Final Report
EECS 764: Analysis of Algorithms
Team:
Manavi Sharma, Rameen Ahmed,
Nathan Bui, Chamisa Edmo

Topic: Multiple Sequence Alignment

Introduction

Multiple sequence alignment (MSA) is one of the most important steps in computational biology. Everything we do afterward, including building phylogenetic trees, predicting protein structures with AlphaFold, finding conserved functional motifs, or searching databases with profile HMMs depends on having a solid alignment. The goal with MSA is to take a set of related DNA, RNA, or protein sequences and insert gaps so that residues coming from the same ancestral position end up in the same column.

The problem is that finding optimal alignment is extremely difficult. The exact dynamic-programming solution explodes exponentially with the number of sequences (roughly $O(L^n)$), so even aligning ten medium-length sequences is already extremely complex. That's why every practical MSA tool uses heuristics, making shortcuts when necessary while trying to maintain a reasonable runtime and alignment accuracy.

In this project, we looked at both the theory behind these shortcuts and attempted to make an implementation that would improve alignment accuracy while still using these shortcuts. We built two progressive alignment pipelines that swap out Clustal Omega's very fast (but approximate) k-mer distances for real pairwise alignment distances from VAT (Versatile Alignment Tool). We then tested both versions against Clustal Omega on real Pfam protein families, checking alignment accuracy (SP and TC scores) and how runtime changes as we increase the sequence counts and lengths. We also made an implementation of Gibbs Sampling attempting to calculate an optimal local MSA.

Our three main questions were:

1. How does the fact that optimal MSA is computationally infeasible shape the way real tools are designed?
2. Does using non-approximate VAT alignment distances give a noticeably better guide tree (and final alignment) than Clustal Omega's k-mer trick?
3. Can Gibbs Sampling be used to refine MSA techniques and escape the local optima using probabilistic methods, computing an optimal local MSA?

Why Multiple Sequence Alignment Matters

Essentially, sequence alignment is about reverse-engineering evolution. As species diverge, their sequences pick up substitutions, insertions, and deletions.

A good multiple-sequence alignment does one crucial job: it tells us which amino acids (or nucleotides) descend from the same spot in the ancestral sequence, and which ones were inserted or deleted later. In protein families, the columns that line up correctly almost always correspond to the important parts: catalytic residues, cysteine bridges, binding pockets, alpha helices, beta strands, and other important genomic regions. Moving those key positions by mistake causes the whole downstream analysis (structure prediction, motif discovery, and phylogeny, etc.) to fall apart. That's exactly why getting the alignment right is so crucial.

Biologists are constantly working with datasets that have hundreds or even thousands of sequences, often very distantly related. Because of that, we need MSA tools that are both fast enough to finish in reasonable time and accurate enough to keep those biologically critical columns intact. That combination has become essential for everything from large-scale comparative genomics and protein-family annotation to the newest structure-prediction pipelines.

A solid multiple alignment allows us:

- Identify residues under strong purifying selection
- Discover motifs that define function or taxonomy
- Calculate reliable distances for phylogenetic trees
- Build sensitive profile HMMs for database searching

Because mistakes made here propagate everywhere else, getting the speed–accuracy trade-off right is key.

Problem Definition & Tractability

Problem description (Optimal Alignment with DP)

Multiple sequence alignment (MSA)[3,8] aims to place several DNA, RNA, or protein sequences into a shred coordinate system that allows evolutionarily corresponding positions to line up. A high-quality alignment allows researchers to identify:

- Conserved or functionally important regions
- Shared sequence motifs
- Evolutionary divergence patterns, and
- Structural or functional relationships across sequences.

Unlike pairwise alignment, MSA attempts to align all sequences simultaneously, which makes the problem significantly more computationally demanding as the number of sequences increases. To construct an alignment, gaps are inserted to the residues occupying equivalent evolutionary positions vertically aligned; these "homologous residues," are characters that originate from the same ancestral location.

Formally, given a set of sequences $S = \{S_1, ..., S_m\}$, an alignment constructs transformed sequences, S'_i, by inserting gaps ("-") into each sequence so that all sequences are stretched to the same length, L, where:

$$L \geq \max_i n_i.$$

Columns are intended to represent evolutionarily corresponding positions, which s_i why valid alignments prohibit columns composed entirely of gaps. Each resulting aligned sequence, S'_i, is a length (L), string over the alphabet $\Sigma \cup \{-\}$. Formally, a valid MSA must also satisfy the following:

1. No column may consist entirely of gaps
2. All homologous positions should ideally appear in the same columns

We denote an alignment as $S' = \{S'_1, ..., S'_m\}$. These constraints ensure that the alignment preserves meaningful biological relationships while allowing gaps only where necessary to maintain positional correspondence among sequences.

Mathematical Formulation

To formally describe multiple sequence alignment, we begin with a collection of biological sequences[4]. Let

$$S := \begin{cases} S_1 &= (S_{11}, S_{12}, \ldots, S_{1n_1}) \\ S_2 &= (S_{21}, S_{22}, \cdots, S_{2n_2}) \\ &\vdots \\ S_m &= (S_{m1}, S_{m2}, \ldots, S_{mn_m}) \end{cases}$$

To represent a set of k sequences, each of which may have a different length. These sequences might be DNA, RNA, or proteins, but for the purposes of alignment, each one is treated as an ordered list of characters drawn from some alphabet (A, C, G, T, for DNA; amino acids for proteins; etc.).

The task of alignment is to transform this original set S into a new set of sequences, S'_i

$$S' := \begin{cases} S'_1 = (S'_{11}, S'_{12}, \ldots, S'_{1L}) \\ S'_2 = (S'_{21}, S'_{22}, \ldots, S'_{2L}) \\ \quad\vdots \\ S'_m = (S'_{m1}, S'_{m2}, \ldots, S'_{mL}) \end{cases}$$

Where every S'_i has been extended, by inserting gap characters "-" where appropriate, so that all aligned sequences share a common length *L.* This creates a coordinated representation in which each column is intended to reflect evolutionarily corresponding positions.

$$L \geq \max\{n_i \mid i = 1, \ldots, m\}$$

We can think of this transformation as "stretching" the sequence, so they all fit into the same frame. Some sequences may require gaps to be inserted; others may already align closely. The goal is not simply to equalize their lengths, but to arrange them in a way that preserves meaningful biological relationships.

Since each *m* sequence may receive gaps at any position, the total number of possible alignments grows extremely rapidly. Even if we fix the final aligned length, *L,* each sequence has many ways to place its gaps, and the total number of possible MSAs is the product of all of these choices across *m* sequences. Even with a fixed aligned length, *L*, each sequence has many possible ways to insert gaps, and combining these choices across m sequences leads to a search space that grows something like:

$$\prod_{i=1}^{m} \binom{L}{L - n_i},$$

This formula follows the analysis by Pevzner[11]. Where the exact number of ways to insert gaps into each of the m sequences assumes that:

- The original sequence S_i has a length n_i
- The final aligned length is *L*
- Gaps can appear at the of the *L* positions

This means that the search space grows exponentially with both the number of sequences and their lengths.

