A Divide-and-Conquer Implementation of Three Sequence Alignment and Ancestor Inference

Feng Yue
Dept. of Computer Science & Engineering
University of South Carolina
Columbia, SC 29208, USA

Jijun Tang
Dept. of Computer Science & Engineering
University of South Carolina
Columbia, SC 29208, USA

Abstract

In this paper, we present an algorithm to simultaneously align three biological sequences with affine gap model and infer their common ancestral sequence. Our algorithm can be further extended to perform tree alignment for more sequences, and eventually unify the two procedures of phylogenetic reconstruction and sequence alignment. The novelty of our algorithm is: it applies the divide-and-conquer strategy so that the memory usage is reduced from $O(n^3)$ to $O(n^2)$, while at the same time, it is based on dynamic programming and optimal alignment is guaranteed. Traditionally, three sequence alignment is limited by the huge demand of memory space and can only handle sequences less than two hundred characters long. With the new improved algorithm, we can produce the optimal alignment of sequences of several thousand characters long.

We implemented our algorithm as a C program package `MSAM`. It has been extensively tested with BAliBASE, a real manually refined multiple sequence alignment database, as well as simulated datasets generated by `Rose` (Random Model of Sequence Evolution). We compared our results with those of other popular multiple sequence alignment tools, including the widely used programs such as `ClustalW` and `T-Coffee`. The experiment shows that `MSAM` produces not only better alignment, but also better ancestral sequence. The software can be downloaded for free at http://www.cse.sc.edu/phylo/MSAM.html

1 Introduction

Multiple sequence alignment is the most fundamental task in computational biology. High quality multiple sequence alignment can be used to verify homologous residues among a set of given sequences, and further reveal information on structural or functional evolution.

`ClustalW` [15] is by far the most widely used program for multiple sequence alignment. It uses a progressive alignment strategy where sequences are added one by one to the multiple alignment according to the order indicated by a pre-computed dendrogram. The problem of `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` [9] is another popular tool which improves the progressive strategy by 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 sometimes we have a phylogeny for the sequences involved [17]. In this case, we can compute the overall similarity based on pairwise alignments along tree edges. However, finding sequence assignment of the internal nodes that maximizes the similarity score is NP-hard [18]. The first exact (exponential-time) algorithm for tree alignment was proposed by Sankoff [13]. Approximation algorithms have also been designed to heuristically compute tree alignments and phylogenies, such as `TAAR` [5] and `GESTALT` [6]. All of these heuristics compute the alignment along a given tree [2] or a simple tree such as the neighbor-joining tree or minimum spanning tree [6].

Sankoff presented the first exact algorithm to perform tree alignment [13] using simple gap costs. Lancia et al. [6] proposed a new algorithm using Steiner sequences of the tree leaves to label the internal nodes of the tree, and gave a high-performance implementation of this algorithm in their program `GESTALT`. However, `GESTALT` still assumes simple gap costs, and thus the alignment and ancestral sequence obtained by `GESTALT` is worse than those produced with affine gap model, as shown in section 5.

In this paper, we will deal with a special case of the tree alignment problem, where there are only three input sequences. These three sequences form the smallest un-rooted binary tree and they are all derived from a common ancestor. Our task is to infer the ancestor sequence at the internal tree node and the multiple alignment of these four sequences (three input and one ancestor), with the goal to minimize the tree alignment cost. Remember that in a binary tree,

each internal node has three neighbors. Therefore, the algorithm proposed in this paper can be used as a foundation to find tree alignment for multiple sequences. [17]

Gotoh [1] presented the first three sequences alignment algorithm under affine gap model. Powell et al. presented an algorithm to give optimal alignments based on tree score using Finite State Machines (FSM) [11], which are explicitly used for the generation of the three sequences from a parent sequence. However, the running time and memory space usage in the above algorithms are both $O(n^3)$, where n is the length of the sequence. Thus, their limitation is obvious – the 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.

