A New Approach For Tree Alignment Based on Local Re-Optimization

Feng Yue and Jijun Tang
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29063, USA
{yuef, jtang}@engr.sc.edu

Abstract

Multiple sequence alignment is the most fundamental task in bioinformatics and computational biology. In this paper, we present a new algorithm to conduct multiple sequences alignment based on phylogenetic trees. It is widely accepted that a good phylogenetic tree can help produce high quality alignment, but the direct dynamic programming solution grows exponentially [13]. To overcome this problem, we first devise a procedure that can produce optimal alignment for three sequences and infer their common ancestor. We then extend the above procedure to compute the alignment of a given tree with more sequences by iteratively relabeling the internal nodes. We have implemented our algorithm as a C program package, which can handle both DNA and protein data and can take simple cost model as well as complex substitution matrices, such as PAM or BLOSUM series. We test our new method with biological and simulated datasets, and compare its performance with those of other popular multiple sequence alignment tools, including the widely used programs such as ClustalW and T-Coffee.

1 Introduction

Multiple sequence alignment is one of the most fundamental and important tasks in bioinformatics, medical informatics and computational biology. Its applications include homologous genes identification, protein tertiary structure prediction and phylogenetic reconstruction. The most popular and commonly used approach for multiple sequence alignment is progressive alignment. Basically, it works by aligning the two closest sequences first and adding the remaining sequences one by one until all sequences have been aligned. ClustalW [3, 16] is one of the best-known sequence alignment tools based on progressive approach. The main problem with ClustalW is that the initial pairwise alignments are fixed, and early errors cannot be corrected later, even if those alignments conflict with sequences added later[8]. T-Coffee is another popular sequence alignment tool and can be viewed as a variant of the progressive method. It has been reported to get the highest scores on the BAliBASE benchmark database [9]. The significant improvement is achieved by pre-processing a data set of all pair-wise alignments and thus allowing for much better use of information in early stages.

An alternative way to compare multiple sequences is tree alignment, which is motivated by the fact that in most cases the sequences are not independent of each other but rather related by a revolutionary tree [18]. The tree alignment problem was developed principally by Sankoff, who also proposed the first exact (exponential-time) algorithm solution [13] via dynamic programming.

However, finding internal node sequence assignment that maximizes the similarity score is NP-hard [19]. Various approximation algorithms have been designed to heuristically compute tree alignments and phylogenies, such as TAAR [6] and GESTALT [1]. All of these heuristics compute the alignment along a given tree [4] or a simple tree such as the neighbor-joining [12] tree or minimum spanning tree [1].

Tree alignments can be improved with the iterative method proposed by Sankoff et al. [14]. For any binary tree, each internal nodes has three neighbors. Re-optimization for the internal nodes can be achieved by iteratively relabeling each of them using its three neighboring nodes until no further improvement is possible. In experiment, the tree will converge after only 4 or 5 iterations [14]. Therefore, how to compute a high quality alignments for three sequences and infer their internal sequence is essential.

Gotoh [2] presented the first three sequences alignment algorithm under affine gap model. Powell et al. [11] presented an algorithm to infer optimal alignments based on tree score by employing Finite State Machines (FSM), which are explicitly used for the generation of the three sequences from a parent sequence. However, the running time and memory space usage in both algorithms are $O(n^3)$, where $n$ is the length of the sequence. The limitation is ob-
vious – huge demand of memory space makes it impossible to work for sequences with length of more than a couple of hundred characters. For example, when $n = 300$, the total memory usage will be around 3G bytes, and when $n = 1000$, the total memory usage will be over 100G bytes, which are way over current work station’s capacity.

Powell et al. presented another algorithm in [11] with the effort to tackle the memory usage problem. The memory complexity of the new version is $O(d^3)$, where $d$ is the tree score of the alignment. It is highly efficient when $d << n$, which requires the input sequences be very similar. But it’s limited to the simple cost model and cannot use complex substitution matrix such as PAM [10] or BLOSUM [5]. And with the increase of distance among the sequences, the value of distance $d$ can easily grow much bigger than $n$. Thus, the problem of memory usage still exists. Later, Yue et al. [20] proposed an algorithm that solved the high memory usage by applying a divide-and-Conquer strategy [20]. The new algorithm reduces the memory usage to $O(n^2)$ while still producing the optimal alignment.

