HMMerge: an Ensemble Method for Improving Multiple Sequence Alignment

Minhyuk Park and Tandy Warnow

Abstract

Multiple Sequence Alignment is a basic bioinformatics step in many pipelines. Despite advances in the last several decades, the alignment of datasets exhibiting substantial sequence length heterogeneity remains an inadequately solved problem. The previous best approaches (UPP and its variants) use a two-stage technique where a sample of the sequences (with less sequence length heterogeneity) is aligned using an off-the-shelf method such as PASTA or MAGUS, and then the remaining sequences are added into the constructed "backbone" alignment. UPP performs the second step by building an ensemble of Hidden Markov Models (HMMs), and then, for each of the remaining sequences, it selects a best-fitting HMM from the ensemble and uses that selected HMM to add the sequence into the backbone alignment. Here we present a new technique for this second step. Instead of using a single profile HMM from the ensemble to add the query sequence, we merge all the profile HMMs into a complex HMM. We show that using this "merged" HMM provides better accuracy than the current approach in UPP. We also show advantages for this approach over other methods for multiple sequence alignment. HMMerge is available at `https://github.com/MinhyukPark/HMMerge`.

1 Introduction

Multiple sequence alignments (MSAs) are necessary first steps in many common bioinformatics pipelines, such as phylogenetic tree inference and metagenomic taxon identification, and so obtaining high quality MSAs has high relevance in downstream analyses [10, 7]. One of the challenges in multiple sequence alignment is when the input sequence dataset has highly variable sequence lengths, a property that is found in many biological datasets [11]. Sequence length heterogeneity can be caused by evolutionary processes (i.e., large indels) which can make sequences that are both longer or shorter than the "full-length" sequences, or by the use of incompletely assembled genes, or even at its extreme the inclusion of reads, which can make for much shorter sequences (which we refer to as "fragments") [15].

UPP [11] is one of the methods that is designed to address alignment when the input has sequence length heterogeneity. UPP uses a two-stage approach, where it first identifies and extracts a subset of the input sequences (based on their sequence length) that it considers to be "full-length", and then aligns these sequences, forming the "backbone alignment". UPP then builds an ensemble of profile Hidden Markov Models (profile HMMs) to represent the backbone alignment, where each profile HMM is built on a subset of the input sequences in the backbone alignment. Then, each of the remaining sequences (which are called "query sequences) are added into the backbone alignment as follows: each query sequence (a) selects a best-fitting HMM from the ensemble based on the bit scores, (b) the Viterbi algorithm is used to find an optimal path for the query sequence to the selected HMM, and (c) the match states in the optimal path define an alignment of the query sequence to the backbone alignment, which is used to add the query sequence into the backbone alignment.

The original paper introducing UPP studied this approach with PASTA [8] for the backbone alignment, and showed it had excellent accuracy on datasets with sequence length heterogeneity, improving over other methods, and also showed that using backbone alignments of at most 1000 sequences sufficed for good accuracy and excellent scalability (capable of aligning datasets with up to 1,000,000 sequences). A subsequent modification used MAGUS [17] for the backbone alignment and found even better accuracy [14].

The basic two-stage approach in UPP –first compute a (backbone) alignment on a subset of the sequences and then add the remaining sequences into the backbone alignment–suggests extensions to this approach where other techniques are used for the second stage (i.e., adding sequences into the backbone alignment).

In this study, we present HMMerge, a new approach for adding sequences into backbone alignments. HMMerge builds on the techniques in UPP, in that it builds an ensemble of HMMs for the backbone alignment. However, while UPP adds each query sequence into the backbone alignment by selecting a single HMM from the ensemble, HMMerge combines the information from all the HMMs in the ensemble to align each query sequence. Specifically, when adding a single query sequence into the backbone alignment using HMMerge, we compute a statistically rigorous *weight* for each HMM in the ensemble, reflecting the probability that the HMM generated the given query sequence. We present results using an extensive simulation study that demonstrate that HMMerge produces more accurate alignments than UPP.

This paper is organized as follows. Section 2 provides more details about UPP. Section 3 provides an overview of the design for the HMMerge pipeline, and Section 4 presents HMMerge, which is the second step of this pipeline (i.e., HMMerge adds sequences into an existing alignment). Section 5 describes the experimental study design, and the results are then shown in Section 6. Section 7 presents a discussion of HMMerge and we conclude in Section 8.