So, formally, the alignment must satisfy three conditions:

1. Each aligned sequence, S'_i , must have length.

2. No column is aligned to be composed entirely of gap characters. Such a column carries no biological information and is excluded from valid alignments.

3. The quality of an alignment is typically evaluated through a sum-of-pairs (SP) scoring function, which assigns rewards or penalties based on aligned character pairs (match, mismatch, or gap). The total SP-score reflects how well the alignment captures similarity across the entire set of sequences.

According to the 2006 Optimal SP MSA paper by Carrillo-Lipman[10], for an alignment of *S'* of *m* sequences, each extended to length *L,* the sum-of-pairs (SP-score) score is defined as:

$$SP(S') = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} d(S'_i, S'_j)$$

Where the pairwise score between two aligned sequences is:

$$d(S'_i, S'_j) = \sum_{k=1}^{L} s\big(S'_i[k],\ S'_j[k]\big)$$

Here, $S'_i[k]$ is the character (or gap "-") in sequence *i* at alignment column *k.* s(.,.) is a scoring function assigning numerical value to each aligned pair (match reward, mismatch penalty, gap penalty, etc.). Therefore, the SP score sums up the scores for all pairs of sequences over all alignment columns. Meaning, the higher the SP score, the better, or more evolutionarily consistent, the alignment is.

Formally, the alignment scoring function used throughout this project follows the classical SP-scoring model. Putting these elements together, the goal of optimal MSA is to find the aligned set *S*' that maximizes the chosen scoring function by balancing matches, mismatches, and gaps in a way that best represents the shared evolutionary structure of the sequences. This can be expressed as an optimization problem:

$$S'^{*} = \arg\max_{S'} SP(S'),$$

Tractability Analysis

While the formulation of optimal MSA is mathematically clean, attempting to compute it exactly reveals why the problem is considered one of the most challenging computational biology problems[8]. The classical dynamic programming (DP) strategy that works well for pairwise alignment can, in theory, be extended to multiple sequences. In this generalized version, the DP table becomes a k-dimensional lattice, where each axis corresponds to one of the sequences being aligned. Every point in this lattice represents the best possible score for aligning prefixes of all k sequences simultaneously.

This structure guarantees that, if we fill the entire DP lattice, we will obtain the globally optimal alignment under the chosen scoring model. This, however, comes at a very high computational cost. Each additional sequence we add also adds an additional dimension to the table, turning what was originally a simple 2D matrix for pairwise alignment, into a 3D cube for three sequences, and then a 4D hypercube, and so on. The size and structure grow so rapidly that the amount of time and memory required becomes unmanageable almost immediately.

Practically speaking, the DP table grows exponentially with the number of sequences[9]. The classical DP solution generalizes the pairwise DP matrix to an m-dimensional space, producing a time and space complexity of:

$$O(L^m),$$

Where L is the extended sequence length, and m, is the number of sequences; because each additional sequence adds a dimension to the DP table, exact DP becomes computationally feasible for only a very small m. Further, even with relatively short sequences, if we align more than four or five sequences optimally, this becomes computationally infeasible. Real biological datasets often contain dozens, hundreds, or even thousands of sequences, which is far beyond what an optimal DP could ever handle in practice. As a result, exact optimal alignment is rarely attempted; instead, it serves as a theoretical ideal against which heuristic methods are compared.

This limitation is not just an engineering inconvenience; it is tied to the fundamental complexity of the problem. Under the widely used SP scoring model, optimal MSA has been proven to be NP-Complete[9]. This classification implies that no known polynomial-time algorithm is expected to solve the problem efficiently for all inputs.

Wang and Jiang[9]proved NP-completeness by reducing the Shortest Common Supersequence (SCS) problem to optimal SP-scored MSA. Intuitively, aligning sequences under SP scoring implicitly constructs a supersequence and matches compress it, while gaps extend it. If optimal MSA were solvable in polynomial time, then SCS would also be solvable in polynomial time, implying $P = NP$.

The exponential explosion observed in the DP formulation reflects this underlying theoretical hardness. Therefore, optimal MSA is reserved for very small datasets, while practical alignment tools rely on approximations or heuristics to produce usable results. Because optimal MSA is NP-Complete and exact DP has super-exponential complexity, practical tools rely on heuristic or progressive strategies that approximate the SP-optimal alignment in tractable time. Under the SP scoring model, optimal MSA is NP-complete and therefore believed to be beyond P.

Progressive Alignment

Computing an optimal multiple sequence alignment is difficult given standard computational limits. Most tools today adopt a far more efficient strategy known as progressive alignment. Instead of attempting to align all sequences at once, which requires exploring a massively large search space, progressive methods build the alignment step by step, following an order that reflects how similar the sequences are to one another.

The process begins by measuring how different each pair of sequences is, often using simple pairwise alignments to estimate evolutionary distances. These distances are then used to construct a guide tree, such as one produced through neighbor-joining, that places the most similar sequences closest together. The guide tree acts as a roadmap for the alignment process.

With this roadmap in hand, the algorithm aligns the most similar sequences first, producing small, high-confidence, sub-alignments. These intermediate alignments are then combined in a hierarchical fashion, moving upward through the guide tree. At each step, the algorithm aligns either two sequences, or an existing alignment against another sequence. By the time the root of the tree is reached, all sequences have merged into a single, coherent alignment.

What makes progressive alignment most useful is that it avoids the high-dimensional dynamic programming requirement for optimal MSA. Instead, it reduces the problem to a sequence of manageable pairwise alignments. While this method does not guarantee a globally optimal result, it achieves a balance between accuracy and computational efficiency. Scalability and speed are the two main reasons progressive alignment is foundational to make widely used alignment tools.

Algorithm Analysis

Clustal Omega

Overview:

Clustal Omega[1] is a bioinformatics tool used to perform Multiple Sequence Alignment (MSA) of protein, DNA or RNA sequences. It is a more scalable, and optimized version of traditional progressive alignment approach, especially for large sequences. It improves the traditional approach by speeding up the pairwise distance calculation, clustering process, and final profile alignment.

Clustal Omega uses k-tuple method to perform pairwise distance computations, eradicating the need to perform full pairwise alignment between every pair of sequences. In this method, the algorithm extracts all k-length substrings and represents the sequences as a set of these k-tuples. And then determines similarity between two sequences based on the number of common tuples that they have.

Once the alignment is done, the algorithm uses low dimensional mBed embeddings to perform faster clustering and guide-tree generation. It uses Unweighted Pair Group Method with Arithmetic Mean (UPGMA) to build the guide tree. This approach orders the sequences from most to least similar. Then final alignment is performed using HHAlign which converts them to HHMs for more efficient alignment.

