Optimal Multiple Parsimony Alignment with Affine Gap Cost Using a Phylogenetic Tree

Bjarne Knudsen

Department of Zoology, University of Florida, Gainesville, FL 32611-8525, USA
bk@birc.dk

Abstract. Many methods in bioinformatics rely on evolutionary relationships between protein, DNA, or RNA sequences. Alignment is a crucial first step in most analyses, since it yields information about which regions of the sequences are related to each other. Here, a new method for multiple parsimony alignment over a tree is presented. The novelty is that an affine gap cost is used rather than a simple linear gap cost. Affine gap costs have been used with great success for pairwise alignments and should prove useful in the multiple alignment scenario. The algorithmic challenge of using an affine gap cost in multiple alignment is the introduction of dependence between different columns in the alignment. The utility of the new method is illustrated by a number of protein sequences where increased alignment accuracy is obtained by using multiple sequences.

1 Introduction

An algorithm for pairwise parsimony alignment of biological sequences was devised by Needleman and Wunsch in 1970 [1]. The minimum cost of an alignment is found by a recursive algorithm in the prefixes of the two sequences. Once the matrix of optimal prefix alignment costs has been found, the optimal alignment can be found by going back through the matrix (backtracking). The cost of gaps in this algorithm is a linear function of the gap length.

The original pairwise alignment algorithm had a time and memory complexity of $O(L^2)$ (for sequences of length L). In 1975, however, Hirschberg described an algorithm for pairwise alignment that reduced the memory complexity to $O(L)$, while only increasing time consumption by a factor of two [2,3].

In 1982, Gotoh published an algorithm for pairwise sequence alignment that can use a gap cost which is an affine function of the gap length, rather than linear [4]. This was done in a way resembling the Needleman-Wunsch algorithm while keeping track of three different types of residue configurations of the last alignment step: 1) residues in both sequences, 2) a residue in sequence one aligned to a gap in the other, and 3) a residue in sequence two aligned to a gap in sequence one. The use of an affine gap cost improves the alignments of biological sequences significantly [5].

At close to the same time, Sankoff and Cedergren described an algorithm for multiple parsimony alignment over a tree with linear gap cost [6]. Other methods have used the sum of pairs scores for multiple alignments, rather than scores based on a tree [7]. This scoring scheme does not have an evolutionary explanation, which gives it some drawbacks [5]. There has also been various approximation methods proposed for multiple alignment [8,9,10].

This work gives an algorithm for optimal multiple parsimony alignment with affine gap cost for any number of sequences related by a tree. It combines ideas from pairwise alignment with affine gap cost, and ideas from multiple alignment with linear gap cost [4,6]. The memory reducing technique of Hirschberg is also applied [2]. The resulting algorithm has a memory complexity of $O(7.442^N L^{N-1})$ and a time complexity of $O(16.81^N L^N)$ (for N sequences). These complexities should be compared to $O(L^{N-1})$ and $O(2^N L^N)$, respectively, for the linear gap cost method [2,6].

Some examples are given to show the utility of the algorithm. Due to the large time and memory complexities, it is not practical to use for more than three sequences or maybe four short sequences. The method could, however, prove useful in a number of other situations, such as alignment quality evaluation and pair-HMM based probabilistic alignment, particularly when using sampling based approaches.

2 The Problem

2.1 Pairwise Parsimony Alignment

Gaps in pairwise alignments are the result of insertions and deletions (indels) of sequence pieces. The cost of indels in the affine gap cost setting is based on a gap introduction cost ($g_i > 0$) and a gap extension cost ($g_e < g_i$), giving a total cost of $g_i + (l-1)g_e$ for an indel of length l. For a sequence alphabet Σ of size n, a symmetric $n \times n$ matrix, $\{m_{ij} \geq 0\}$, gives the cost of matches for all possible pairs of residues. The cost of matching two identical residues is zero. Here is an example:

```
G C G G G T -
C C - - G T A
```

Cost: $m_{GC} + m_{CC} + g_i + g_e + m_{GG} + m_{TT} + g_i$
$= m_{GC} + 2g_i + g_e$

Notice that the total indel cost is independent of the nature of the residues, given the alignment. The optimal alignment of two sequences is the one that minimizes the total cost.

