Comparison of Algorithms for Multiple Sequence Alignment

Elliott Pryor

16 December 2021

1 Introduction

Multiple Sequence Alignment (MSA) seeks to find the optimal global alignment of a set of k input strings. This is a generalization of the global alignment problem to three or more strings. The goal is to align strings $1, \ldots, k$ in order to minimize a scoring function (see Equation 3). Finding the optimal MSA alignment has been shown to be NP-Complete [4]. So different heuristic methods are applied in order to approximate the MSA alignment.

In this work, we examine different MSA algorithms, Center-Star and ClustalW. We implement them in python and compare their running time as well as the quality of alignment produced.

1.1 Global Alignment

The global alignment problem seeks to find the optimal alignment of two strings, S, T that minimizes some similarity function. An alignment is constructed by inserting gaps at different locations into the strings. The length of the two aligned strings must be the same. A common similarity function that is used in this work is given in Equation 1.

$$S = \sum_{i=1}^{n} \delta(S[i], T[i]), \quad \delta(S[i], T[i]) = \begin{cases} 0 & \text{if } S[i] = T[i] \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

We can see that if we have the same strings ($S = T$) the score is 0 from Equation 1. The optimal alignment is the alignment of strings S, T that minimizes the similarity score. For example consider the strings: $ACAATCC$ and $AGCATGC$. They have an optimal alignment with score of 3:

```
A_CAATCC
AGCA_TGC
```

The global alignment problem is known to be solvable in $O(n^2)$ time where n is the length of the longest input string using the Needleman-Wunsch algorithm [1]. The Needleman-Wunsch algorithm is a dynamic programming algorithm. We construct an initial matrix $V = m \times n$, $V[0,0] = 0$, $V[j,0] = j$, $V[0,i] = i$. We assume that $S[0] = T[0] = \_$, then we can compute the optimal score by filling matrix V out using Equatin 2. The optimal score is $V[m,n]$. The optimal string alignment can be determined by backtracing along the optimal b_1, b_2 at each step.

$$V[i,j] = \min_{(b_1,b_2)\in\{(0,1)\}^2\setminus\{(0,0)\})} V[i-b_1, j-b_2] + \delta(S[i \cdot b_1], T[j \cdot b_2]) \tag{2}$$

2 Methods

2.1 Optimal Dynamic Program

This is an extension to the Needleman-Wunsch algorithm that extends to find the optimal solution to the MSA problem. We already know that this problem is NP-Complete, so the algorithm runs in exponential time with respect to k (the number of strings). As a matter of fact the running time is $O(n^k 2^k k^2)$ and it uses $O(n^k)$ space. This is not ideal, but it was expected. We first define the sum of pairs distance that is used to score MSA's. The formula to align a set of characters is given in Equation 3 (we note this function is a parallel of δ in the global alignment problem). If we want to compute the score of an MSA, then $SOP(S_1, \ldots, S_k) = \sum_{i=1}^{n} SP(S_1[i], \ldots, S_k[i])$

$$SP(a_1, \ldots, a_k) = \sum_{1\le i<j\le k} \delta(a_i, a_j) \tag{3}$$

The algorithm is a very simple extension to the Needleman-Wunsch algorithm. Instead of creating V to be a two dimensional matrix, we create V to be a k-dimensional tensor. The tensor is filled out using Equation 4

$$V[i_1, \ldots, i_k] = \min_{(b_1,\ldots,b_k)\in\{(0,1)\}^k\setminus\{(0,\ldots,0)\})} V[i_1 - b_1, \ldots, i_k - b_k] + SP(S_1[i_1 \cdot b_1], \ldots, S_k[i_k \cdot b_k]) \tag{4}$$

We impose the restriction that if $i_j = 0$ then $b_j = 0$ this prevents us from indexing outside of the tensor. Similarly to Needleman-Wunsch, the optimal alignment can be recovered by backtracking along the optimal $\vec{b}$ at each index.