In this paper, we propose a new multiple sequence alignment method based on given phylogenies. The new method generates the best alignment for a given tree by iteratively performing local re-optimization of the internal nodes. Tree building strategies and memory reduction techniques are also adopted to get better performance. In later sections, we also present the experimental results on biological and simulated datasets, as well as some open questions.

2 The Algorithm

In this section, we will discuss our method that iteratively labels the internal nodes to find the multiple sequence alignment and score of a given tree. If the evolutionary tree is given, we will use it as the base tree to align the sequences along the edges. Otherwise, a neighbor-joining tree based on the pairwise alignment score will be computed and used as the phylogeny.

2.1 Overview

Suppose we are given $k$ sequences and a scoring model, our goal is to compute the alignment of a tree of $2 * k - 2$ nodes, where each leaf is labeled with one of the given sequences and the internal nodes are inferred. The tree score is defined as the sum of all edge lengths defined by the pairwise distance between the two sequences at both ends of the edge. To score a tree, we start by assigning each internal nodes a sequence. It can be as simple as picking a random sequence from the input sequences or it can be computed as the median of its three nearest leaf nodes. Then we iteratively relabel every internal nodes until the tree score converges. Figure 1 shows the outline of our scoring algorithm.

Tree Initialization:
add input sequences to leaf nodes;
lable each internal node as the median of its three nearest leaf nodes;

Tree Optimization/Scoring Tree:
while tree score can be minimized do
for all internal nodes do
compute a new internal sequence with its three neighbors;
if new sequence is better then
relabel the current node with new sequence;
end
end
end

Figure 1. The scoring algorithm.

2.2 The Median Problem (Internal Nodes Optimization)

For $n$ sequences $\{S_1, S_2, \ldots, S_n\}$, the median problem is to find a sequence $S_0$ such that $\sum_{i=1}^{n} d_{0i}$ is minimized, where $d_{0i}$ is the distance between $S_0$ and $S_i$. When $n = 3$ we will call this the median problem of three, or just the median problem. We will describe our median solver briefly in this section and please refer to Yue et al. [20] for more detail.

Assume the input are three sequences, $A$, $B$ and $C$ of length $X$, $Y$ and $Z$ respectively. The output is three aligned sequences, $A'$, $B'$ and $C'$ of the same length $L$, as well as an aligned median sequence $M'$. There is a straight-forward solution for the median problem using dynamic programming technique. Assume each of the three sequences is generated independently from their common parent sequence (the median) by a three-state Finite State Machine (FSM). The possible states for the FSMs are $I$ (insert), $D$ (delete) and $M$ (match/mismatch). The problem of aligning sequences is then transformed into finding how the aligned sequences are generated. At each site in the aligned sequences, there are 27 ($3^3$) possible combinations of states (MMM, IMM, IMD, ...). We can construct a cube of size $|A| \times |B| \times |C|$ for each combination of states (i.e. 27 matrices total), and the result can be computed directly and is optimal.

However, the time and memory complexities of the above algorithm are both $O(n^3)$, where $n$ is the length of the sequence. Thus, it is restricted by the high demand of memory usage and can only work on sequences of less than a few hundred characters. Myers et al. [7] presented a linear space algorithm for pairwise alignment using affine gap costs. Our algorithm uses a similar divide-and-conquer approach to split the problem and reduce the memory requirement to $O(n^2)$. 


If we only want to know the score of three sequence alignment, the solution is simple. Observe in the above three-dimensional matrices, due to its dynamic programming nature, the cells at level \( i \) are only determined by the cells at level \( i - 1 \). Thus, two \((Y \times Z)\) memory spaces are adequate to deliver the score. We use space \( U \) (upper) to store the cells at even levels and space \( D \) (lower) to store the cells at odd levels. The minimum score is stored in \( U[Y, Z] \) (\( X \) is an even number) or \( D[Y, Z] \) (\( X \) is an odd number) when the process stops. However, the alignment cannot be retrieved, because the only tracing information kept in the scoring procedure is only for the latest two levels.

When alignment is desired, the procedure is much more complicated. Let \( i = \frac{X}{2} \), the plane defined by \( i \) will cut the cube into two halves (Figure 2 left). The basic idea of our divide-and-conquer approach is to find the midpoint on the plane \( i \) where the final alignment passes through. Once the midpoint is identified, we will apply the above procedure to the two small cubes, one defined by points \((0, 0, 0)\) and \((i, j, k)\), and the other by \((i, j, k)\) and \((X, Y, Z)\) (Figure 2 right). The process will be executed recursively until boundary conditions are encountered.