In their paper, Powell proposed another algorithm to reduce the running time and the memory usage based on Ukkonen's speed-up. This algorithm requires $O(n + d^3)$ time on average and $O(d^3)$ space, where d is the tree score of the alignment. The requirement for the algorithm is that a match must have a cost of zero and that all other mutation costs be small positive integers. It is highly efficient when $d << N$, which means the input sequences are very similar. The restrictions of Powell's algorithm is obvious. With the increase of distance among the sequences, the number d can easily grow bigger than n. It is not suitable for protein sequences alignment where complex substitution matrix such as PAM [10] or BLOSUM [3] are commonly used. And even for DNA sequences with simple cost model, with the increase of length or mutation rate, the memory and speed complexity algorithm will be even greater than $O(n^3)$. Powell reduced the memory complexity to $O(d^2)$ in [12] with a method similar to Hirshberg's space-saving approach with the speed complexity of $O(d^3 + n \times logd)$. However, the restriction of scoring model remains and the memory usage is still a serious bottleneck as shown in section 5.

We present an algorithm using Hirshberg's space-saving approach and reduce the memory usage from $O(n^3)$ to $O(n^2)$, while the running time is still $O(n^3)$. We have implemented the algorithm as a C program `MSAM`, which can take the complex substitution matrix such as PAM or BLOSUM, and thus can be used to align both protein and DNA sequences. We also show the experimental results for both biological and simulated datasets. The experiments show that when compared with `ClustalW`, `T-Coffee` and `GESTALT`, our algorithm not only gives better alignment , but also infers sequences that are closer to the true ancestor. When compared with Powell's software package, our program runs much faster and requires much less memory and does not have the limitations in their algorithm.

2 The Problem

For n sequences $\{S_1, S_2, \cdots, S_n\}$, 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*. The three sequences form a unique (un-rooted) phylogeny, thus solving the median problem of three can be viewed as the simplest case to simultaneously perform alignment and reconstruct phylogeny.

The goal of our new algorithm is to find the median sequence with affine gap costs, using direct optimization approach derived from dynamic programming. The input is 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'. The median sequence M can be obtained by removing gaps from M'.

3 Three Sequence Alignment in $O(n^3)$ space

A dynamic programming algorithm (DPA) was presented by Gotoh [1] to align three sequences with respect to SP scores using affine gap costs. Later Powell et al. proposed an algorithm that can produce optimal tree alignment also with affine gap costs [11].

Powell et al. assume each sequence is generated independently from a common parent sequence (the median) by a three-state Finite State Machine (FSM). The possible states for the FSMs are `I` (insert), a character is inserted into the output sequence without affecting the input; `D` (delete), a character is deleted from the parent sequence; `M` (match/mismatch), a character is taken from the input sequence and generates a character in the output sequence, which can stay the same or mutate. An importance rule for the FSMs is that if one of the machines is in `I` state, the rest of the machines will idle and stay in the previous state.

Using FSMs, the problem of aligning sequences is 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, each directly describing an alignment output format in accordance with the median character. For a site l in the final alignment, combination `MMM` means $M'[l]$, $A'[l]$, $B'[l]$ and $C'[l]$ are all characters; `MMD` means $M'[l]$, $A'[l]$ and $B'[l]$ are all characters, but $C'[l]$ is a gap '-'.

Note there are a few combinations that lead to the same results in the aligned sequences, but they should be treated differently when used for the computation of the alignments after this site. For example, columns 3 and 4 in Figure 2 (top) and Figure 2 (bottom) look exactly the same, but the gaps in column 4 at C' are different: it is a gap opening if column 3 is in states `IMM` but a gap extension if column 3 is in `IMD`. This difference cannot be verified by the traditional

$$T_{\delta_1\delta_2\delta_3}(i,j,k) = min \begin{cases} T_{MMM}(i',j',k') + Cost(A[i],B[j],C[k],\delta_1\delta_2\delta_3) - g \times Transition(\delta_1\delta_2\delta_3, MMM) \\ T_{MMD}(i',j',k') + Cost(A[i],B[j],C[k],\delta_1\delta_2\delta_3) - g \times Transition(\delta_1\delta_2\delta_3, MMD) \\ T_{MDM}(i',j',k') + Cost(A[i],B[j],C[k],\delta_1\delta_2\delta_3) - g \times Transition(\delta_1\delta_2\delta_3, MDM) \\ \cdots\cdots \\ T_{DDM}(i',j',k') + Cost(A[i],B[j],C[k],\delta_1\delta_2\delta_3) - g \times Transition(\delta_1\delta_2\delta_3, DDM) \end{cases}$$