2.2 Center Star

Center Star is a 2-approximation algorithm for MSA. This means that $SOP(\text{center star}) \le 2 * SOP(\text{optimal})$. This performance guarantee is very nice and can be very useful in applications where the score cannot be too far away from optimal. The running time is $O(k^2 n^2)$.

Algorithm 1 Center Star Algorithm

1: **function** CENTER-STAR($S_1, \ldots, S_k$)
2: K = kxk matrix, $K[i, j]$ =edit distance between S_i, S_j
3: Compute distance to other strings $D_i = \sum_{j=1}^{k} K[i, j]$
4: take center string S_c, where $c = argmin_i D_i$
5: Compute alignments with S_c
6: Merge alignments together (need to add spaces into S_c)
7: **end function**

In line 2, we compute the pairwise distances between all strings, the edit distance is computed using the Needleman-Wunsch algorithm. Then in line 4 we find the center string S_c that minimizes the sum of distances to all other strings. We then create the optimal alignment by sequentially merging pairwise alignments. We align S_i, S_c using Needleman-Wunsch. Then if there is a blank introduced into S_c, we insert a blank at this location in all previously aligned sequences S_{i-j} for $j > 0$. We note that aligning the next string in the sequence S_i with S_c uses the current version of S_c (that has blanks from previously aligned strings).

2.3 ClustalW

ClustalW is a popular heuristic for computing MSA. ClutalW was proposed by Thomson et. al. in 1994 [3] It does not have a proven performance guarantee. But is very commonly used to compute MSA in practice. It runs in $O(k^2n^2 + k^3)$ time. I was not able to find any psudocode describing the algorithm, so one of the main contributions of this work is to provide clear psudocode. The main steps of the ClustalW algorithm are:

1. Calculate all possible pairwise alignments, record the score for each pair.
2. Calculate a guide tree based on the pairwise distances via Neighbor Joining.
3. Find the two most closely related sequences
4. Align the sequences by progressive methods (profile-profile alignment)
   (a) Calculate a consensus of this alignment
   (b) Replace the two sequences with the consensus
   (c) Find two next most closely related sequences
5. report the MSA

Step one is easy, it is the same as line 2 in Algorithm 1. There is also no very useful algorithm for neighbor joining algorithm. The most clear implementation is on Wikipedia.

Algorithm 2 Neighbor Joining Algorithm

function NEIGHBOR-JOIN(distance matrix D)
  Initialize T star graph with one leaf for each taxa (string)
  while $D > 2 \times 2$ (D has more than 2 nodes) **do**
    $Q[i,j] = (n-2)D[i,j] - \sum_{k=1}^{n} D[i,k] - \sum_{k=1}^{n} D[j,k]$ ▷ *Compute Q matrix*
    find i, j with $i \neq j$ such that $Q[i,j]$ is minimized
    Connect i, j into new node u in T
    $\delta(i,u) = \frac{1}{2}D[i,j] + \frac{1}{2(n-2)}\left[\sum_{k=1}^{n} D[i,k] - \sum_{k=1}^{n} D[j,k]\right]$ ▷ *Compute distance from i,j to u*
    $\delta(j,u) = D[i,j] - \delta(i,u)$
    $D[u,k] = \frac{1}{2}\left[d(i,k) + d(j,k) - d(i,j)\right]$ ▷ *Compute distances from all other nodes to u*
    Replace i, j with node u and using the distances computed in previous step
  end while
  return T
end function