![Figure 2. How to split 3-D cube.](image)

We use two steps of forward and backward to find the best midpoints defined by the plane \( i = \frac{X}{2} \). In order to find this midpoint, we will call the score-only function with parameters \((C, B, A, Z, Y, i)\) for the forward step, and with the parameters of \((\text{rev}(C), \text{rev}(B), \text{rev}(A))[i...X], Z, Y, X - i)\) for the backward step. Here \( \text{rev}(\cdot) \) is used to represent the reverse string of a given sequence. By comparing the two planes computed from the forward and the backward functions, we can then find the midpoint that minimize the combining score.

### 2.3 Scoring a Given Tree

For a given tree, each input sequence is assigned to a leaf node with respect to the tree topology. If the internal sequences are given or has been computed, the tree score is simply the sum of all edge lengths defined by the pairwise distance between the two sequences at both ends of the edge. However, labeling the internal nodes of a tree is very challenging and no efficient algorithm is known that would produce an optimal solution for trees with more than three leaves. Our strategy to label the internal nodes and minimize the tree score consists of two main steps: 1) initialize each internal node; 2) iteratively refine the internal sequences until no further change occurs.

Internal sequences can be initialized trivially, by giving each internal node a random DNA or protein sequence. However, other complex procedures yield better results. A better and more complex initialization method can be devised by assigning each internal node the median solution from its three nearest leaves (in term of topology), using the median solver discussed above. We will arbitrarily pick one if there are multiple choices of neighboring leaves for a given internal node.

The initial assignment of internal sequences is likely to be very far away from optimal, thus we must refine these sequences until optimum is achieved. Again, the refinement is based on the median computation. For each internal node \( M \) in the tree, we can re-compute a new sequence \( M' \) with its three neighboring nodes using the median procedure described in section 2.2. Since the new sequence \( M' \) may be different from the previous assignment, we need to check whether to keep the new sequence \( M' \) or not. The criterion is simple: if the sum of the three new edge lengths is better than the old values, we will replace the original node with the new computed sequence. Otherwise, we will keep the original one. If there is any gap in the new node, we will remove all gaps so that the next median problems will still deal with gapless sequences.

Starting from the root node, we can repeatedly relabel each internal node until no further improvement is possible. Our algorithm performs the iteration following a depth-first procedure, although a breadth-first procedure can also be used. If the tree is unrooted, we can randomly pick an internal node as the root. In our practice, we found that this iterative procedure is very robust and generally requires only three to four iterations to converge, and the impact of starting root is very small.

### 2.4 Output Final Tree Alignment

After choosing the best tree with the optimal sequence assignment on all internal nodes, our last step is to produce a multiple alignment along the tree edges, with a requirement that all the sequences should have exactly the same length after being aligned.

The algorithm works as the follows: starting from the root of the tree, we align it with its left child node (if applicable) using pairwise alignment; we then choose its right child node (if applicable) and align it with the previous alignment. Special cares are needed to align a sequence with an alignment where gaps are present. The basic assumption here is similar to the “once a gap, always a gap” strategy used by progressive alignment methods. The pro-
cess will traverse the whole tree in a level by level fashion until all tree nodes have been covered.

3 Experimental Results

The algorithm is implemented as a C program, which can handle both DNA and protein sequences. Users can specify different mutation cost matrices such as PAM or BLOSUM series, as well as costs for gap opening and extension. All experiments are done on a series of Dell workstation with Intel 3.4GHz Xeon CPUs, each equipped with 2G memory.

3.1 BALiBASE Benchmark

The reference alignment we used is version 3.0 of the BALiBASE [17] benchmark database. We concentration our experiments on two families of data: Ref11, Ref12. Ref11 contains datasets each having a small (~ 6) number of equi-distant sequences with low similarity (< 20% identity). Ref12 contains datasets with medium similarity (20-40% identity). There is a total of 38 alignments in RV11 and 44 alignments in RV12, and according to the sequence length they are further divided into three sub-groups: short, length less than 100; medium, length between 100-400 and long, length over 400.

We also compare the new method with T-Coffee and ClustalW, both of which are widely used and have been reported to have high scores on BALiBASE benchmark. The alignments are then assessed using bali_score, a program provided by BALiBASE to compare the inferred and the supposedly correct alignments. bali_score reports two scores: SP and CS. SP (Sum-of-Pair) score represents percent of residue pairs correctly aligned, and CS (column score) represents percent of columns correctly aligned. Higher SP and CS scores suggest better performance. We use default parameters of T-Coffee and ClustalW, and allow these programs to search for the best parameters. For the new method, we use PAM120 substitution matrix and set the gap open penalty as 16 and the gap extension penalty as 4.