Optimizations

The key optimizations this approach offers are the use of mBed embeddings, k-tuple distances, and HMM-based profile alignment. The use of embeddings reduces the pairwise comparision cost from $O(N^2)$ to $O(N * k)$ where k is the number of representative sequences for the embedding space, which results in the clustering complexity of $O(N \log N)$. Whereas the k-tuples remove the need to perform pairwise alignments, which are expensive. Furthermore, the HMM-based profile alignments are faster and more accurate than traditional profile alignments.

Pros

The Clustal Omega has a lot of benefits attributed to the optimizations discussed above. Firstly, it is very scalable, especially for large datasets. It also has good accuracy for a large number of sequences. ClustalO also has a significantly faster runtime when compared to the traditional progressive alignment algorithm.

Tradeoffs

Despite the pros, there are some shortcomings to this tool. Even though it is faster than progressive alignment, it comes at the cost of lower accuracy. Secondly, when compared to other methods, it can still be slower.

Our *Initial* Two implementations

Progressive Alignment using VAT distances and mBed

Overview

Although our project initially focused on two implementations, we later incorporated a third method (Gibbs Sampling) as an iterative refinement strategy to compare deterministic and stochastic optimization behaviors while attempting to compute a local alignment of shared regions.

Our first implementation leverages the Versatile Alignment Tool (VAT) to perform pairwise alignments and mBed embeddings to optimize guide tree generation. In this approach, we use the mBed to embed sequences in lower dimensional space to a smaller set of sequences. And then clustering is performed in this lower dimensional space to build the guide tree more efficiently. This largely reduces computational cost, especially large sequences. At the same time, it leverages the fast pairwise alignment provided by VAT, which removes the need to perform full-tree computation.

Code Structure

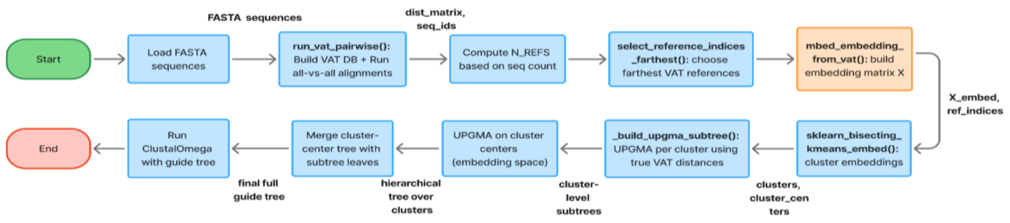

Start
Load FASTA sequences
FASTA sequences
run_vat_pairwise(): Build VAT DB + Run all-vs-all alignments
dist_matrix, seq_ids
Compute N_REFS based on seq count
select_reference_indices_farthest(): choose farthest VAT references
mbed_embedding_from_vat(): build embedding matrix X
X_embed, ref_indices
sklearn_bisecting_kmeans_embed(): cluster embeddings
clusters, cluster_centers
_build_upgma_subtree(): UPGMA per cluster using true VAT distances
cluster-level subtrees
UPGMA on cluster centers (embedding space)
hierarchical tree over clusters
Merge cluster-center tree with subtree leaves
final full guide tree
Run ClustalOmega with guide tree
End

Fig 2.1: Overview of the VAT-based MSA pipeline.

The code starts with running VAT on the FASTA sequences to compute pairwise sequence distances. Then it converts those distances into numerical mBed embeddings. Next, it clusters the sequences, and it builds local UPGMA trees inside each cluster. Then it combines all clusters to produce a full guide tree. Finally, this tree is fed into Clustal O for the final alignments using the HHAlign tool.

Tradeoffs

While this implementation improves the speed of accuracy of progressive alignment as it uses VAT when compared to Clustal O or other heuristics. However, it has greater runtime than pure heuristics methods but is manageable. The runtime is mostly dominated by the VAT distance matrix creation.

Progressive Alignment using VAT distances directly

Overview

The second implementation constructs the guide tree directly from VAT distance matrix unlike the previous implementation which constructed the tree through the mBed embeddings. This approach performs UPGMA hierarchical clustering completely using VAT distance directly. The direct VAT-based hierarchical clustering favors smaller datasets with approximately 50-200 sequences. It does not use embeddings and therefore does not suffer from loss of accuracy due to approximations as in the first implementation. Therefore, this method produces more accurate phylogenetic structures and more biologically meaningful alignments.

Code Structure

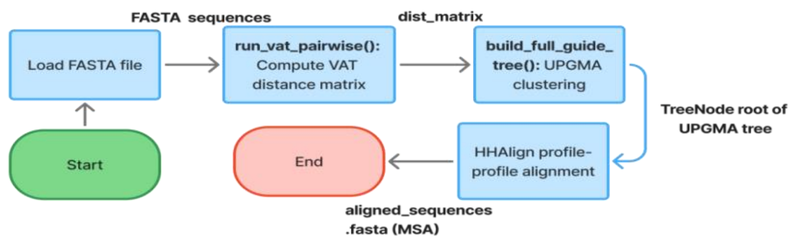

FASTA sequences
dist_matrix
Load FASTA file
run_vat_pairwise(): Compute VAT distance matrix
build_full_guide_tree(): UPGMA clustering
TreeNode root of UPGMA tree
Start
End
HHAlign profile-profile alignment
aligned_sequences .fasta (MSA)

Fig 2.2: Overview of the code workflow used in this project.

This implementation performs all of the major steps as the previous implementation, such as distance matrix and guide tree generation. The main difference in both implementations is that instead of creating mBed embedding through distance matrix, it uses the VAT distances directly to build the guide tree through UPGMA clustering. And uses that tree to perform profile-profile alignment.

Tradeoffs

While there are accuracy advantages of this approach, it is limited due to the computational cost of building the full tree using VAT distance matrix instead of using lower dimensional mBed embeddings. Here as well, the creation of the VAT distance matrix dominates the runtime performance. It is also important to note that for smaller datasets, the extra clustering overhead is not significant.

Gibbs Sampling Method for Local Optimal Alignment

We also chose a method that computes local optimal alignment using Gibbs Sampling[2]. Gibbs Sampling is a method that uses probabilistic methods in an attempt to escape local optima and compute an optimal local alignment. Essentially, this implementation looks for a shared region or motif of length W (specified by the user) between all sequences in the MSA, which must contain no gaps. Our implementation does this by randomly selecting a starting position for each sequence first and computing the residue frequencies of the background. This is the frequency of each base (in this case amino acids) across all sequences. Then it iterates a set number of times (we chose 2000 iterations for our experiment), potentially reaching convergence and escaping local optima to find the local optimal alignment. For each iteration, we remove a sequence from the group of sequences. For all of the remaining sequences, we extract a window of length W based on each sequence's current start position. This is what will constitute our profile, which is calculated based on the residue frequencies of each column. We use this to find a new motif position for the removed sequence where the motif of length W might be found, based off of the probabilities of each window in the removed sequence fitting the motif. This is done by scoring every possible W-length window in the removed sequence, and the new position is selected by sampling probabilistically based off of these scores. This iteration continues until the maximum number of iterations occurs. After that it produces the local MSA containing the motifs at their sequence starting position, with the rest of the positions filled with gaps so that all sequences are the same length as their raw ungapped sequences.