Figure 1. How to update the cell

character-oriented alignment method, and this is the main contribution of Powell's definition of FSMs.

```
A':  G A A C  M M I M
B':  G A - C  M M M M
C':  G A - -  M M M D
M':  G A - C
columnID:  1 2 3 4   1 2 3 4

A':  G A A C  M M I M
B':  G A - C  M M M M
C':  G - - -  M M D D
M':  G A - C
columnID:  1 2 3 4   1 2 3 4
```

Figure 2. `IMM` **(top) and** `bottom` **(right)**

Our dynamic programming algorithm uses the framework presented by Powell et al. in [11], with minor variations and speedup procedures designed for protein sequences. We define a cube of size $|A| \times |B| \times |C|$, and each cell contains 27 variables to store the state information. These variables are named $T_{\texttt{MMM}}$,$T_{\texttt{MMD}}$, $T_{\texttt{MDM}}$, $\cdots$. For example, $T_{\texttt{MMM}}(i,j,k)$ stores the best score aligning $A[0...i]$, $B[0...j]$ and $C[0...k]$ with alignment ending in format `MMM`, and $T_{\texttt{MMD}}(i,j,k)$ stores the best score at the same position but ending in format `MMD`. For a variable $T_{\delta_1\delta_2\delta_3}$ at cell (i,j,k), its value can be determined as Figure 1 (assuming gap opening cost is g):

The function `Cost` is used to determine the cost for aligning $A[i],B[j],C[k]$ using a given state $\delta_1\delta_2\delta_3$. Assuming $A[i] = T$, $B[j] = G$ and $C[k] = G$, state `MMM` suggests that we need to have a character in the median to align with the three characters. In this example, putting character G in the median will give the minimum score (two matches and one mismatch). On the other hand, suppose we still have $A[i] = T$, $B[j] = G$, $C[k] = G$, but the state is in `MMD`. It then requires us to put a character in the median to align with characters from A, B and a gap from C. Assigning either character G or T in the median will give the same minimal score, which is the sum of one gap opening, one match, and one mismatch (if all mismatches have the same cost). For complex substitution matrices like PAM250 and BLOSUM62, mismatches are usually not equal. Therefore, we must check all characters to determine which character (or gap) should be put in the median: we need to loop through all 5 candidates for DNA sequences, and 21 candidates for protein sequences. Remember this procedure is nested within the three-layered loops ($X \times Y \times Z$), so the time cost is very expensive. To avoid such huge and unnecessary loop, we build a $21 \times 21 \times 21$ look-up table when the program starts, so that when the `Cost` function is called, it can read in the minimal value in one step.

The `Cost` function may overcharge since it ignores whether gaps are already opened or not, thus we must remove such overcharges. In the above example, $A[i] = T$, $B[j] = G$, $C[k] = G$, and state `MMD` indicate that either G or T can be put in the median to align with characters from A, B and a gap from C. The cost is calculated as the sum of one gap opening, one match, and one mismatch. However, if the previous state is also `MMD`, meaning a gap has already been opened in sequence C, the `Cost` function overcharged one gap opening. To solve such a problem, we designed another function `Transition` to discover how many gap openings are overcharged and remove them in the computation.

In the above algorithm, running time and memory space usage are both O(n^3), where n is the length of the sequence. It can produce the optimal alignments and the common parent sequence, but is restricted by the high demand of memory usage and can only work on sequences of less than two hundred characters long and won't be of any use for DNA or Protein comparison, where the length can easily reach a couple of thousands.

4 Three-Sequence Alignment in $O(n^2)$ Space

Note that in the above three dimensional matrices, the cells at level i are only determined by the cells at level $i-1$. Therefore, 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. We start by computing level 0 and 1 and store the results in U and D respectively. Since all the cells at level 2 are determined only by the cells at level 1, we can store the scores at level 2 in U, overwriting the old information. We will continue the process till $i = X$, where X is the length of the first sequence A. 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 in this way, because we have erased the tracing information along the way. We define this score-only algorithm as $scoreThree$.

