X_i and X_c induced by $\mathcal{A}_c$ is an optimal alignment of X_i and X_c, and this is true also for the alignment of X_c and X_j. □

Let A^* be the optimal multiple alignment of the k strings $\mathcal{X}$, and let $V(A^*)$ denote its value. Let $d^*(X_i, X_j)$ be the value of the pairwise alignment of strings X_i and X_j induced by A^*. Then $V(A^*) = \sum_{i<j} d^*(X_i, X_j)$.

Theorem 3.1 $V(\mathcal{A}_c)/V(A^*) \leq 2(k-1)/k < 2$.

Proof First, define $v(\mathcal{A}_c) \equiv \sum_{(i,j)} d(X_i, X_j)$ and $v(A^*) \equiv \sum_{(i,j)} d^*(X_i, X_j)$, where the pair (i, j) is an *ordered* pair in each case. Clearly, $v(\mathcal{A}_c) = 2V(\mathcal{A}_c)$ and $v(A^*) = 2V(A^*)$, so $V(\mathcal{A}_c)/V(A^*) = v(\mathcal{A}_c)/v(A^*)$. It is more convenient to work with the second ratio. $v(\mathcal{A}_c) = \sum_{(i,j)} d(X_i, X_j) \leq \sum_{(i,j)} [D(X_i, X_c) + D(X_c, X_j)]$, by Lemma 3.1. For any fixed j, $D(X_c, X_j) (= D(X_j, X_c))$ shows up in this expression exactly $2(k-1)$ times. So $v(\mathcal{A}_c) \leq 2(k-1) \times \sum_j D(X_c, X_j) = 2(k-1)M$.

From the other side, $v(A^*) = \sum_{(i,j)} d^*(X_i, X_j) \geq \sum_{(i,j)} D(X_i, X_j) = \sum_i \sum_j D(X_i, X_j) \geq k \times \sum_j D(c, j) = kM$ (by the choice of X_c). So $V(\mathcal{A}_c)/V(A^*) = v(\mathcal{A}_c)/v(A^*) \leq 2(k-1)M/kM = 2(k-1)/k < 2$. □

Note that for $k = 3$ the guaranteed upper bound is 1.33. Translated into lower bounds this says that for $k = 3$, $V(A^*) \geq .76V(\mathcal{A}_c)$. 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 1.66.

Corollary 3.1 $kM \leq \sum_{i<j} D(X_i, X_j) \leq V(A^*) \leq V(\mathcal{A}_c) \leq [2(k-1)/k] \sum_{i<j} D(X_i, X_j)$.

In practice one can better measure the goodness of $\mathcal{A}_c$ by the ratio $V(\mathcal{A}_c)/\sum_{i<j} D(X_i, X_j)$. By Corollary 3.1 this ratio is always less than two, but the analysis is worst case so one can expect it to be considerably less than two in many cases. Similarly, one should expect that $V(\mathcal{A}_c)/V(A^*)$ will often be considerably less than two, since typically $\sum_{i,j} D(X_i, X_j)$ will be

considerably larger than kM, $V(A^*)$ will not generally be close to $\sum_{i<j} D(X_i, X_j)$ for any but strings which are very similar, and $D(X_i, X_j)$ will be less than $D(X_i, X_c) + D(X_c, X_j)$ for most typical strings.

Corollary 3.1 is also useful in the Carillo-Lipman algorithm [4], since that method uses $\sum_{i<j} D(X_i, X_j)$ as a lower bound on $V(A^*)$, but it also requires knowing an efficiently computed upper bound on $V(A^*)$ and does not suggest how to obtain one. By Corollary 3.1, 2(k-1)/k times the lower bound is an efficiently computed upper bound.

3.1 Faster, randomized alignments

The center method requires the computation of all $\binom{k}{2}$ optimal pairwise alignments. For large k and large strings, this may involve a great deal of computation, and so it may also be valuable to more quickly compute a multiple alignment with a "reasonable" worst case or expected error bound. Suppose one randomly selects a string X_i, then computes $D(X_i, X_j)$ for every $j \neq i$, and then builds the multiple alignment consistent with the star centered at X_i. What can be expected if p such stars are built and the best multiple alignment $\mathcal{A}$ is taken? With such a method, at most $(k-1)p$ optimal pairwise alignments need be computed. Contrary to what might seem intuitive, even when p is fairly small this approach will, with high probability, give alignments with reasonable worst case deviation from the optimal alignment. The following theorems partially capture the situation.