1. Time Complexity of Traditional Progressive Alignment

The standard progressive alignment approach (as used in older tools like ClustalW) has three main stages:

- Full pairwise alignment of all sequence pairs using Needleman–Wunsch → $O(L^2)$ per pair × $O(N^2)$ pairs = $O(N^2 L^2)$
- Guide tree construction using neighbor-joining → $O(N^3)$
- Progressive merging of profiles along the tree → $O(N^3)$

The dominant term is clearly the $O(N^2 L^2)$ distance matrix computation. For this reason, the overall time complexity is considered $O(N^2 L^2)$. This quadratic scaling in both sequence count and length makes the method impractical for datasets with more than a few hundred sequences.

2. Time Complexity of Clustal Omega

Clustal Omega improves dramatically on the classic approach by avoiding full dynamic programming during distance calculation.

- Pairwise distances are estimated using k-mer (k-tuple) counting → $O(L)$ per pair → total $O(N^2 L)$
- Guide tree is built using mBed embedding followed by fast clustering → $O(N \log N)$
- Final alignment uses HHalign on profile HMMs → $O(N L^2)$

The overall complexity therefore becomes approximately $O(N^2 L)$, which is an order of magnitude faster than $O(N^2 L^2)$ in practice. This is why Clustal Omega can align tens or hundreds of thousands of sequences in reasonable time.

3. Time Complexity of VAT Distance-Based Guide Tree using Full Distance Matrix (our "VAT-direct" version)

In this version we skip mBed entirely and build the guide tree straight from the complete VAT distance matrix using classic UPGMA clustering.

- VAT distance matrix creation: $O(N \times T_{VAT}(N, L))$ → One VAT query per sequence against a database containing all N sequences
- Guide tree construction with UPGMA on the full matrix: $O(N^3)$
- Final progressive alignment with HHalign: $O(N L^2)$

Total time complexity: $O(N \times T_{VAT}(N, L) + N^3 + N L^2) \approx O(N \times T_{VAT}(N, L))$

Because $T_{VAT}(N, L)$ is much larger than N^2 or L^2 in practice, the VAT distance calculation completely dominates. This pipeline therefore scales roughly linearly with the number of sequences N (after the fixed cost of building the database), but with a very large constant factor. It works fine up to a few hundred sequences, but becomes noticeably slower than the mBed version once N gets larger.

4. Time Complexity of VAT Distance-Based Guide Tree using mBed (our "VAT + mBed")

This is the version that keeps Clustal Omega's fast clustering trick but feeds it better (VAT) distances.

- VAT distance matrix creation: $O(N \times T_{VAT}(N, L))$ → Still N VAT queries, so same heavy cost as above → However, VAT is internally optimized and in practice a bit faster than running $N^2/2$ full Needleman–Wunsch alignments
- mBed embedding and fast guide-tree clustering: $O(N \log N)$
- Final progressive alignment with HHalign: $O(N L^2)$

Total time complexity: $O(N \times T_{VAT}(N, L) + N \log N + N L^2) \approx O(N \times T_{VAT}(N, L))$

Again, everything is dominated by the VAT step. The huge win compared to the "full matrix" version is that we replace the $O(N^3)$ UPGMA clustering with $O(N \log N)$, so this version stays usable even when we go from a few hundred to a couple thousand sequences.

Time/Space Complexity

Advantages and Tradeoffs

Method	Time Complexity	Space Complexity	Trade Offs
Progressive Alignment Strategy	$O(N^2L^2)$	**O(NL)** for storing sequences + $O(L^2)$ DP matrix	Accurate but slow; dominated by all-vs-all alignment and DP matrices
ClustalO	$O(N^2 L)$	**$O(NL + N^2)$** for distance estimates and embeddings	Much faster; approximates full distances; slight loss of accuracy vs. full DP-based methods but still maintains good accuracy
Gibbs Sampling for MSA	$O(K \times N \times L \times I)$	**O(NL)**	No global optimal guarantee, good for local conserved patterns. Not ideal for full-length progressive alignment
VAT distance-based guide tree using mBed	$O(N * T_{vat}(N^2, L))$	**$O(N^2)$** for the VAT distance matrix + **$O(N \times d)$** embedding	Good balance: higher accuracy than ClustalO; faster than full UPGMA; accuracy limited by embedding approximation
VAT distance-based guide tree using full distance matrix	$O(N * T_{vat}(N^2, L))$	**$O(N^2)$** storage for full VAT distance matrix	Most accurate guide tree; slower tree building than mBed; practical for $N \leq$ ~200 where VAT dominates runtime anyway

Fig 2.3: Summary of time and space complexities for the MSA methods evaluated in this project, along with their practical tradeoffs.

The table above summarizes the time-space complexities and practical tradeoffs of the algorithms evaluated in this project. Classical progressive alignment is computationally expensive because it depends on all-vs-all alignments and DP matrices. ClustalO, or Clustal Omega, significantly improves runtime (by reduction) by using alignment-free distance estimation and embedding-based approximations, sacrificing only a small amount of accuracy. Gibbs Sampling is used here to improve local motif structure but does not guarantee globally optimal alignment. VAT distance-based guide tree using mBed, has higher accuracy than Clustal Omega, but is limited by the embedding approximation. VAT distance-based guide tree using full distance matrix is the most accurate guide tree but is slower than the VAT distance-based guide tree. Overall, these methods demonstrate some common MSA tradeoffs between speed, memory, and alignment accuracy.

Experiments

For our Experiment, we compared our implementations and Clustal Omega to Pfam[5] which is a database containing a large collection of protein families. Each Pfam Family has a reference MSA that we compared each of our generated alignments to. This is an alignment of sequences that are representative of each protein family. We chose 4 Pfam families, with sequence counts ranging from ~55-275 sequences: PF00005, PF00010, PF00019, and PF00029. For each Pfam, we converted its alignment seed into the raw sequences by removing all gaps from the alignment. The raw sequences were used as input for our MSA implementations, and the Pfam seeds or alignments were used for alignment scoring.

Scoring Metrics

For scoring metrics, we chose to evaluate alignment accuracy or quality using the SP (Sum-of-Pairs) and TC (Total Column) Score like the ones used in BAliBASE[6]. The SP score compares residue pairs between a reference alignment and a predicted MSA from one of our implementations to determine how similarly our predicted MSA places residues to the reference. The SP score is determined by the number of residue pairs aligned the same between the reference Pfam alignment and our implementation's MSA.