Now we have a guide tree, so we need to merge the sequences along it. But when we are merging two nodes, if they are interior nodes then there are multiple sequences to align! How do we do this? We need a profile profile alignment score to align alignments A_1, A_2:

$$PSP(A_i[i], A_2[j]) = \sum_{x,y \in \Sigma} g_i(x), g_j(y\delta(x,y))$$

Where Σ is the alphabet, $g_i(x)$ is the number of time x occurs in column i of A_1, and $g_j(y)$ is the number of time y occurs in column j of A_2. We then can merge alignments using Needleman-Wunsch with the scoring function PSP. In other words: $V[i,j] = \max \begin{cases} V[i-1, j-1] + PSP(A_1[i], A_2[j]) \\ V[i-1, j] + PSP(A_1[i], \_) \\ V[i, j-1] + PSP(\_, A_2[j]) \end{cases}$

Thus the progressive alignment algorithm is:

Algorithm 3 Progressive Alignment

function PROGRESSIVE-ALIGN(guide tree T with leaves labeled $S_1, \ldots, S_k$)
  repeat
    Choose two adjacent leaf nodes u, v with parent p
    Compute profile-profile alignment using Needleman-Wunsch with PSP
    Label node p with this alignment A_3
    remove u, v from T
  until only one node in T
  return alignment label on last node remaining
end function

Now we finally have all the pieces in order to give psudocode for the full ClustalW algorithm

Algorithm 4 ClustalW

```
function ClustalW(S_1, ..., S_k)
    D = kxk matrix, D[i, j] =edit distance between S_i, S_j
    T = Neighbor-Join(D)
    label leaf nodes of T with S_1, ..., S_k
    alignment = Progressive-Align(T)
    return alignment
end function
```

3 Experiments

3.1 Experimental Setup

The algorithms described above were implemented in python. Their implementation can be found open sourced on Github/ElliottP-13/ComputationalBiology. We cap the running time for the exact algorithm at 900 seconds. We do not kill the program after it exceeds 900 seconds, but we stop attempting to compute the optimal alignment.

We ran experiments using different number of strings all of length 50. We compare the running time for each algorithm to compute an MSA, as well as the SOP score of the alignment We generate three different topologies for k random strings.

1. Random - Generate k purely random strings
2. Center - Generate one center sequence, then $k-1$ strings with between 5 and 16 mutated characters.
3. Tree - Generate root string. Generate 2 children with between 5 and 16 mutated characters. Repeat until we have k leaf nodes. We note that this simulates a binary phylogeny.

3.2 Results

Results from experiments are shown in Figure 1. Comparisons of MSA scores are in Figure 1a - 1c, and comparisons of running time are in Figure 1d - 1f. The exact algorithm was only able to compute MSA for 3 and 4 strings. Computing MSA for 4 strings took over 20 minutes! On the center string topology, center star was optimal for the 3 strings, and 1 away from optimal for 4. This is expected given the nature of the topology. It is worth noting ClustalW also got optimal for 3 strings on center topology.