To reduce memory requirement when alignment is desired, we apply the divide-and-conquer approach to split the three-dimensional cube. For obvious reasons, we will always split along the shortest sequence. We will also assume sequence A is the shortest with length X. Let $i = \frac{X}{2}$, the surface defined by i will cut the cube into two halves (Figure 3 left). The basic idea of our divide-and-conquer approach is to find the point (*midpoint*) on the surface i where the final alignment passes through. Once the midpoint is identified, we will apply the above procedure on 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 3 right). The process will be executed recursively until boundary conditions are encountered.

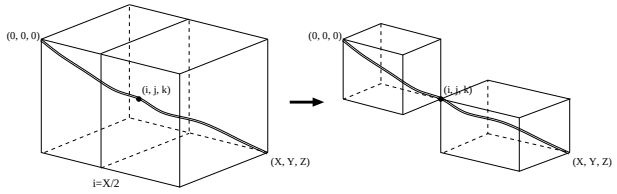


Figure 3. How to split 3-D cube.

4.1 How to find the midpoint

Myers et al. [7] presented an algorithm to identify the midpoints for pairwise alignment using affine gap costs. We extended their approach to handle three sequences. However, the extension is much more complicated than it appears, due to the added one dimension and our additional goal of inferring the ancestral sequence (median).

Our method still uses the two steps, *forward* and *backward*, to find the midpoints. Since the scoring algorithm moves from the top of the cube to its bottom, to obtain the values on the surface defined by $i = \frac{X}{2}$, we must call the function `scoreThree` using parameters (C, B, A, Z, Y, i) for the forward step, and parameters $(rev(C), rev(B), rev(A)[i...X], Z, Y, X - i)$ for the backward step. Here $rev()$ is used to represent the reverse string of a given sequence. For pairwise alignment, the midpoint can be found by just comparing the characters from each direction, but the added median sequence requires us to compare both characters and states information.

Obviously, the change of sequence order in the `scoreThree` function will also change the state information, and thus will produce the wrong median sequence and alignment. We must correct the state informations on all cells residing on surface i before doing the concatenation. Figure 6 shows one example, where Figure 4(a) is the final alignment and state information, and (b) and (c) are the results of the forward and backward procedure. When the cube is split at $i = 3$, the forward call of `scoreThree` produces column 3 of Figure 4(b), and the backward call produces column 4 of Figure 4(c) (information of other columns is added for illustration only). Column 3 in (b) is different from column 3 in (a) due to the change of orders when we call `scoreThree`, and such difference can be simply corrected by reading bottom-up, i.e. the state of column 3 being changed from `MMI` to `IMM`.

The difference of column 4 in the backward step is more difficult to handle. First, we should read the state information from bottom-up the same way we do in the forward step. Second, a new state information must be obtained from the corrected state information of the forward step: when the backward states contain at least one I, the other state(s) must be corrected according to the forward state(s) at the same position. For example, the states of column 4 in (c) will be first changed from `DMI` to `IMD`, then to `IMM`.

A':	C	A	A	A	G	G		F_1:	M	I	I	I	M	M
B':	C	-	-	-	G	G		F_2:	M	M	M	M	M	M
C':	C	-	-	-	-	G		F_3:	M	M	M	M	D	M
M':	C	A	A	A	G	G								
	1	2	3	4	5	6			1	2	3	4	5	6

(a) Aligned (output) Sequences and FSM states

C':	C	-	-	M	M	M		$rev(C)'$:	G	-	-	M	D	D
B':	C	-	-	M	M	M		$rev(B)'$:	G	G	-	M	M	M
A':	C	A	A	M	I	I		$rev(A)[1...i]'$:	G	G	A	M	M	I
	1	2	3	1	2	3					6	5	4	6	5	4

(b) forward (c) backward

Figure 4. Example for state corrections.

After correcting the states information, we can then concatenate the alignment from the forward and backward steps. For each cell (i, j, k) $j \in [0, \cdots, Y]$ and $k \in [0, \cdots, Z]$ on the surface i, its value can be determined from the 27 values at position (i, j, k) produced by the forward step, and the 27 values at position $(X - i, Y - j, Z - k)$ produced by the backward step. Note that the total number of candidates for this cell is 27×27. The cell with the minimum value will be taken as the midpoint.