Theorem 3.2 *For any $r > 1$, define $e(r)$ to be the expected number of stars needed to be chosen at random before the value of best resulting alignment is within a factor of $2+1/(r-1)$ of the optimal alignment. Then $e(r) \leq r$.*

For example, $e(r)$ is at most two for an error bound of 3, and $e(r)$ is at most ten for a bound of 2.1112. Note that $e(r)$ is independent of k and of the lengths of the strings.

Proof For ease of exposition, we first prove the case for $r = 2$. For each string X_i define $M(i) = \sum_j D(X_i, X_j)$. Then $M(c) = M$. Using this notation, recall from the proof of Theorem 3.1 that $\sum_{(i,j)} D(X_i, X_j) = \sum_i M(i) \leq 2(k-1)M$, so the average value of $M(i)$ is less than $2M$. Then since the minimum value for $M(i)$ is M, it follows that the median of the $M(i)$ values is less than $3M$. The expected number of centers selected at random before a selected $M(i)$ is less than the median, is two.

Now suppose the median is actually ϵM, for $1 \leq \epsilon \leq 3$. Then $\sum_{(i,j)} D(X_i, X_j) \geq kM/2 + k\epsilon M/2$, and the value of the alignment obtained from any below median star is at most $2(k-1)\epsilon M$. Hence the error ratio for this star is at most $\frac{2\epsilon}{(1/2+\epsilon/2)}$. This ratio is maximized when ϵ is as large as allowable, i.e., when $\epsilon = 3$, where the error ratio is 3. Hence $e(2) \leq 2$.

Generalizing the above proof, we note that at least k/r stars have $M(i)$ less than or equal to $(2r-1)M/(r-1)$, which again follows from the fact that the minimum $M(i)$ is M and the mean is less than $2M$. Suppose that the point below which k/r of the $M(i)$ fall is actually ϵM for $1 \leq \epsilon \leq (2r-1)/(r-1)$. The expected number of stars to pick until one is chosen with $M(i)$ less than ϵM, is r. The error ratio of such a star is $2\epsilon/[\frac{1}{r} + \frac{r-1}{r}\epsilon]$,

which again is maximized for the largest allowable ϵ, at which point the error ratio is $(2r-1)/(r-1) = 2 + 1/(r-1)$. $\Box$

It may be more useful to put the theorem in terms of probabilities rather than expectations, since generally one is interested in how well the method might do for any fixed instance, rather than how it will do over a sequences of instances. The proof of the following is easily modified from the proof of Theorem 3.2.

Theorem 3.3 *Picking p stars (centers) at random, the best resulting alignment will have value within a factor of $2+1/(r-1)$ of the optimal with probability at least $1-[(r-1)/r]^p$.*

Theorems 3.2 and 3.3 say that one can expect to get a multiple alignment with a reasonable worst case *SP* error ratio with significantly less computation than is needed to compute $\mathcal{A}_c$, and indeed less than is used for most other multiple alignment heuristics in the literature. However, even these two theorems are too pessimistic – the analysis used in their proofs is very loose. For example, in Theorem 3.2 the case of $r = 2$ was proven by considering the median $M(i)$ value, and then setting the median to $3M$, since that is where the analysis gives the largest (hence certain) error ratio. But, if the median were actually $3M$, then the distribution of the $M(i)$ values would be known precisely: $M(i) = M$ for half the stars, and $M(i) = 3M$ for the other half. Then $\sum_{(i,j)} D(X_i, X_j) = 2kM$, the denominator in the error ratio is $2kM$, and so an *optimal SP alignment* would be obtained from any center string X_i with $M(i) = M$; such a string is selected with probability one-half. The same conclusion holds for each r. That is, were the extreme conditions used in the proof of Theorem 3.2 to actually hold, then an optimal SP alignment would be constructed from $1/r$ of the stars. So the analysis used in the proofs is quite pessimistic, and Theorems 3.2 and 3.3 should be taken as "back of the envelope" estimates which give sufficiently positive results to encourage the experimentation of randomized methods on real data of interest. This is consistent with the experimental results mentioned in the next section. If one wants complete certainty, we have the following

Theorem 3.4 *If p stars are chosen in any manner, and $\mathcal{A}$ is the best resulting multiple alignment, then $V(\mathcal{A})/V(A^*)$ is guaranteed to be less than $(k-1)/k + (k-1)/p$.*

As an interesting aside, let T be *any tree* with k nodes labeled with the strings of $\mathcal{X}$, and let $\mathcal{A}$ be the multiple alignment of $\mathcal{X}$ consistent with T. Using Theorem 2 in the paper by Wong [**?**], it can be shown that $V(\mathcal{A})/V(A^*) \leq 2k$. This provides a very quick way (since only $k-1$ pairwise alignments need by computed) to obtain a bounded error multiple alignment, but the error in this case may so large as to make the alignment uninteresting.