2 UPP

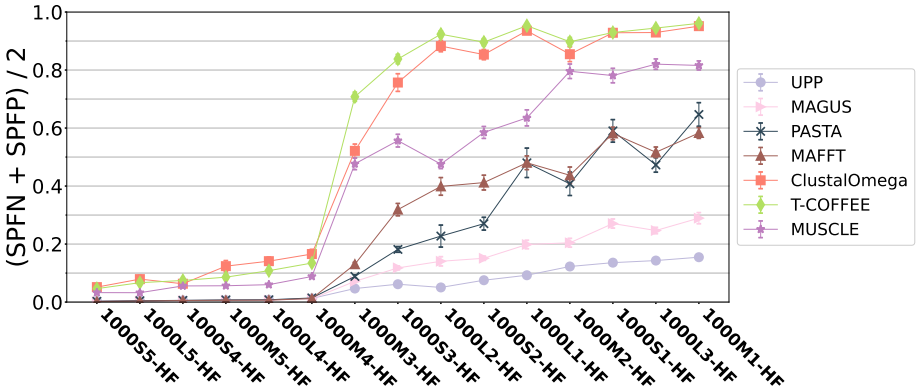

Figure 1: **Alignment error for standard methods on simulated datasets** Alignment error (average of SPFN and SPFP) of the final alignment produced by each method. These are highly fragmented ("HF" for short) datasets created from ROSE simulated datasets introduced in the SATé study [6]. All datasets have 1000 sequences, with rates of evolution that increase from left-to-right. The error bars indicate standard error over 19 replicates.

UPP (Ultra-large alignment using Phylogenetic Profiles) is a method that is designed to align sequence datasets that contain substantial sequence length heterogeneity. In Figure 1, we show the alignment error of standard methods on 15 model conditions that we generated for this study (see Section 5), each with 1000 sequences, a large fraction of fragmentary sequences, and with rates of evolution that varied from low (as you move from left-to-right across the figure). Clearly UPP has lower alignment error than the other methods, but the differences are most noticeable under the high rates of evolution.

The strength of UPP comes from its multi-stage pipeline to align sequences. From a simplified perspective, a full UPP pipeline operates in two stages: the backbone stage and search-and-align stage.

Backbone stage In the backbone stage, the input sequences are split into two sets, with one set containing some of the "full-length" sequences and the remaining sequences denoted "query" sequences. An alignment and tree are built on the full-length sequences, using external methods (e.g., PASTA for the alignment and FastTree 2 [12] for the tree). Then, the backbone tree is decomposed into subsets by edge deletions (removing "centroid edges" that roughly split the leaf-set in half) until each subset is not too large (the default

in UPP is to decompose until each subset has at most 10 leaves). The full set of backbone sequences and any subset produced by the decomposition pipeline becomes one of the subsets. A profile HMM is constructed on each subset using *hmmbuild* from the HMMER suite [2]. The collection of profile HMMs produced in this stage is referred to as an "ensemble of HMMs" for the backbone alignment, or "eHMM".

Search-and-align stage In the search-and-align stage, for every query sequence, an HMM is chosen from the eHMM by choosing the profile HMM with the highest bit-score according to *hmmsearch*. Once the best-fitting HMM is chosen, the query sequence is then aligned to the selected HMM using *hmmalign*. This alignment of the query sequence to the selected HMM defines a way of adding the query sequence to the induced subset alignment, and hence to the full backbone alignment. Thus, the search-and-align stage adds each query sequence into the backbone alignment. The final merged alignment is done using transitivity. Both *hmmsearch* and *hmmalign* are from the HMMER suite.

3 Proposed MSA Pipeline

The pipeline for multiple sequence alignment that we propose is motivated by UPP, in that it uses the same high-level two-stage structure. More specifically, we use the same exact backbone stage, but we modify the second stage (search-and-align). Very specifically, for a given set of sequences exhibiting sequence length heterogeneity, we perform the same steps for the backbone stage (producing therefore the same backbone alignment and tree, and the same ensemble of HMMs). However, when we want to add the remaining query sequences into the backbone alignment, we use a different technique, HMMerge, which is described in the next section.