4.2 How to keep the state information

The midpoint of the cube is associated with a combination of states. After we split the alignment problem into two sub-problems, we need to make sure the previous state information will be passed through the recursive call. We will still use Figure 4 as the example. We found the midpoint is $(3, 1, 1)$ in the previous section, the alignment in the forward procedure end with `IMM`, and the alignment of

PAM250 with gap open = 10, gap extension = 1						
	Ref12 (20-40% id)		Ref11 (<20% id)		Ref2 (orphan)	
	SP	CS	SP	CS	SP	CS
MSAM	**0.850**	**0.781**	**0.579**	**0.462**	**0.744**	**0.629**
T-Coffee	0.843	0.779	0.495	0.366	0.722	0.596
ClustalW	0.842	0.760	0.489	0.363	0.739	0.616

PAM250 with gap open = 16, gap extension = 3						
	Ref12 (20-40% id)		Ref11 (<20% id)		Ref2 (orphan)	
	SP	CS	SP	CS	SP	CS
MSAM	**0.851**	**0.785**	**0.597**	**0.486**	**0.761**	**0.653**
T-Coffee	0.847	0.781	0.518	0.383	0.734	0.614
ClustalW	0.850	0.779	0.427	0.306	0.759	**0.653**

PAM250 with gap open = 20, gap extension = 8						
	Ref12 (20-40% id)		Ref11 (<20% id)		Ref2 (orphan)	
	SP	CS	SP	CS	SP	CS
MSAM	**0.828**	**0.755**	**0.631**	**0.522**	**0.758**	**0.650**
T-Coffee	0.812	0.733	0.524	0.395	0.717	0.589
ClustalW	0.822	0.735	0.359	0.228	0.749	0.636

Table 1. Performance of `MSAM`, `T-Coffee` and `ClustalW` on BAliBASE datasets.

the backward procedure end with `IMM` (after correction). To ensure the correctness of the final alignment, we must start the second (bottom) sub-problem with `IMM` and end the first (upper) sub-problem with `IMM`. This can be achieved by adding two variables $st1$, $st2$ in the recursive function and a $start$ variable in the $scoreThree$ function.

$st1$ and $st2$ will be used to store the state information in the forward and backward procedures. Their initial values are set as $Null$. At the end of the first call in our example, $st1$ is set to `IMM` and $st2$ is set to `IMD`. On the second level of call, the input state parameters are $Null$ and `IMM`, which tells us that the alignment to this sub-problem can start with any state but must end in `IMM`. Suppose that at the end of the call, we find that $st1$ and $st2$ are `MMM` `MMD`, then what we know about the alignment for the next level of recursion is that the first sub-sub-alignment can still start with any format but must end in `MMM`, while the second sub-sub-alignment must start with `MMD` and end in `IMM`. In this way, we not only pass through the state information and retrieve the correct alignment, we can also dump a lot cells that do no need to be computed.

4.3 Boundary Conditions

There are two boundary conditions: X is 1, or one of the input sequence is zero length. $X = 1$ means that there are only two levels in the cube, i.e. $X = 0$ and $X = 1$, thus we can use the existing two tables U and D to produce the alignment. When one of the input sequence is empty, the problem is very similar to pairwise alignment and can be done with $O(n^2)$ space and time complexity.

5 Experimental Results

The algorithm is implemented as a program named `MSAM`[1], which can handle both DNA and protein sequences. Users can specify different mutation cost matrices such as PAM250 or BLOSUM62. Users can also specify costs for gap opening and extension. The whole program is implemented using C, with proper consideration to minimize memory requirement. We test `MSAM` extensively using both biological and simulated data, and compare the results with those obtained through other methods. Since there are many alignment packages available, it is impossible for us to compare `MSAM` with them all. The methods we compared are the leading packages `T-Coffee` and `ClustalW`, a relatively new tree-alignment package `Gestalt`, and our direct competitor Powell's algorithm. All testing are done on a series of Dell workstation with Intel 3.4GHz Xeon CPUs, each equipped with 2G memory.