2.2 Multiple Parsimony Alignment

Assume that N sequences are given, as well as an unrooted binary tree relating them. The N sequences are located at the leaves of the tree, while the ancestral sequences are at the internal vertices. Assume for a moment that the ancestral sequences are known, so an alignment of all these sequences implies a pairwise alignment of the two sequences at the ends of each edge in the tree. For these pairwise alignments, columns with gaps in both sequences are removed. Now, the

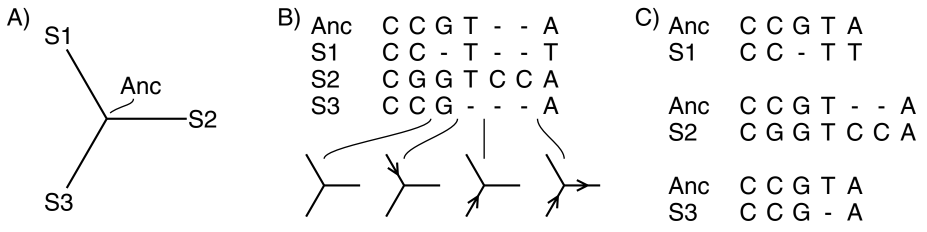


Fig. 1. A) The phylogenetic tree relating the three sequences S1, S2, and S3, and their common ancestor, Anc. B) The multiple alignment of the four sequences and indel configurations at various points (see Sect. 3.3). C) The implied pairwise alignments. The cost of the multiple alignment is $3g_i + g_e + 5m_{CC} + 2m_{GG} + 2m_{AA} + 2m_{TT} + m_{AT} + m_{CG} = 3g_i + g_e + m_{AT} + m_{CG}$.

price of the whole alignment is the sum of the costs of all the implied pairwise alignments of neighboring sequences (see Fig. 1).

If the ancestral sequences are not given, the cost of the multiple alignment of the N sequences is the minimum cost over all possible ancestral sequences and all possible alignments of the sequences. An optimal multiple alignment of the N sequences is one with minimal cost.

2.3 The Objective

Given N sequences, a phylogenetic tree relating them, a matrix of match costs, a gap introduction cost, and a gap extension cost, we seek the optimal multiple parsimony alignment of the sequences.

3 Algorithm

3.1 Residue Configurations

For a given tree, label each vertex with either a residue (#) or a gap (-), and denote that a residue configuration for the tree. Call a residue configuration *acceptable* if 1) the vertices labeled by residues form a connected graph and 2) at least one leaf has a residue (see Fig. 2).

A residue configuration will be used to define what occurs in a single column of a multiple alignment. When aligning two residues, the biological interpretation is that they are related to each other, which is the reasoning behind condition one of an acceptable residue configuration. A column with no residues in any of the sequences being aligned can not be part of an optimal alignment (since $g_i > 0$), leading to condition two for an acceptable configuration.

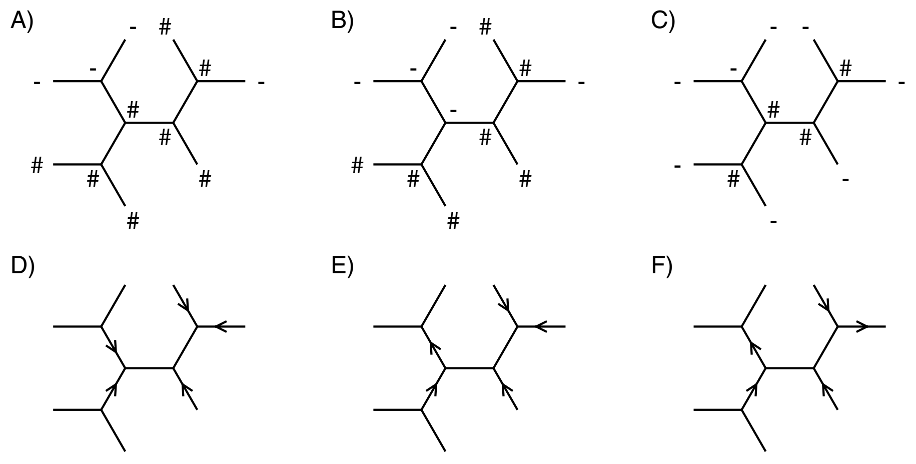


Fig. 2. The three top trees show various residue configurations, while the bottom three shows indel configurations. A) An acceptable residue configuration, since the vertices with residues form a connected subtree and some leaves have residues. B) and C) Unacceptable residue configurations. D) An unacceptable indel configuration, since some vertices are isolated from all the leaves. E) Another unacceptable indel configuration, where changing the direction of one arrow isolates some vertices from all the leaves. F) An acceptable indel configuration.