The TC score is calculated by comparing how many columns match between our reference Pfam alignment and our predicted MSA. This is a much stricter metric because it requires entire columns to match rather than individual residue pairings. Because of this, a higher SP score tends to be more indicative of alignment quality than TC scores.

For determining runtime, we directly measured the time each tool takes to run in seconds.

Results

For our mBed implementation, we found that the tool tended to have a higher SP score when compared to Clustal Omega for overall scoring metrics. The exception to this was for PF00029, where Clustal Omega had a higher SP score than our VAT MSA. In terms of TC score, Clustal Omega scored higher for every Pfam.

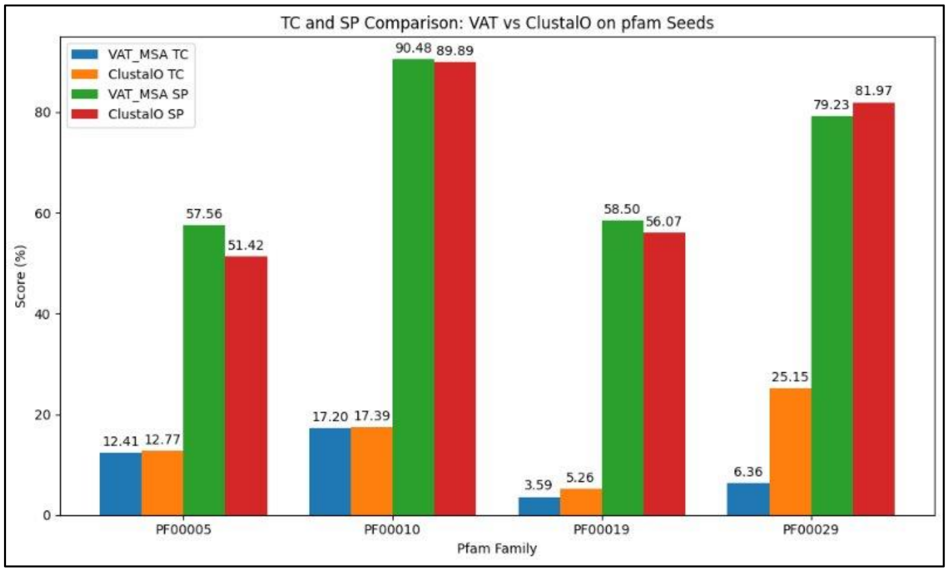


Fig 3.1: Comparison of TC (Total Column) and SP (Sum-of-Pairs) scores between the MSA implementation using mBed to construct the guide tree and Clustal Omega.

These are the scoring results for our implementation that directly created a guide tree using our distance matrix created by VAT. For SP score, this implementation consistently scored higher than

Clustal Omega. However, similar to our previous implementation, Clustal Omega tended to achieve a higher TC score; but this was to a lesser extent compared to the TC scores of our mBed MSA method.

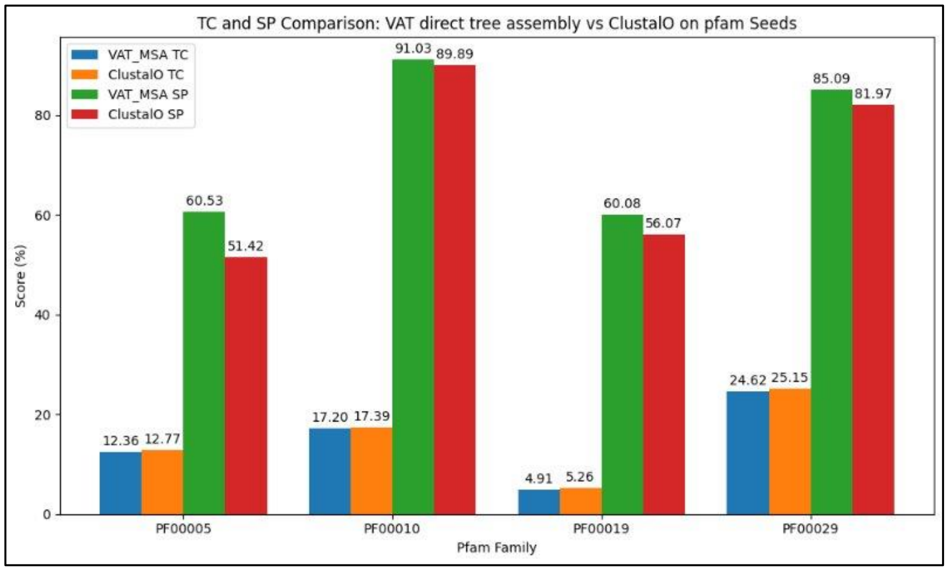


Fig 3.2: Comparison of TC (Total Column) and SP (Sum-of-Pairs) scores between the MSA implementation that directly constructs a guide tree from the VAT distance matrix and Clustal Omega. Comparisons are split between each Pfam family.

For runtimes, both methods seemed to score very poorly, having significantly higher runtimes than Clustal Omega and our theoretical expectations for the time complexity of the VAT-MSA tools. Our MSA tool that used mBed to construct a guide tree had runtimes ranging from 62.94 to 376.87 seconds, whereas Clustal Omega never had runtimes over 8.79 seconds.

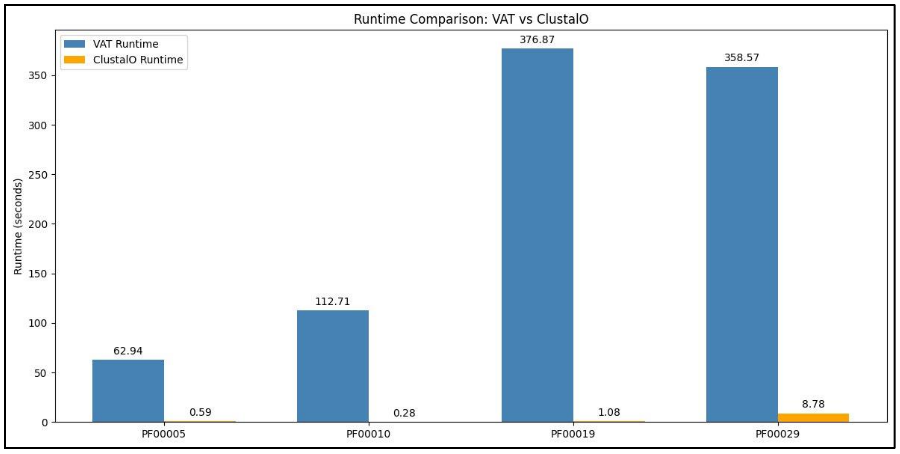
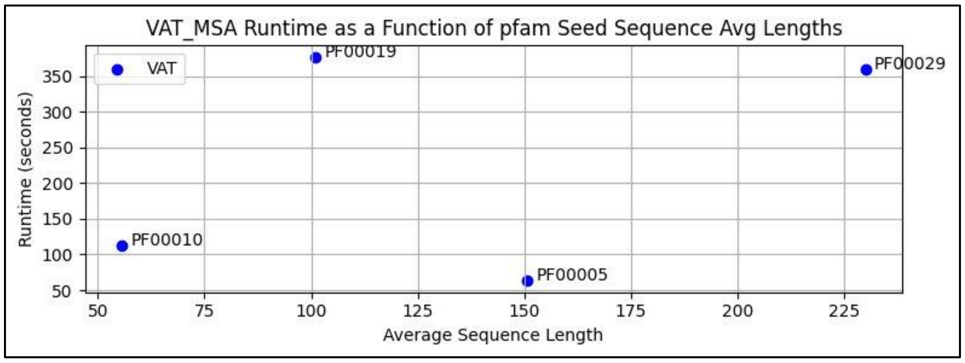