3.2 Comments and Empirical Results

We should point out that the above theorems, although correct, are not informative when the strings are extremely different. Let $W(X_i, X_j)$ be the value of the worst possible alignment between strings X_i and X_j. If $\sum_{i<j} W(X_i, X_j) / \sum_{i<j} D(X_i, X_j) \leq 2(k-1)/k$ then Theorem

3.1 holds vacuously. So the theorem is informative only when the strings are sufficiently similar. This is the case for many applications involving biological sequences, but probably the most interesting cases are when the strings are highly dissimilar. However, we should not conclude that the center star method is not useful for highly dissimilar strings, as illustrated in the sketch of the empirical results given below. Further, in cases when the dissimilar strings can first be grouped into subsets of more mutually related strings, then each subset can be aligned separately using the star method, and then the centers of each star can be aligned, again by the star method.

One might also object that the SP measure is based on global alignment, applying to the entire length of each string, while a measure based on local alignment would be of more use. This is most likely true, but the SP measure may still be important in multiple local alignment, as for example in the program MACAW [13]. In that program, regions of local similarly that extend throughout the strings are first found and aligned. These regions are called diagonals. Every consecutive pair of diagonals defines a set of substrings consisting of the strings between the two diagonals. These substrings are then globally aligned by MACAW, and the goodness of the entire alignment is evaluated with respect to the SP measure. Hence the need for global alignment can arise even in the context of more locally oriented objectives.

A final comment is that the center star method is similar in some ways to earlier progressive alignment methods, but quite different in one important way. Once the center star is determined, the actual alignment obtained follows the ideas of progressive alignment in that progressive alignment methods also first build, explicitly or implicitly, some tree to guide the alignment. However, the key issue is how the initial tree is found. The progressive alignment method in [6] essentially first finds a *minimum spanning tree* or, in other words, first does a single-link clustering, based on the edit distances. Algorithms to build minimum spanning trees are called "myopic algorithms" in the literature precisely because each successive decision about which link to include in the tree is made without considering the implication of that choice on possible future choices. This is consistent with the stated philosophy in [6]: "once a gap always a gap". In contrast, the center star is based on a much more global consideration of all the data. No claim is being made here that a more global approach is better than a myopic approach, just that it is certainly different.

A sketch of empirical results

We ran the above methods on several sets of biological sequences. The objective function used counted zero for a match, two for a mismatch and one for a space. Typical cases had between seven and twenty strings of lengths between 40 and 200 characters. A more complete write up of these and other experiments is forthcoming. However, in all cases, the results were considerably better than the bounds given in the above theorems, and we will give two illustrations, one where the strings were quite similar, and one with much greater variability.

We aligned 19 amino acid sequences of homeoboxs from different species. This experiment was a case where the strings were quite similar. The average string length was

60 and the average optimal (pairwise) alignment value was 25.5 with an average number of equalities in an alignment of 46. For these strings, the bound from Corollary 3.1 of $2(k-1)M/\sum_{i,j} D(X_i, X_j)$ was 1.34, and the ratio of worst possible multiple alignment value to the lower bound was greater than this, making the bound informative. As expected, the actual deviation of $V(\mathcal{A}_c)$ from the lower bound was much less: $V(\mathcal{A}_c)/\sum_{i<j} D(X_i, X_j)$ was 1.018, i.e. the multiple alignment obtained from the center star had a value whose deviation from the lower bound was less than two percent. Further, in 11 out of the 19 multiple alignments (each obtained from a different choice of center), the deviation was less than five percent. The average deviation from the lower bound for the alignments produced using centers whose $M(i)$ ranked below the median center was three percent. Generally there was a rough, but not perfect, correlation between the rank of $M(i)$ and the rank of the value of the multiple alignment produced using string X_i as center. Further, the center string, with the smallest $M(i)$, did give the best of the 19 alignments, and the string with worst $M(i)$ gave the worst multiple alignment.