3.2 Indel Configurations

Given a tree, we can choose to label each edge in one of three ways: 1) an arrow pointing from one end to the other, 2) an arrow pointing the opposite direction, and 3) just by a line. We will call such a labeling an indel configuration for the tree.

For an indel configuration on a tree, a vertex, a, is isolated from another, b, if the path from a to b has an arrow pointing opposite to the direction of the path. An indel configuration is *acceptable* if it satisfies the following constraints: 1) no vertex is isolated from all the leaves and 2) this still holds if the direction of any single arrow is reversed (see Fig. 2).

The indel configurations are used for keeping track of ongoing indels in a multiple alignment. There is no immediate biological reasoning for the unacceptable indel configurations, but it turns out that they can never be part of an optimal alignment, see Sect. 3.4.

3.3 The Recursion

Define a prefix alignment of the N sequences as the optimal alignment of certain prefixes of the sequences. An N dimensional integer vector $\boldsymbol{v}$ defines the number of residues from each sequence that is included in the prefix alignment.

The algorithm starts with the empty prefix alignment (no residues from any sequences) and builds it up column by column, while keeping track of the minimal cost. Using this approach, the optimal cost of the entire alignment can be found, and by backtracking through the costs an optimal alignment may also be found. Notice that there could be more than one optimal alignment.

For a prefix alignment, the indel cost of an additional column will depend on which pairwise indels have been initiated, since the gap cost is affine. The indel configurations are used to keep track of ongoing indels between two sequences that are neighbors on the tree. An arrow indicates that an indel was present between the sequences at the ends of the corresponding edge in the last column involving any of these two sequences. The arrow points in the direction of the sequence that had a residue and away from the one with a gap. A line with no arrow indicates a match in the last column involving the two sequences (see Fig. 1).

Because of the dependence on past indels, we will keep track of each possible ending indel configuration for the prefix alignments. Let $E_{\boldsymbol{v}}^{(d)}$ denote the optimal cost of the prefix alignment of the sequences corresponding to the sequence vector $\boldsymbol{v}$ and indel configuration d.

The addition of a column in the alignment may change the indel configuration depending on which matches are made and which gaps are introduced and extended. For a given indel configuration, let us say that we apply a residue configuration when adding a column with the corresponding residues to give a new indel configuration (see Fig. 3). The change in indel configuration is given by one of the following three possibilities. 1) An edge with residues at both ends changes the indel configuration to a line at that edge (at no cost). 2) An edge with a residue at one end, and a gap at the other, changes the indel configuration to an arrow pointing toward the residue (at a cost of g_e or g_i if the same arrow was there already or not, respectively). 3) An edge with gaps at both ends does not change the indel configuration (and has no cost). Thus, the gap cost of applying a residue configuration to an indel configuration is the gap cost of the new alignment column, given the existing prefix alignment.

For a vector, $\boldsymbol{e}$, with coordinate values of one or zero for each sequence (including ancestors), the corresponding residue configuration is the one with residues for ones and gaps for zeros. For an indel configuration d', applying the residue configuration corresponding to the vector yields another indel configuration, d. We write this as $d' \xrightarrow{\boldsymbol{e}} d$. The gap cost of applying the residue configuration is written $G_{\boldsymbol{e}}^{(d')}$.

When going from prefix alignment $\boldsymbol{v} - \hat{\boldsymbol{e}}$ to $\boldsymbol{v}$ (with $\hat{\boldsymbol{e}}$ denoting the vector with only the N entries of $\boldsymbol{e}$ corresponding to the leaves), the residues of the sequences are known, and the optimal ancestral residues are found by a post order tree traversal giving the minimal residue match cost [11]. The match cost is written as $M_{(\boldsymbol{v}-\hat{\boldsymbol{e}})\rightarrow\boldsymbol{v}}$, which is not dependent on which ancestral sequences has residues, since matching two identical residues is free ($m_{ii} = 0$). For the alphabet, Σ, the match costs can be found for all possible residue configurations