5.1 Experiments with Biological Benchmarks

We test our algorithm with the most widely used benchmark, `BAliBASE`[16], with concentration on three families of data: Ref11, Ref12 and Ref2. Ref11 contains datasets each having a small (~ 6) number of equi-distant sequences with low similarity (<20% identity); Ref12 contains datasets each having a small (~ 6) number of equi-

[1] the code is available at www.cse.sc.edu/phylo/MSAM.html

	$l = 100$		$l = 200$		$l = 400$		$l = 800$	
	r = 0.1	r = 0.5	r = 0.1	r = 0.5	r = 0.1	r = 0.5	r = 0.1	r = 0.5
MSAM	**20.56**	53.11	**44.16**	**107.46**	**90.58**	**203.50**	**190.7**	**420.21**
Powell's	20.60	**52.52**	45.67	108.16	90.72	205.3	N/A	N/A
GESTALT	27.94	54.51	62.47	109.98	117.89	250.3	223.6	500.10
T-Coffee	21.20	57.63	46.69	115.15	95.62	210.68	195.3	435.22

Table 2. Median accuracy for `MSAM`, `GESTALT`, `T-Coffee` and Powell's Ukkonen algorithm, measured by the distance between the inferred median and the true ancestor (r is the residue substitution rate).

	$l = 100$		$l = 200$		$l = 400$		$l = 800$	
	r = 0.1	r = 0.5	r = 0.1	r = 0.5	r = 0.1	r = 0.5	r = 0.1	r = 0.5
MSAM	0.370	**0.171**	0.303	**0.122**	0.253	**0.110**	**0.201**	**0.10**
Powell's	0.363	0.168	0.301	0.118	0.249	**0.110**	N/A	N/A
GESTALT	0.322	0.168	0.234	0.109	0.206	0.090	0.178	0.081
T-Coffee	**0.389**	0.140	**0.327**	0.103	**0.269**	0.105	**0.201**	0.075

Table 3. Alignment quality for `MSAM`, `GESTALT`, `T-Coffee` and Powell's algorithm, measured by SP score reported by `bali_score`.

distant sequences with medium similarity (20-40% identity); Ref2 contains datasets of families aligned with one or several highly divergent "orphan" sequences.

We compare `MSAM` with `T-Coffee` and `ClustalW` to examine the performance of `MSAM` on real data. Because `MSAM` is designed to handle three sequences so far, for each dataset in Ref11 and Ref12, we generate new datasets (with three sequences) by checking all combinations of three, while for each dataset in Ref2, we generate new datasets by picking one from the orphan sequences and two from the family. 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. Obviously, higher SP and CS scores suggest better performance. We have tested both methods using PAM250 cost matrix and three popular sets of gap costs.

Table 1 reports the average SP and CS scores for the three reference datasets. This table shows that `MSAM` achieves better performance than `T-Coffee` and `ClustalW` in all three reference datasets. This result indicates that tree-based alignment methods can obtain more accurate alignment than pairwise-based methods.

5.2 Simulated Results

We test our algorithm extensively with simulated datasets as well, for two reasons. First, benchmarks are generally designed to be very strict (as in BAliBASE), thus simulations are necessary to test the robustness of an algorithm on all situations. Second, since one goal of our algorithm is to infer the ancestors, we can easily assess the quality of the inferred ancestors using simulations, where the ancestors are known. We use `Rose`(Random Model of Sequence Evolution) [14] to generate the datasets, and compare our results with those produced by `GESTALT` and Powell's algorithm, as well as those from `T-Coffee`. Since some of the programs we compare cannot handle amino acid sequences, we limit our tests on nucleotide sequences only. Below are some of the important parameters for `Rose`: the sequences are on the character set of $\{$A, C, G, T$\}$; the mutation frequencies are set as [0.25, 0.25, 0.25, 0.25]; the insertion/deletion threshold is set as the default value for DNA (0.03); the insertion/deletion length frequencies are set as [.2,.2,.2,.1,.1,.1,.1], which control the probabilities for gaps of lengths from 1 to 7; two groups of testing are done for different evolve rate (0.1 and 0.5): the larger the substitution rate is, the more likely residues at the same site are different; three different expected sequence lengths are tested, ranging from 100 to 800.

For each combination of residue substitution rate and sequence length, we generate 100 datasets, and report the average results. All programs are tested using the same set of parameters, i.e., match costs 0, mis-match costs 1, gap opening penalty is 3, and gap extension costs 1. Although `GESTALT` is not designed to work on affine gap models, it can compute an average cost for a single gap based on the user's estimation of the average gap length. Thus in our experiments, we gave `GESTALT` different estimations of gap lengths and report the best results.