Fig 3.3: Runtime comparisons between our MSA implementation that uses mBed to construct the guide tree and Clustal Omega. Comparisons are split between each Pfam family.

For the influence of sequence counts and length on the runtime of the tool, both seemed to influence the runtime of the tool, with sequence counts contributing more than sequence length.



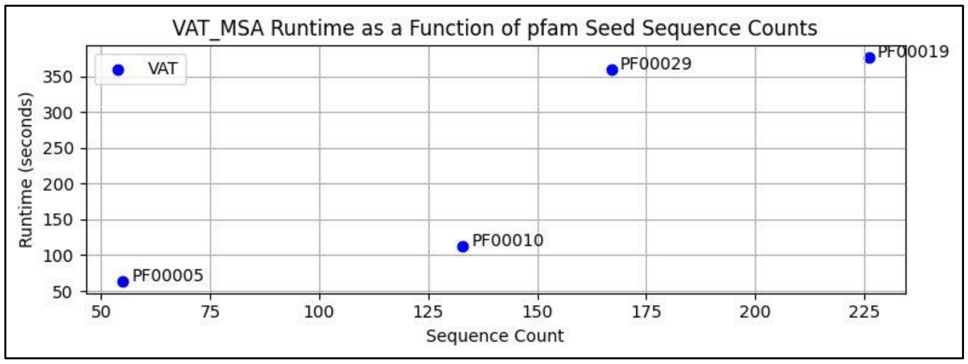


Fig 3.4: Observation of runtimes plotted in a scatter plot as a function of each Pfam seed's average sequence length and as a function of each Pfam seed's sequence count.

For our direct guide tree method, the runtimes were very similar to the other implementation, ranging from 49.71 seconds to 363.14 seconds. Clustal Omega in this case never ran for longer than a 2.56 seconds.

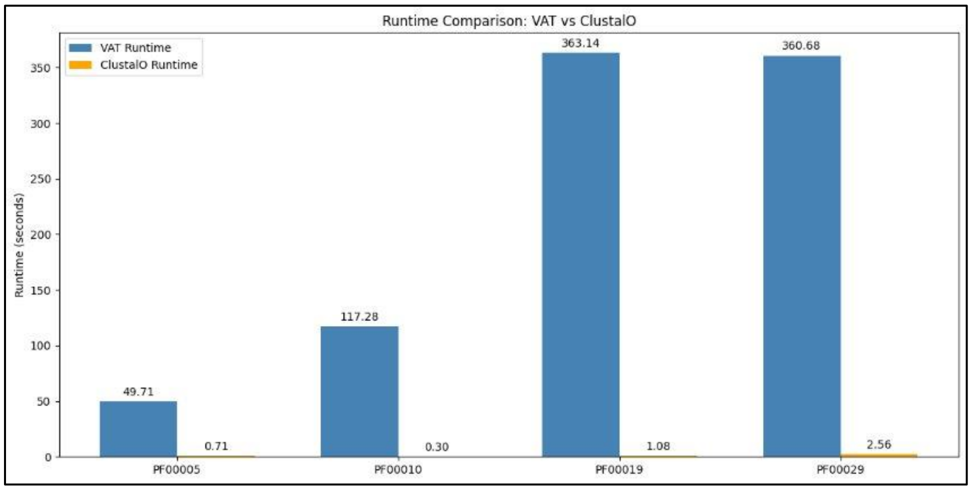


Fig 3.5: Runtime comparisons between our MSA implementation that directly constructs the guide tree using the VAT distance matrix and Clustal Omega. Comparisons are split between each Pfam family.

Gibbs Sampling Results

Looking at our implementation of Gibbs Sampling, we found that the SP Scores were very high for each Pfam. This makes sense because for the motif in each sequence, Pfam should be aligning residues in a very similar way. This suggests that our method does achieve a biologically meaningful local alignment, showing a shared region between all sequences. However, we do see that the TC scores for all Pfam families are 0. Columns do not match as the reference alignment contains other aligned residues besides the shared region or motif that was found from Gibbs Sampling because this is only computing local alignment.

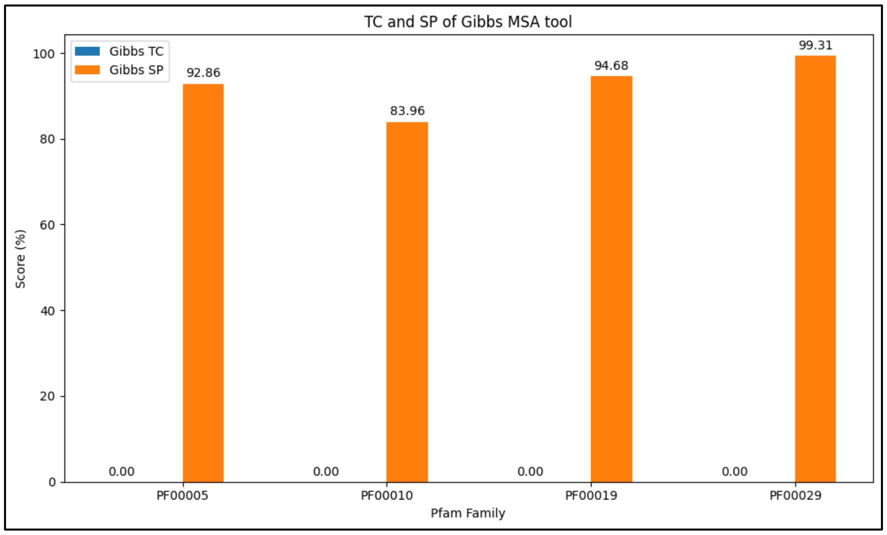


Fig 3.6: TC (Total Column) and SP (Sum-of-Pairs) scores of our Gibbs Sampling implementation for local MSA for each of the 4 Pfam families.