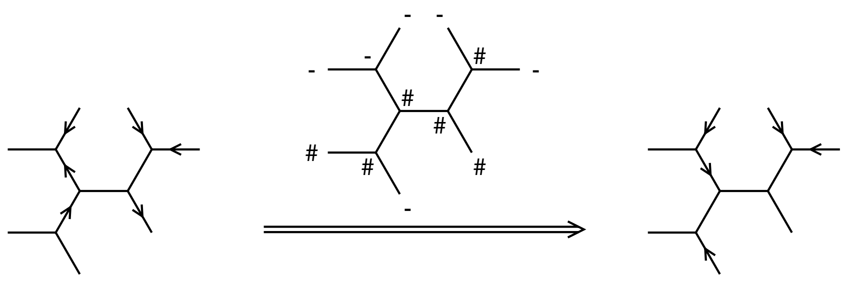


Fig. 3. A residue configuration (over double arrow) is applied to an indel configuration (left) giving a new indel configuration (right). The associated cost is $2g_i + 2g_e$.

before starting the recursion, which will have no effect on the time or memory complexity for large L.

Each acceptable residue configuration defines a possible column in the alignment, and thus a step from one prefix alignment to the next. We can write the algorithm for finding the optimal cost as this recursion:

$$E^{(d)}_{\boldsymbol{v}} = \min_{\boldsymbol{e}, d' \xrightarrow{\boldsymbol{e}} d} \{ E^{(d')}_{\boldsymbol{v}-\hat{\boldsymbol{e}}} + G^{(d')}_{\boldsymbol{e}} + M_{\boldsymbol{v}-\hat{\boldsymbol{e}}\to\boldsymbol{v}} \} \ , \tag{1}$$

with $E^{(d)}_{\mathbf{0}} = 0$ for the indel configuration, d, where all edges are labeled with a line, and $E^{(d')}_{\mathbf{0}} = \infty$ for all other indel configurations, d'. Keeping all the optimal prefix alignment scores, we can see which shorter prefix alignment each one came from. Doing this from the full alignment to the empty alignment, we obtain an optimal multiple alignment.

3.4 Reduced Resources Recursion

We saw in Sect. 3.1 that only acceptable residue configurations can be part of an optimal alignment. It will be shown here that the same applies to indel configurations.

Assume that we have an indel configuration where some vertices are isolated from all the leaves. Let a be the last of these vertices to have a residue (or one of the last if there is more than one), and consider the column in the prefix alignment where that occurred. Since only acceptable residue configurations are used, this column must have had a residue at a leaf and there would be a connected set of vertices with residues between a and the leaf. Since this is the case, a can not be isolated from that leaf immediately after this column. The only way to isolate a from the leaf is by having a later column with a residue at a and none at the leaf, which is a contradiction.

Now, assume that we have an indel configuration where no vertices are isolated from all leaves, but the reversal of an arrow at the edge α isolates some vertices from all leaves. The last column involving residues at either end of α must have had a residue at one end and a gap at the other end. The vertex, a, that had the gap must become isolated upon the reversal of the arrow at α. Thus, the last column, C_i, with a residue at a could not have had a residue at any of the leaves on the same side of the tree relative to α. Let C_j be the last column before C_i, where a leaf on this side of the tree had a residue.

The prefix alignment of all sequences (including ancestral), leading to this indel configuration, could be changed to have a gap at a, and all other vertices on that side of the tree, in the columns C_{j+1} to C_i. This would not change the alignment of the leaf sequences. Furthermore, the cost of this prefix alignment would be lower, since the match cost would be the same and it has less gap introductions or less gap extensions, or both. This means that the assumed indel configuration could never be part of an optimal alignment if $g_e > 0$. If $g_e = 0$, both configurations could be optimal, so the assumed indel configuration is again not necessary to ensure finding an optimal alignment.

With this, it has been shown that only acceptable indel configurations need to be taken into account in the recursion. Furthermore, for certain indel configurations only a subset of the acceptable residue configurations (the ones leading to acceptable indel configurations) needs to be applied.

3.5 Hirschberg's Memory Reduction

In (1), each prefix alignment cost, $E_{\boldsymbol{v}}^{(d)}$, only depends on the costs for $\boldsymbol{v}'$, where any coordinate is the same or one lower than $\boldsymbol{v}$. Denote a row of costs as a set of costs where the coordinate of $\boldsymbol{v}$ corresponding to sequence one is the same. To find the cost of the optimal alignment, we only need to keep the present and the previous row of costs in memory. This reduces the memory complexity by a factor of L. With this we can not, however, find the optimal alignment since we lost the information we need to backtrack.