5.2.1 Qualities of Alignments

Since we know the *true* alignments when `Rose` generates the data, we use `bali_score` again to assess the quality of the final alignments. Because CS scores are closely related to SP scores in our simulated results, we only re-

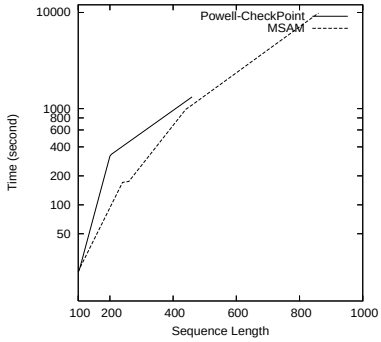
port the SP scores in Table 3. This table shows that MSAM and T-Coffee achieve better results than the other two, and MSAM has the best performance when the sequences are more different. The different alignment accuracy reported by MSAM and Powell's Ukkonen method may result from the different path of finding the best alignment, although their optimal score is the same. Powell's algorithm runs out of memory when the sequences have ~ 800 characters, even using the memory-reduction version, hence no result is reported.

5.2.2 Accuracy of Median Sequences

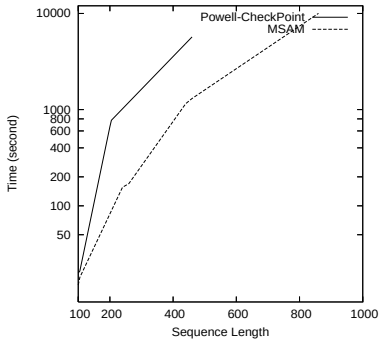
We then check the quality of the *inferred* median sequences produced by the above programs. Since T-Coffee does not produce medians, the median sequences for T-Coffee are created by performing consensus vote at each site of the final alignments. The inferred median sequences are then compared to its corresponding true sequences (generated by Rose). The measurement we use is the alignment score between the median and the true sequences, computed with affine gap costs, which is widely accepted for pairwise alignment. To be consistent, we use the same parameters for pairwise alignment as those used in the multiple alignments. The results are shown in Table 2. Again MSAM gave the best estimation of the ancestral sequences for almost all test cases. This table also shows that consensus vote could not produce good ancestral sequences. Thus, direct optimization methods should be preferred.

5.2.3 Speed

Computation time is another concern in our experiments. Among all four programs, the time used by T-Coffee is trivial compared to the other three, while Powell's algorithm is always the slowest. GESTALT is about $10 \sim 20$ times faster than MSAM. The speed of MSAM is only affected by sequence lengths, and is not affected by other parameters such as gap penalties, mutation costs or similarity among the sequences. Although Powell's algorithm and MSAM produce comparable alignment results, the performance of Powell's algorithm is greatly limited by the input sequences and alignment parameters. There are two versions of Powell's Ukkonen-based method, the standard one ($O(d^3)$ memory) and a better version (checkpoint) that further reduces the space requirement to $O(d^2)$, where d is the tree alignment score (edit distance). Figure 5 shows the computation time of MSAM and $O(d^2)$ version of Powell's Ukkonen method. This figure confirms that Ukkonen-based methods use much more time than the DPA-based MSAM. The reason is that MSAM has time complexity of $O(n^3)$ and is not affected by the edit distances, while Powell's Ukkonen methods have average time complexity of $O(n + d^3)$ for the standard version, and $O(n \times log(d) + d^3)$ for the checkpoint version. Even for simple costs as used in this experiments, d can easily grow larger than n.



(a) r=0.1



(b) r=0.5

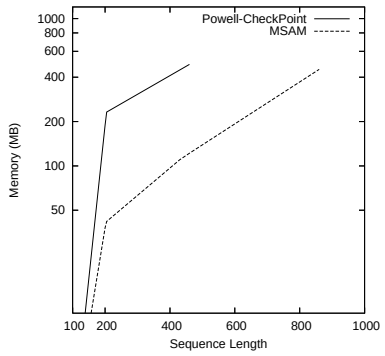
Figure 5. Time usage for MSAM, and Powell's Ukkonen method