Looking at runtimes for this implementation, we see that they are extremely high, to the point that attempting to compute optimal local alignments for a large group of sequences (something like 10,000) would be infeasible due to the sheer number of iterations that would be needed to reach convergence. This is in line for expectations of optimal multiple alignment, that is considered an NP-Complete problem and infeasible for larger datasets. In our case, the max number of iterations was set to 2000, and the runtime seemed to be affected by both the number of sequences and the sequence lengths. This makes this method infeasible for computing an optimal global alignment as it can only compute a single motif of a specified width between sequences with large runtimes for each shared region computation.

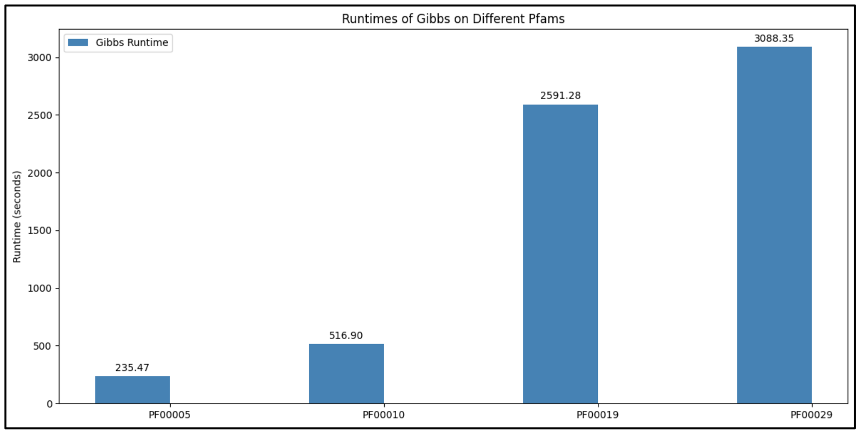


Fig 3.7: Runtimes of our Gibbs Sampling MSA implementation for each of the 4 Pfam Families.

Discussions

Looking at our alignment scoring, both of our implementations perform on par or outperform Clustal Omega in SP score, in most cases, with our direct tree method outperforming in all cases. This indicates that our tools are making more biologically meaningful alignments compared to Clustal Omega. For TC score, both implementations performed worse, with our direct guide tree method only performing slightly worse, and our mBed implementation performing worse in all cases.

For runtime, we found that our runtimes are significantly slower than initially theorized, even compared to much slower methods for computing distance matrices like Needleman-Wunsch[3]. Each VAT alignment to the database seems to take around 1-2 seconds to complete, which is much higher than the couple of milliseconds it should take per VAT alignment. This is most likely due to the large overhead of making VAT alignment calls for each individual sequence. This leads to increased runtime based on the number of sequences in each Pfam. In terms of optimization, it would make sense to change our method for creating the distance matrix using VAT to a more optimized method that avoids making repeated program calls.

Improvements

To handle the high runtimes of the previous implementations of our MSA tools, we chose to change our method for creating a distance matrix using VAT. Rather than aligning *N* sequences to a database containing all sequences, we chose to perform a batch alignment of all sequences to a database containing all sequences. This greatly reduced the overhead of running VAT once for each sequence, combined with VAT's internal optimizations which lead to a significantly faster runtime for both implementations.

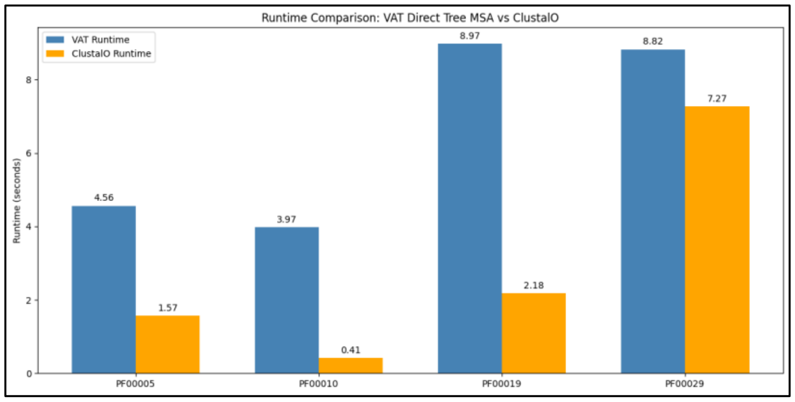


Fig 4.1: Runtimes comparison between Clustal Omega and our MSA implementation that constructs the guide tree using mBed after modifying the VAT matrix creation method to use a single batch alignment. Compares runtimes on each of the 4 Pfam families

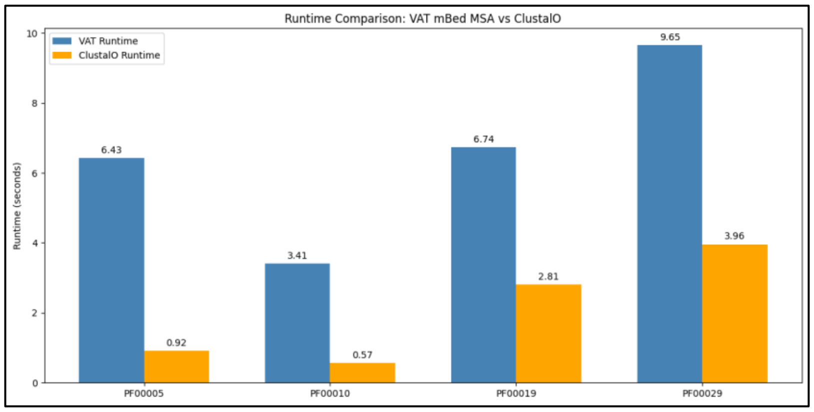


Fig 4.2: Runtimes comparison between Clustal Omega and our MSA implementation that constructs the guide tree directly from the VAT distance matrix after modifying the VAT matrix creation method to use a single batch alignment. Compares runtimes on each of the 4 Pfam families

We also changed our scoring system to utilize a tool called FastSP[7], which can calculate the SP and TC scores while also taking into account the residue indexes. That way, the scoring will not make mistakes based on column misalignment. This is evidenced by the much higher TC and SP scores. Though, part of the changes in scores compared to previous results could also be due to changes in the method of making the distance matrix using VAT. In general, these improved methods reduce the runtime drastically with some decrease in alignment score compared to ClustalO. But the decrease is not significant besides the decrease in SP for PF00005, which might indicate that these implementations might struggle with accurately aligning a smaller number of sequences or a less conserved protein family.