Using a backwards recursion for finding the prices from the full alignment to shorter prefixes, an optimal prefix in the middle of the alignment (relative to the first sequence) can be chosen and the alignment problem can be split into two problems of smaller size. The two problems together will take at most half as much time as the original problem, so the total time will be no more than a factor $1 + \frac{1}{2} + \frac{1}{4} + \cdots = 2$ longer [2]. With this, the memory complexity has been reduced by a factor of L, while the time complexity has only increased by a factor of two.

4 Algorithm Analysis

Here, it is assumed that the tree is binary. For trees with higher vertex orders, the time and memory limits will still hold.

4.1 Memory Complexity

The memory complexity of the algorithm is in the order of L^{N-1} times the number of acceptable indel configurations for N sequences.

The indel configurations for a rooted tree can be represented by a three dimensional vector, $\boldsymbol{f}$. The first coordinate is the number of indel configurations where the root is not isolated from all leaves and reversing a single arrow does not change that. The second coordinate is the number of indel configurations where the root is not isolated from all leaves, but reversing a single arrow changes that. The third coordinate is the number of indel configurations where the root is isolated from all leaves.

The configurations for a rooted tree can be calculated from the two rooted subtrees (a and b) connected to the root as:

$$\boldsymbol{f}_{ab} = \left[\boldsymbol{f}_a \boldsymbol{A} \boldsymbol{f}_b^T \; \boldsymbol{f}_a \boldsymbol{B} \boldsymbol{f}_b^T \; \boldsymbol{f}_a \boldsymbol{C} \boldsymbol{f}_b^T\right] \ ,$$

where:

$$\boldsymbol{A} = \begin{bmatrix} 6 & 3 & 1 \\ 3 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 2 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \boldsymbol{C} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \ .$$

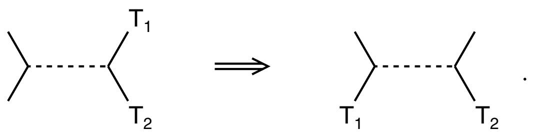
These relationships can be found by connecting the roots of the two subtrees to a vertex (the new root) with all nine combinations of labels for the two new edges.

Two rooted trees (a and b) can be connected by a single edge to give an unrooted tree. There are three configurations for the connecting edge, but only some of these lead to acceptable indel configurations for the whole unrooted tree, depending on the configurations of a and b. By going through these possibilities, the total number of acceptable indel configurations for the unrooted tree can be determined as:

$$\boldsymbol{f}_a \begin{bmatrix} 3 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \boldsymbol{f}_b^T \ , \qquad (2)$$

where $\boldsymbol{f}_a$ and $\boldsymbol{f}_b$ are the configurations for the two subtrees.

Define a linear tree as one where any vertex is at most one edge away from a leaf. Here, a proof will be sketched that for a given number of leaves, the linear tree has the highest number of acceptable indel configurations. Consider the following rearrangement of a tree, where T_1 and T_2 represent subtrees with at least two leaves and the dashed lines represent a linear connecting tree:



The number of acceptable indel configurations never decreases by such a rearrangement. This can be proven by showing that it applies for all symmetric (in the two ends) pairs of configurations for the connecting linear tree (proof left out because of space limitations). By doing a finite number of these changes, the linear tree will always be reached and it therefore has the maximal number of acceptable configurations.

For the linear rooted tree with n leaves, we can write the number of configurations of the different kinds as $\boldsymbol{f}_n$, with $\boldsymbol{f}_1 = [1\ 0\ 0]$:

$$\boldsymbol{f}_n = \left[\boldsymbol{f}_{n-1}\boldsymbol{A}\boldsymbol{f}_1^T\ \boldsymbol{f}_{n-1}\boldsymbol{B}\boldsymbol{f}_1^T\ \boldsymbol{f}_{n-1}\boldsymbol{C}\boldsymbol{f}_1^T\right] = \boldsymbol{f}_{n-1}\boldsymbol{X} = \boldsymbol{f}_1\boldsymbol{X}^{n-1}$$
$$\text{with } \boldsymbol{X} = \begin{bmatrix} 6\ 2\ 1 \\ 3\ 2\ 1 \\ 1\ 1\ 1 \end{bmatrix} .$$