5.2.4 Memory

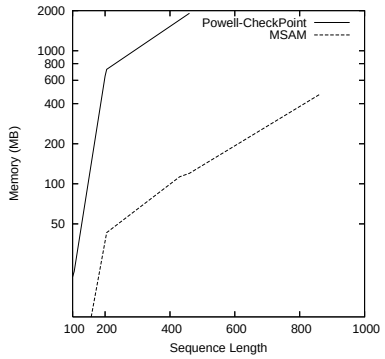
Memory is the bottleneck in three-sequence alignment. Figure 6 gives the memory usage of MSAM and Powell's checkpoint method on datasets with sequence length of $200 \sim 800$. Even for the simplest cost (mismatch and gap extension have cost 1 and gap opening has cost 3), the edit distance will quickly exceed the edge length when the sequences are getting distant (when r is increased), hence more memory is required for Powell's method. The situation will become worse when other costs are used. The memory usage of Powell's reduced-memory method is still large and grows fast. For example, when distance $d = 161$, memory usage is around $20M$ Bytes; when $d = 548$, the memory usage increases to $1720M$ Byte. It cannot be used for more than 600 characters on today's work station, while $MSAM$ has no difficulty to handle several thousand characters.

6 Conclusions

We have presented an algorithm to align three sequences and infer their median, using $O(n^2)$ memory space with time complexity of $O(n^3)$. Our experiment shows that the



(a) r=0.1



(b) r=0.5

Figure 6. Memory usage for `MSAM`, and Powell's Ukkonen method).

algorithm is efficient and accurate, and can produce high quality alignment and median for the given sequences. Our next steps of research include 1) increasing speed of `MSAM` with parallel computing technology and 2) extending `MSAM` to reconstruct phylogenetic trees and align more sequences.

7 Acknowledgments

The authors were supported by a subcontract under US NSF CIPRES project (EF 03-31654). J.T. is also supported by US National Institutes of Health (grant number R01 GM078991-01). We thank Dr. Tandy Warnow and David Zhao for comments and suggestions.

References

[1] Gotoh, O. (1986) Alignment of three biological sequences with an efficient traceback procedure. *J. Theo. Biol.* 121, 327–337.

[2] Hein, J. (1989). 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(6) 649–668.

[3] Henikoff, S., and J.G. Henikoff (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA* 89, 10915–10919.

[4] Hirschberg, D. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM.* 18(6), 341–343.

[5] Jiang, T. and F. Liu (1996). Tree alignment and reconstruction application software.

[6] Lancia G. and R. Ravi (1999). GESTALT: Genomic Steiner Alignments. *10th Annual Symposium on Combinatorial Pattern Matching (CPM 1999)* 101–114.

[7] Myers, E.W. and W. Miller (1988). Optimal Alignments in Linear Space. *Computer Applications in the Biosciences* 4(1), 11-17.

[8] Notredame C. (2002). Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics.* 3(1), 131-44.

[9] Notredame C, D.G. Higgins and J. Heringa (2000). T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.* 302(1), 205-17.

[10] Pearson, W.A. (1990). Rapid and Sensitive Sequence Comparison with FASTP and FASTA. *Methods in Enzymology* 183, 63–98.

[11] Powell, D.R., Lloyd Allison and Trevor I. Dix (2000). Fast, Optimal Alignment of Three Sequences Using Linear Gap Costs *J. theo. Biol.* 207, 325-336.

[12] Powell, D.R. (2002). Algorithms for Sequence Alignment. Ph.D. Dissertation, Monash University.

[13] Sankoff, D. (1975). Minimal mutation trees of sequences. *SIAM J. on App. Math.* 28, 35–42.

[14] Stoye, J., Evers, D. and Meyer, F. (1998) Rose: generating sequence families. *Bioinformatics* 14(2), 157–63.

[15] Thompson, J., D. Higgins and T. Gibson (1994). CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting position specific gap penalties and weight matrix choice. *Nucleic Acids Research* 22, 4673–4690.

[16] Thompson, J. D., F. Plewniak and O. Poch (1999). BAliBASE: A benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics* 15, 87–88.

[17] Vingron, M. and A. von Haeseler (1997). Towards integration of multiple alignment and phylogenetic tree construction. *J. Comp. Biol.* 4(1), 23–34.

[18] Wang, L. and T. Jiang (1994). On the complexity of multiple sequence alignment. *J. Comp. Biol.* 1(4), 337–348.