To test strings which were not so similar, and where Theorem 3.1 was not informative, we took ten sequences near the homeoboxs. The average string length was 43, the average optimal pairwise alignment value was 56.5 and the average number of equalities in an optimal alignment was 13.7. The ratio $2(k-1)M/\sum_{i,j} D(X_i, X_j)$ was 1.61, but again the the actual deviation of $\mathcal{A}_c$ from the lower bound was much lower: using $\mathcal{A}_c$, the ratio was 1.162 (a 16.2% deviation from the lower bound), while the string with next best $M(i)$ gave an alignment which deviated from the lower bound by only 16.0%. Despite this, there was again a rough correlation between the rank of $M(i)$ and the rank of the alignment obtained from string X_i. Also, M was 466, while the median star had $M(i)$ equal to 504, much less than the $3M$ bound shown above. Note that the deviation of 16.0% is from the *lower bound* $\sum_{i<j} D(X_i, X_j)$ and we don't know what the actual deviation from the optimal alignment is. But since the average pairwise alignment value is large compared to the average string length, it seems unlikely that the optimal is very close to the lower bound. Hence a 16% deviation from the lower bound seems quite good.

Both of the above experiments support the belief that Theorems 3.1, 3.2, and 3.3 are generally pessimistic compared to the typical situation arsing in practice.

4 Multiple Alignment and Evolutionary Trees

One of the main uses for aligning more than two strings simultaneously is in building evolutionary trees for the taxa associated with set of biological sequences. The typical approach has been to first find a multiple alignment of the strings, then obtain distances or clusterings from that alignment to construct a tree "explaining" the evolutionary derivation of the set of strings (see [6, **?**, 5] for examples). Often, one can identify major clusters, and the pattern of evolution, by the places that long contiguous sequences of spaces have been inserted into the alignment. With this approach, the *SP* measure may be sufficient, and a close to optimal alignment may identify the same clusters that an optimal alignment would.

However, another approach is to first choose the typology of the tree and then map

the strings (with additional strings possibly added) to the nodes of the tree. The string alignment is then the alignment which is consistent (discussed in the previous section) with the pairwise alignments of the strings at the ends of the edges of the tree. The value of the alignment is just the sum of those selected pairwise alignments. This second approach to multiple alignment is called the *tree alignment (TA)* approach [1] [**?**].

The above specific approaches to building evolutionary trees connect the multiple alignment problem with the tree building problem, and the papers cited above treat the evolutionary tree problem in the context of the multiple string alignment problem. However, the main goal is the tree itself and the alignment is either part of a *method* to build the tree, or is a reflection of the goodness of the tree. Hence the goodness of the alignment is judged by the goodness of the tree associated with it. For this reason, we will focus on the tree problem, but the bounds obtained there translate of course into the alignment problem with the *TA* objective function.

4.1 Formal definitions

Let K be an input set of k strings, and let $K' \supseteq K$ be a set of strings containing (possibly equal to) K. *An evolutionary tree $T_{K'}$ for K* is a tree with at least k nodes, where each string in K' labels exactly one node, and each node gets exactly one label from K'. The value of $T_{K'}$ is $V(T_{K'}) = [\sum D(X,Y) : (X,Y)$ label the ends of an edge in $T_{K'}]$. As before $D(X,Y)$ is the value of the optimal pairwise alignment of strings X and Y. Given the set K, the problem is to find a set of strings $K' \supseteq K$ and an evolutionary tree $T_{K'}$ for K which minimizes $V(T_{K'})$ over all evolutionary trees for K.

Although the correct root (most ancestral string in K) may not be known, if the root were known and the edges of $T_{K'}$ directed away from the root, then $T_{K'}$ provides a model of the evolutionary change involved in deriving the set of strings K from the root string. The alignment value $D(X,Y)$ associated with each directed edge (X,Y) is interpreted as the minimum "cost" to transform string X to string Y, and the therefore the sum of the alignment values of the edges gives the evolutionary cost implied by the tree.

The set of strings in $K' - K$ model hypothesised ancestors of the taxa associated with K; the nodes labeled with $K' - K$ give the hypothesised historical positions of these taxa. It is easy to construct examples where the optimal (minimum value) evolutionary tree must contain such ancestors. Of course, one cannot know for sure that the "ancestors" are real, but the optimal evolutionary tree none-the-less provides the best general lower bound on the amount of evolutionary change involved in the "true" history[3], and is of course a lower bound on the best possible value obtainable by the two specific approaches to building evolutionary trees mentioned above.