Table 1 shows the average SP scores from these three programs. Almost all programs align the short sequences better than longer sequences. Our method achieves the highest score in the short group of both RV11 and RV12. In fact, it even scores 1.00 for 1fjLA family, which means all the characters are correctly aligned. We also find that the twilight zone mentioned in [17] still exists. All tested programs suffer an inconsistency with RV11 dataset, where the identity percentage is less than 25%. For example, the SP scores of T-Coffee on RV11 dataset range from 0.941 to 0.106. The overall average scores of T-Coffee and ClustalW are slightly better than our method for longer sequences, but these results are within a comparable range.

3.2 Simulation

We use the widely used simulator Rose (Random Model of Sequence Evolution) [15] to generate the datasets, and compare our results with those produced by ClustalW and T-Coffee. The reason why we test our algorithm with the Rose simulated datasets is that the true alignments are known and thus it is easy to access the quality of the inferred alignments. Below are some of the important parameters for Rose: we use two groups of the insertion/deletion threshold (0.001, 0.005), and we set the substitution rate as 0.3. We also test three different expected sequence lengths (200, 400 and 800). For each combination of parameters, we generate 100 datasets, and report the average results. The alignments are assessed using bali_score again.

Table 2 shows the average SP scores from the three pro-

<table>
<thead>
<tr>
<th></th>
<th>Ref11 (&lt; 25% id)</th>
<th></th>
<th>Ref12 (20–40% id)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>short medium long</td>
<td>overall</td>
<td>short medium long</td>
<td>overall</td>
</tr>
<tr>
<td>T-Coffee</td>
<td>0.477</td>
<td>0.564</td>
<td>0.546</td>
<td>0.875</td>
</tr>
<tr>
<td>ClustalW</td>
<td>0.734</td>
<td>0.560</td>
<td>0.508</td>
<td>0.893</td>
</tr>
<tr>
<td>New Method</td>
<td>0.814</td>
<td>0.507</td>
<td>0.424</td>
<td>0.905</td>
</tr>
</tbody>
</table>

Table 1. SP scores for T-Coffee, ClustalW and the new method.

<table>
<thead>
<tr>
<th></th>
<th>l = 200</th>
<th></th>
<th>l = 400</th>
<th></th>
<th>l = 800</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>indel = 0.001</td>
<td>indel = 0.005</td>
<td>indel = 0.001</td>
<td>indel = 0.005</td>
<td>indel = 0.001</td>
<td>indel = 0.005</td>
</tr>
<tr>
<td>T-Coffee</td>
<td>0.1926</td>
<td>0.1402</td>
<td>0.2026</td>
<td>0.1261</td>
<td>0.1903</td>
<td>0.1004</td>
</tr>
<tr>
<td>ClustalW</td>
<td>0.1148</td>
<td>0.1065</td>
<td>0.1047</td>
<td>0.1016</td>
<td>0.1032</td>
<td>0.0890</td>
</tr>
<tr>
<td>New Method</td>
<td>0.2517</td>
<td>0.1795</td>
<td>0.2218</td>
<td>0.1329</td>
<td>0.1998</td>
<td>0.1286</td>
</tr>
</tbody>
</table>

Table 2. SP scores for T-Coffee, ClustalW and the new method.
grams. The alignments of our method are obtained based on the true tree, which we know during the simulation. From this table, we find that the new method achieves the highest score in every category, suggesting that its scoring method is able to obtain high quality alignments when the underlying topology is accurate.

4 Discussion

We have presented an algorithm to simultaneously align multiple sequences based on a given phylogeny. Our extensive experiments suggest that this method is very promising, yet there are several intriguing aspects should be addressed. One question is how to better use the internal node information to achieve better overall alignment. This method has shown to produce better three sequences alignment and common ancestor than its counterparts [20], but its overall alignment score for more sequences are lower than the other two. It would be very useful to develop methods to search the best tree with the best alignment, instead of using simple methods such as neighbor-joining.

5 Acknowledgments

The authors were supported by a subcontract under US NSF CIPRES project (EF 03-31654), and by US National Institutes of Health (grant R01 GM078991-01).

References