By connecting the rooted tree with $n-1$ leaves to the rooted tree with one leaf, the maximal number of acceptable configurations for the unrooted linear tree with n leaves (g_n) can be found by using (2):

$$g_n = \boldsymbol{f}_{n-1} \begin{bmatrix} 3\ 2\ 1 \\ 2\ 1\ 0 \\ 1\ 0\ 0 \end{bmatrix} \boldsymbol{f}_1^T = [1\ 0\ 0]\boldsymbol{X}^{n-2} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} .$$

[3 2 1] is a linear combination of all three eigenvectors of $\boldsymbol{X}$, so g_n grows exponentially as $O(m^n)$, where m is the maximal eigenvalue of $\boldsymbol{X}$:

$$m = 3 + \frac{1}{\sqrt[3]{2}}\left(\sqrt[3]{21 + i\sqrt{59}} + \sqrt[3]{21 - i\sqrt{59}}\right) = 7.441622\ldots$$

It follows that the memory complexity of the algorithm is $O(m^N L^{N-1})$. For specific numbers of acceptable indel configurations, see Table 1.

4.2 Time Complexity

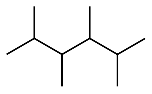
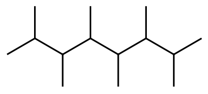
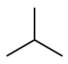
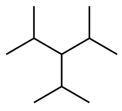
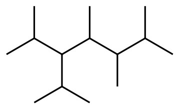
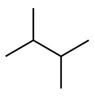
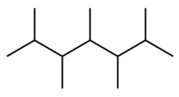
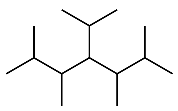
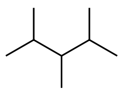
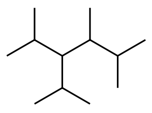
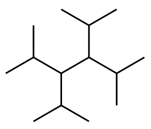
Since each acceptable residue configuration is applied to each acceptable indel configuration, the time complexity of the algorithm is at most proportional to the memory complexity times the number of acceptable residue configurations.

Here, the growth rate of the numbers of acceptable indel configurations as a function of the number of leaves will be found. First, let us look at rooted trees that has a residue at the root. For a rooted tree, a, let x_a denote the number of residue configurations with a residue at the root and where the vertices labeled with a residue form a connected graph. These configurations are acceptable, except for the possibility that no leaf has a residue.

Combining two rooted trees, a and b, to a new root vertex gives:

$$x_{ab} = (x_a + 1)(x_b + 1) ,$$

Table 1. The number of the various configurations as a function of the topology (from the top): 1) total indel configurations, 2) acceptable indel configurations, 3) total residue configurations, and 4) acceptable residue configurations

	3 3 4 3		19,683 9,534 1,024 124		1,594,323 527,985 16,384 543
	27 23 16 10		19,683 9,494 1,024 132		1,594,323 525,342 16,384 603
	243 172 64 25		177,147 70,950 4,096 263		1,594,323 524,975 16,384 620
	2,187 1,281 256 57		177,147 70,603 4,096 287		1,594,323 522,336 16,384 671

where the addition of one comes from having no residues in the corresponding subtree. Define:

$$y = \log(x + 1) \ .$$

Let x_i and y_i be the maximal x and y values, respectively, over the rooted tree topologies with i leaves. Also let $w_1 = 1$, and:

$$w_{2^{k+1}} = (w_{2^k} + 1)^2 \quad \text{and} \quad z_{2^k} = \log(w_{2^k} + 1) \quad \text{for } k \geq 0 \ .$$

By recursively forming balanced trees, it is clear that $x_i \geq w_i$ for $i = 2^k$. It follows that the growth rate of x_i is at least equal to the growth rate of w_i (when defined, i.e. $i = 2^k$). To see that the growth rate is not higher, we will show that:

$$y_i \leq \frac{2^{k+1} - i}{2^k} z_{2^k} + \frac{i - 2^k}{2^k} z_{2^{k+1}} \quad \text{for } 2^k \leq i < 2^{k+1} \ . \tag{3}$$

The right side is an affine function going through z_{2^k} for $i = 2^k$ and $z_{2^{k+1}}$ for $i = 2^{k+1}$. By showing this, we have a strict limit for the growth of x_n between the values where w_n is defined. We have:

$$\frac{w_{2^{k+1}} + 1}{w_{2^k} + 1} < \frac{(w_{2^k} + 1)^2}{(w_{2^{k-1}} + 1)^2} \quad \Rightarrow \quad \frac{z_{2^{k+1}} - z_{2^k}}{2^{k+1} - 2^k} < \frac{z_{2^k} - z_{2^{k-1}}}{2^k - 2^{k-1}} \ .$$