Finding the optimal evolutionary tree is a very difficult computational task, and only special cases of it have been addressed in the literature [11, 1]. In this section we discuss a method which gives an evolutionary tree whose value is never more than twice that of

[3]Provided that we measure evolutionary change in terms of weighted edit distance of the associated strings.

the optimal evolutionary tree, hence never has more than twice the minimum possible evolutionary change.

4.2 Method

To describe the method, we first define the *minimum spanning tree* of a edge weighted graph. Let G be a graph with k nodes where every node is labeled with a distinct string in K. The weight given to any edge (X, Y) is $D(X, Y)$, the value of the optimal alignment of strings X and Y. The minimum spanning tree (denoted MST) of G is a subtree of G containing all k nodes, such that the sum of the weights on its edges is the minimum possible over all such subtrees of G. A minimum spanning tree of a graph can be computed very efficiently by a variety of methods [**?**]. Clearly, given a set of strings K, the MST constructed as above is an evolutionary tree for K.

For any set of strings K, let T^* denote the optimal evolutionary tree for K. We will show that $V(MST)/V(T^*) < 2$.

Let C be a traversal of the edges of tree T^* which traverses every edge exactly once in each direction. Clearly its value, the sum values of the edges it traverses is exactly $2V(T^*)$ since it traverses every edge twice. Now consider a numbering of the strings in K in the order that these strings are first encountered on traversal C. Let $C_1, C_2, ..., C_k$ be this numbering. Define $V(C)$ to be $D(C_k, C_1) + \sum_{i<k} D(C_i, C_{i+1})$.

Lemma 4.1 *For any $i < k$, $D(C_i, C_{i+1})$ is at most the sum of the values of the edges on the traversal C between string C_i and C_{i+1}. Similarly, $D(C_k, C_1)$ is at most the sum of the values of the edges on C between C_k and C_1.*

Proof Follows immediately from triangle inequality on the distance function D. □

Corollary 4.1 $V(C) \leq 2V(T^*)$.

Now let $D(C_{i^*}, C_{i^*+1})$ be the largest distance of any adjacent strings C_i, C_{i+1} including C_k, C_1.

Lemma 4.2 $V(MST) \leq V(C) - D(C_{i^*}, C_{i^*+1}) \leq V(C) - V(C)/k$.

Proof Any $k-1$ of the k pairs $\{(C_i, C_{i+1}) : 1 \leq i < k\} \cup (C_k, C_1)$ specify a set of edges which form a subtree of G containing all k nodes. In particular, the set of pairs consisting of all pairs but (C_{i^*}, C_{i^*+1}) form a spanning tree. The value of that spanning tree is exactly $V(C) - D(C_{i^*}, C_{i^*+1})$. But MST is the minimum spanning tree of G, implying the first inequality. Clearly, $D(C_{i^*}, C_{i^*+1}) \geq V(C)/k$, implying the second inequality. □

In summary we conclude,

Theorem 4.1 *For any set K of k strings, $V(MST)/V(T^*_K) \leq 2(k-1)/k < 2$.*

More exactly,

Theorem 4.2 $V(MST)/V(T_K^*) \leq (k-1)/kV(C)/V(T_K^*) \leq 2(k-1)/k.$

Generally, we can expect that $V(C)$ will be considerably less than $2V(T^*)$, and further since $V(MST) \leq V(C) - D(C_{i^*}, C_{i^*+1})$, we can also expect that $V(MST)/V(T_K^*)$ will be considerably less than two. However, unlike the case of the *SP* bound, we do not know how to compute (as opposed to prove beforehand) a better bound than that given in Theorem 4.2,

Corollary 4.2 $V(T_K^*) > kV(MST)/2(k-1)$

Corollary 4.2 gives an efficient method to compute a non-obvious *lower bound* on $V(T_K^*)$.

5 Extension to other distances

All the results established in this paper hold for more complex weight functions than discussed above. For example, an important extension is the introduction of the concept of a *gap*, a contiguous sequence of spaces in an alignment. A single evolutionary event might insert or delete a contiguous sequence of characters of quite variable length, causing a gap in an alignment of the unmodified and the derived strings. Hence it is not always correct to weigh the spaces in a gap by simply summing up the weights given by each individual space. Instead more complex gap weight functions have been suggested and studied. We will not discuss these here, but simply point out that under any gap weight function if the definition of the optimal alignment value (edit distance) satisfies the triangle inequality, then all results still apply.