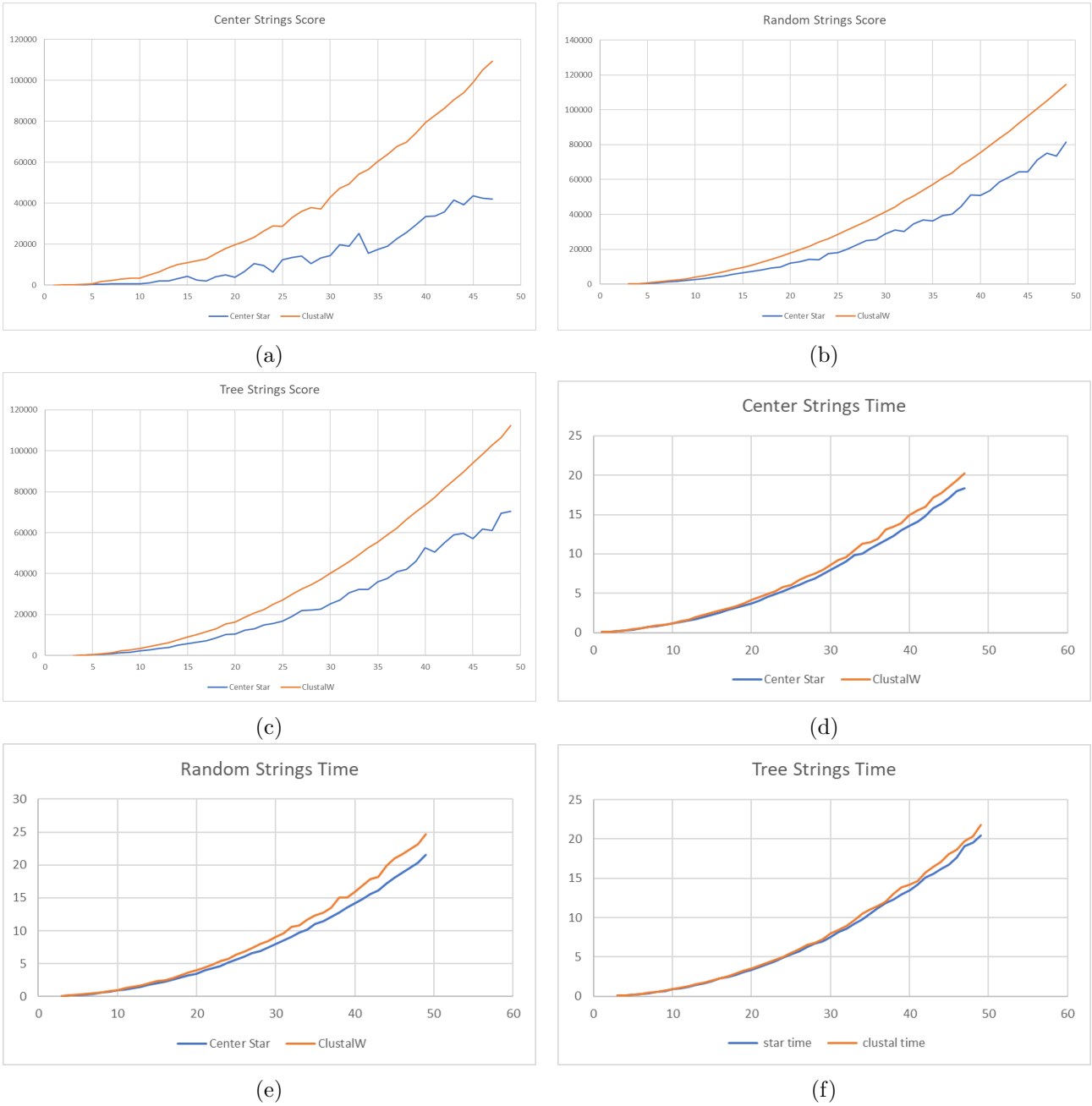

Figure 1: Results showing running time and MSA scores on different string generation strategies

4 Discussion

The results were quite surprising. I expected ClustalW to perform significantly better than CenterStar. However it clearly did not. I have no idea why. It is possible that my data was too synthetic and the algorithm got no benefit from using the guide tree.

The running time makes sense. Both are fairly similar, but ClustalW has an extra k^3 term so it makes sense that it would be slower. Both of the running times are so much shorter than the exact algorithm. It is absolutely absurd! I did not expect the optimal algorithm to be so slow. It takes over 15 minutes to solve only four strings of length 50! This is quite small in the grand scheme of things. I hadn't anticipated the exponential term being so brutal at increasing real running time.

In terms of implementation details, ClustalW was very hard. It requires 2 very non-trivial algorithms in order to work. There is also a non-zero chance that the performance of ClustalW is simply due to implementation error. I do not see where I went wrong, but I had to make several assumptions about what lines of code meant.

References

[1] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[2] Wing-Kin Sung. *Algorithms in bioinformatics: A practical introduction*. CRC Press, 2009.

[3] Julie D Thompson, Desmond G Higgins, and Toby J Gibson. 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(22):4673–4680, 1994.

[4] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.