This means that the slope of the affine function in (3) decreases as k increases. If (3) holds, we then also have:

$$y_i \leq \frac{2^{k+1}-i}{2^k} z_{2^k} + \frac{i-2^k}{2^k} z_{2^{k+1}} \quad \text{for all } k \geq 0 \ . \tag{4}$$

Equation (3) holds for $i = 1$ since $y_1 = z_1 = \log(2)$. Assume that (3) also holds for $i \leq n$ and choose k so that $2^k \leq n+1 < 2^{k+1}$. By using (4) we have:

$$y_i \leq \frac{2^k - i}{2^{k-1}} z_{2^{k-1}} + \frac{i-2^{k-1}}{2^{k-1}} z_{2^k} \quad \text{for } i \leq n \ .$$

It follows that:

$$\begin{aligned} y_{n+1} &\leq \max_{1\leq i\leq n} \log(\exp(y_i + y_{n+1-i}) + 1) \\ &= \log\left(\exp\left(\frac{2^{k+1}-(n+1)}{2^{k-1}} z_{2^{k-1}} + \frac{(n+1)-2^k}{2^{k-1}} z_{2^k}\right) + 1\right) &(5) \\ &\leq \frac{2^{k+1}-(n+1)}{2^k} z_{2^k} + \frac{(n+1)-2^k}{2^k} z_{2^{k+1}} \ . &(6) \end{aligned}$$

The fact that (5) is less than or equal to (6) for $2^k \leq n+1 < 2^{k+1}$ can be seen by realizing that A) the two expressions are identical for $n+1 = 2^k$ and for $n+1 = 2^{k+1}$, and B) (6) is affine in n, while (5) is convex in n, since $\frac{d^2}{dx^2}\log(\exp(ax+b)+1) > 0$ for $a > 0$.

Induction leads us to conclude that (3) holds for all i. Aho and Sloane [12] have shown that the recursion $q_{i+1} = q_i^2 + 1$ with $q_0 = 1$ grows as $O(a^{2^i})$ with a given by:

$$\begin{aligned} a &= \exp\left(\frac{1}{2}\log 2 + \frac{1}{4}\log\frac{5}{4} + \frac{1}{8}\log\frac{26}{25} + \frac{1}{16}\log\frac{677}{676} + \dots\right) \\ &= 1.502837\dots \end{aligned}$$

Because of the affine relationship described in (3) and because $w_{2^k} = q_{k-1}$, we see that x_n grows as $O(a^{2n})$, where $a^2 = 2.258518\dots$.

These calculations were for rooted trees with a residue at the root and the possibility of no residue at any leaves. Due to the exponential growth rate of x_n, it can easily be shown that the contributions from the configurations with no residues at the root, and from those with no residues at any leaves, are negligible. Furthermore, the growth rate for unrooted trees is the same as for rooted trees.

This means that the time complexity of the algorithm is $O(t^N L^N)$, with $t = ma^2 = 16.807040\dots$. A slightly smaller exponential value than t is likely to apply for two reasons. 1) For a given indel configuration, only the residue configurations leading to acceptable indel configurations need to be considered. 2) The worst case was the linear tree for the residue configurations and the balanced tree for the indel configurations. It is uncertain whether there is any type of tree topology that is worst case for both types of configurations.

Much like for the indel configurations, specific numbers of acceptable residue configurations can be found by considering vectors representing different cases for rooted trees and then joining two rooted trees to form an unrooted one, see Table 1.

5 Results

The HOMSTRAD database consists of 1033 protein alignments (April 2003) which were made using structural information [13]. These alignments will be considered correct in the following analyses. Three data sets of triple alignments were made by choosing sequences with pairwise identities in the ranges [15%; 25%), [25%; 35%), and [35%; 50%) (see Table 2). Since the starting and ending points of the sequences varied, indels in the ends of the alignments were removed. As many triples as possible (16 to 18) within the identity ranges and with an average sequence length between 100 and 150 amino acids (after removing the ends) were randomly chosen from different HOMSTRAD families to form the data sets.