6 Appendix

For completeness of this report we show how to construct the multiple alignment $\mathcal{A}_c$, which is consistent with the center tree. Without loss of generality, assume $X_c = X_1$. We will follow a method that is simple to describe, but not the most efficient method.

After computing all the optimal pairwise alignments, let s_0 be the maximum number of spaces placed before the first character of X_1 in any of the alignments, let s_f be the maximum number of spaces placed after the last character of X_1 in any of the alignments, and for each i let s_i be the maximum number of spaces placed between characters $X_1(i)$ and $X_1(i+1)$ of X_1 in any of the alignments. To create the multiple alignment $\mathcal{A}_c$ we first insert spaces into X_1. Insert s_0 spaces before X_1, s_f spaces after X_1, and s_i spaces between character $X_1(i)$ and $X_1(i+1)$ for each i. Let $\bar{X_1}$ denote the string X_1 with these spaces inserted. Then for each string X_j, find the optimal pairwise alignment of X_j with $\bar{X_1}$ with the constraint that no additional spaces are put into $\bar{X_1}$. The result is also an alignment of X_1 and X_j, so $D(X_j, \bar{X_1}) \geq D(X_j, X_1)$. Conversely, s_0 (s_f) is greater or equal to the number of spaces placed before (after) X_1 in the optimal alignment of X_1 and X_j, and each s_i is greater or equal to the number of spaces between $X_1(i)$ and $X_1(i+1)$ in the optimal

X_1, X_j alignment. Hence $D(X_j, \bar{X}_1) \leq D(X_j, X_1)$, so $D(X_j, \bar{X}_1) = D(X_j, X_1)$ for each X_j. Then since no additional spaces were inserted into $\bar{X}_1$, these pairwise alignments form a multiple alignment $\mathcal{A}_c$ which is consistent with the center tree.

Acknowledgements

Thanks to Krishna Balasubramanian and Dalit Naor for helpful comments and discussions during this research, and to John Kececioglu for helpful comments on an early draft.

References

[1] S. Altschul and D. Lipman. Trees, stars, and multiple sequence alignment. *SIAM J. on Applied Math*, 49:197–209, 1989.

[2] P. Argos and M. Vingron. Sensitivity comparisons of protein amino acid sequences. In R. F. Doolittle, editor, *Methods in Enzymology vol. 183. Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, pages 352–365. Academic Press, 1990.

[3] D. Bacon and W. Anderson. Multiple sequence alignment. *J. Mol. Biol.*, 191:153–161, 1986.

[4] H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J. on Applied Math*, 48:1073–1082, 1988.

[5] R. F. Doolittle. *Of Urfs and Orfs: A primer on how to analyze derived amino acid sequences.* University Science Books, Mill Valley, CA., 1986.

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

[7] D. Gusfield. The steiner tree problem in phylogeny. no. 334. Technical report, Yale Univeristy computer science department, 1984.

[8] T.H. Jukes and C. R. Cantor. Evolution of protein molecules. In H.N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academic Press, 1969.

[9] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *ACTA Informatica*, 15, 1981.

[10] M. Murata, J. Richardson, and J. Sussman. Simultaneous comparison of three protein sequences. *Proc. of the Nat. Academy of Science*, 82:3073–3077, 1985.

[11] D. Sankoff and R. Cedergren. Simultaneous comparisons of three or more sequences related by a tree. In D. Sankoff and J. Kruskal, editors, *Time warps, string edits,*

and macromolecules: The theory and practice of sequence comparion, pages 253–264. Addison Wesley, 1983.

[12] D. Sankoff and J. Kruskal (Eds). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparion*. Addison Wesley, Reading, Mass., 1983.

[13] G.D. Schuler, S.F. Altschul, and D.J. Lipman. A workbench for multiple alignment construction and analysis. *Proteins: Structure, Function and Genetics*, 9:180–190, 1991.

[14] R. Schwarz and M. Dayhoff. Matrices for detecting distant relationships. In M. Dayhoff, editor, *Atlas of protein sequences*, pages 353–358. National Biomedical Research Foundation, 1979.

[15] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Adv. Math.*, 20:367–387, 1976.