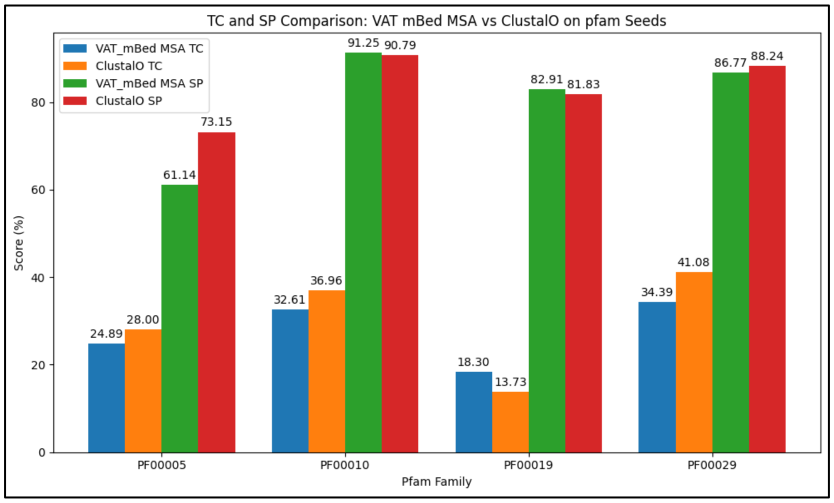


Fig 4.3: Comparison of TC (Total Column) and SP (Sum-of-Pairs) scores between the MSA implementation using mBed to construct the guide tree and Clustal Omega. This plot takes into account new improvements to the VAT matrix creation using batch alignments as well as a new scoring method for getting TC and SP scores using FastSP[7]

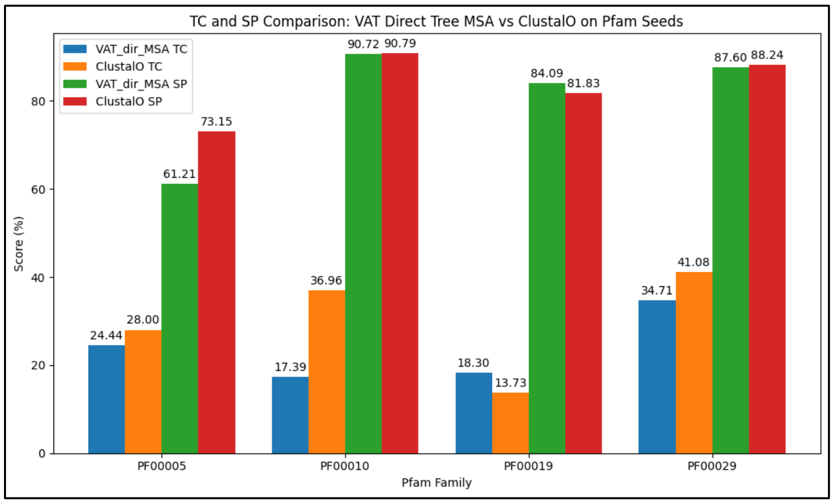


Fig 4.4: Comparison of TC (Total Column) and SP (Sum-of-Pairs) scores between the MSA implementation directly constructing the guide tree using the VAT distance matrix and Clustal Omega. This plot takes into account new improvements to the VAT matrix creation using batch alignments as well as a new scoring method for getting TC and SP scores using FastSP[7]

Conclusion and Future Work

For our implementation of MSA tools, we found that our methods performed on par or only slightly worse than Clustal Omega for scoring metrics. In some cases, we saw our implementations outperforming Clustal Omega, but only in specific Pfam families. This shows promise for these tools being able to make more meaningful biological alignments than Clustal Omega with improvements to some of the techniques used in each implementation down the line and making use of new techniques to improve accuracy.

Our Gibbs Sampling implementation has the potential for computing optimal local alignments with increased iterations until reaching convergence, but that comes with the tradeoff of extremely high or even infeasible runtimes depending on the size of dataset we are trying to compute an optimal local alignment for.

There is still room for improvement in terms of optimizating methods like our mBed implementation and our guide tree construction, especially when compared to Clustal Omega, a well-established tool that has been refined over the period of more than a decade. The improved runtime results using the batch alignments for VAT matrix creation show that there is potential for reducing overhead and optimizing the implementations for improved runtime. Though, due to the techniques used in the implementations, the runtimes will at best be on par with Clustal Omega for smaller sets of sequences and slower for larger scale datasets. However, there is still potential for these implementations of progressive alignment to run faster than previous accurate alignment approaches while being more accurate than Clustal Omega.

For future work, it would be a good to look at larger datasets ranging from around 1000 sequences and test on alignment scoring benchmarks like BAliBASE[6] to determine more conclusively how well each implementation performs on different and more divergent datasets. It would also be useful to conduct a more thorough runtime comparison on larger datasets to see how well each implementation will scale compared to tools like Clustal Omega, as well as how each technique affects runtime on a larger scale.

References

1. F. Sievers *et al.*, “Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega,” *Molecular Systems Biology*, vol. 7, no. 1, article 539, 2011, doi: 10.1038/msb.2011.75.
2. C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, “Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment,” *Science*, vol. 262, no. 5131, pp. 208–214, Oct. 1993.
3. S. Zhang, *Lecture 06: Multiple Sequence Alignment*, EECS 730: Introduction to Bioinformatics, University of Kansas. Lecture slides provided by instructor, 2025.
4. “Multiple sequence alignment,” Wikipedia, The Free Encyclopedia, last edited 14 Nov2025. [Online]. Available: https://en.wikipedia.org/wiki/Multiple_sequence_alignment. [Accessed: 09-Dec-2025].
5. Pfam, “Pfam Protein Families Database,” *Pfam.xfam.org*. https://pfam.xfam.org/ (accessed Dec. 09, 2025).
6. BAliBASE, “BAliBASE: Benchmark Alignment Database,” *lbgi.fr*. https://www.lbgi.fr/balibase/ (accessed Dec. 09, 2025).
7. Mirarab, Siavash, and Tandy Warnow. “FastSP: linear time calculation of alignment accuracy.” Bioinformatics (Oxford, England) vol. 27,23 (2011): 3250-8. doi:10.1093/bioinformatics/btr55=
8. S. K. Gupta, J. D. Kececioglu, and A. A. Schäffer, “Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment,” *Journal of Computational Biology*, vol. 2, no. 3, pp. 459–472, 1995. doi: 10.1089/cmb.1995.2.459
9. W. Wang and T. Jiang, “On the Complexity of Multiple Sequence Alignment,” *Journal of Computational Biology*, vol. 1, no. 4, pp. 337–348, 1994. DOI: 10.1089/cmb.1994.1.337.
10. A. C. Klapper and D. W. Mount, “Optimal sum-of-pairs multiple sequence alignment using incremental Carrillo and Lipman bounds,” *Workshop on Algorithms in Bioinformatics (WABI)*, pp. 1–12, 2006.
11. P. A. Pevzner, Computational Molecular Biology: An Algorithmic Approach. Cambridge, MA, USA: MIT Press, 2000.