Table 2. The percentages of correctly aligned positions for pairwise and triple alignments of three data sets. Each set has a number of triple alignments from different families in the HOMSTRAD database [13]. The minimum and maximum pairwise identities between each pair in the alignments define each data set. The size refers to the number of alignments each set contains. Gap introduction and extension costs were chosen to optimize the number of correctly aligned positions. The percentage of correctly aligned positions by Clustal W is included for comparison [10].

		Identity	Pairwise				Triple			
Set	Size	range	g_i	g_e	Correct	Clustal	g_i	g_e	Correct	Clustal
A	18	35% – 50%	3.6	1.3	93.9%	94.1%	3.2	1.6	93.7%	94.2%
B	16	25% – 35%	4.4	2.0	82.1%	83.2%	4.5	2.0	83.8%	85.2%
C	18	15% – 25%	4.4	1.2	60.9%	66.6%	4.1	1.3	64.0%	66.5%

For each triple alignment in the data sets, two analyses were made. 1) All three possible pairwise alignments were performed and the percentage of correct matches was found. 2) The triple alignment was performed and the percentage of pairwise correct matches was found. For each data set, the gap introduction and extension costs were chosen to maximize the percentage of correct matches, which are recorded in Table 2. Triple alignments are better than pairwise alignments for the less similar sequences. For the most similar sequences the performance is very close to identical.

Alignments made by the program Clustal W are a little better than with the algorithm presented here. This is not surprising given the more sophisticated position specific gap penalties of Clustal W and the simple distance matrix used here.

6 Conclusion

It has been shown how an optimal multiple parsimony alignment with affine gap cost over a tree can be found. The algorithm has a memory complexity of $O(7.442^N L^{N-1})$ and time complexity of $O(16.81^N L^N)$. This should be compared to the memory complexity of $O(L^{N-1})$ and time complexity of $O(2^N L^N)$ of the linear gap cost algorithm of [6] combined with the Hirschberg approach [2].

Triple alignment has been shown to outperform pairwise alignment on protein sequences. It is likely that this trend will continue for more sequences, but time and memory become limiting factors in the calculations. However, it may be worthwhile to spend the extra computation time to do triple rather than use pairwise alignments in progressive alignment methods.

Given a multiple alignment, the present method may be used to find its cost by limiting the dynamic programming to the prefixes defined by the alignment. The time and memory complexity for this would be quite small. This may be useful in heuristic alignment methods that seek to improve alignments by local iterative adjustments.

The advances made in this work may form the basis of more probabilistic methods. Statistical alignment methods may be developed that can utilize some of the same approaches for keeping track of gap introductions and extensions. This could also be useful in the context of analyses of multiple alignments using stochastic grammars, e.g., in RNA and protein structure prediction, and gene finding.

7 Acknowledgments

I would like to thank the University of Florida and the Carlsberg Foundation for support. I would also like to thank G. A. Lunter, R. Lyngsø, I. Miklós, and Y. S. Song for helpful comments.

References

1. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48** (1970) 443–453
2. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Comm. Assoc. Comp. Mach. **18** (1975) 341–343
3. Myers, E.W., Miller, W.: Optimal alignments in linear space. Comput. Appl. Biosci. **4** (1988) 11–17
4. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162** (1982) 705–708

5. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis: Probabilistic models of proteins and nucleic acids. Cambridge University Press, Cambridge, UK (1998)
6. Sankoff, D., Cedergren, R.J.: Simultaneous comparison of three or more sequences related by a tree. In Sankoff, D., Kruskal, J.B., eds.: Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison, Reading, MA, Addison-Wesley (1983) 253–264
7. Carrillo, H., Lipmann, D.: The multiple sequence alignment problem in biology. SIAM J. Appl. Math. **48** (1988) 1073–1082
8. Feng, D.F., Doolittle, R.F.: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J. Mol. Evol. **25** (1987) 351–360
9. Hein, J.: A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. Mol. Biol. Evol. **6** (1989) 649–668
10. Thompson, J.D., Higgins, D.G., Gibson, T.J.: 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** (1994) 4673–4680
11. Fitch, W.M.: Toward defining the course of evolution: minimal change for a specific tree topology. Syst. Zool. **20** (1971) 406–416
12. Aho, A.V., Sloane, N.J.A.: Some doubly exponential sequences. Fib. Quart. **11** (1973) 429–437
13. Mizuguchi, K., Deane, C.M., Blundell, T.L., Overington, J.P.: HOMSTRAD: a database of protein structure alignments for homologous families. Protein Sci. **7** (1998) 2469–2471