4 HMMerge algorithm: adding sequences into an existing alignment

Here we describe how HMMerge, the second step of our proposed pipeline, operates. We describe in its most general context, where the input is a multiple sequence alignment (which we will refer to as the backbone alignment) and additional sequences (called query sequences) and the output is an alignment on the full set of sequences (backbone sequences and query sequences) that induces the backbone alignment. Thus, the problem that the HMMerge step addresses is as follows:

- Input: Backbone alignment A and query sequence set Q
- Output: Extended alignment A^* that induces A and includes all the sequences in Q

Note that this description does not require that the input contain a tree T on A nor the ensemble of HMMs representing A.

4.1 High-level description of HMMerge

We now describe at a high level how HMMerge operates, given the input backbone alignment A and query sequence set Q. This algorithm has two steps, but the purpose of the first step is to produce an ensemble of HMMs (eHMM) for the backbone alignment. This requires the calculation of a tree on the alignment, and then the decomposition, etc. Depending on what is already computed, part of Step 1 can be omitted, or perhaps even the entire Step 1.

Step 1: Construction of the eHMM Given the backbone sequence alignment, a maximum likelihood tree T is computed using a preferred method (e.g., FastTree 2 [12], RAxML [19], etc.). Then the tree is broken into subsets, and a profile HMM is built on each subset. The collection of HMMs is referred to as the ensemble of HMMs (eHMM) for the backbone alignment. We use the same technique as in UPP.

Step 2: HMMerge, adding remaining sequences into the backbone alignment using the eHMM This step replaces the "search-and-align" stage for UPP. For every query sequence, all input HMMs are merged to create a single HMM to which the sequence is aligned. The sequence then is aligned to the full-length alignment. We now provide details for how HMMerge operates, beginning with terminology we will use.

4.2 Terminology

substring Given a sequence, or string, S, $S[i]$ denotes the ith character in the sequence while $S[i..j]$ denotes the substring starting at character $S[i]$ and ending at character $S[j]$. The first letter is $S[1]$ throughout this document.

Indexing Given an indexable object such as a matrix, index $\cdot$ refers to the entire range of the current axis. For example, $A[\cdot]$ refers to all entries of A while $B[1, \cdot]$ refers to all column entries of B whose row index is 1.

Precondition The adjacency matrix M, transition probability matrix T, and emission probability matrix P encode the states of an HMM. These states must be ordered such that for any two indices where $i < j$, state i appears before state j in the matrix ordering, and that state i appears before state j in a topologically sorted graph that is represented by M.

- Σ is the set of letters that can be emitted. The set Σ is a finite set.
- n is the number of states in our HMM
- M, the adjacency matrix of dimensions n by n, which represents the HMM. The entry $M[i, j]$ is 1 if there is an edge from state i to state j and $i \neq j$. If there is an edge from state i to state i, then the entry $M[i, i]$ is 2. If there is no edge from state i to state j, then $M[i, j]$ is 0.
  - $M[i, j] = 0$ if there is no edge from state i to state j
  - $M[i, j] = 1$ if there is an edge from state i to state j

- $M[i,i] = 2$ if there is an edge from state i to state i

- T, the transition probability matrix with the same dimensions as M where $T[i,j]$ indicates the transition probability of going from state i to state j. If there is no edge from state i to state j, then the entry $T[i,j]$ is 0. Since the matrix T denotes the transition probabilities, the row sum of any entry is 1 except for the final accepting state, which does not have any outgoing edges and hence has a row sum of 0.

- P, the emission probability matrix of dimensions n by $|\Sigma|$ where $P[i,s]$ is the probability of state i emitting letter s. If state i does not emit any characters, then $P[i,\cdot] = 0$. If state i is an emission state, $P[i,\epsilon] = 0$ where ϵ is the empty string.

4.3 Merged HMM Construction

Here, we describe the procedure for obtaining a hidden Markov model from a set of input profile hidden Markov models and a query sequence.

1. **Topology Construction**: For every column in the backbone alignment, we need to identify which HMMs provide relevant information. Once the relevant HMMs are identified, all of their edges are inserted into the output HMM. To define relevant HMMs in this context, since subset alignments will have fully gapped columns removed, it is possible for certain HMMs to entirely omit columns. Therefore, not all HMMs will provide information about every column in the backbone alignment. In the cases where backbone columns are omitted in the subset alignment, the backbone column immediately before and after the omitted columns are considered to be connected. This is represented with transition edges that skip over omitted columns in the final output HMM.

2. **Numeric Parameters**: Although the topology remains fixed for each query sequence, the numeric parameters vary for each query sequence. The weights of each HMM, and hence the weights of the edges each HMM contributes, are calculated according to the adjusted bit-score of the query sequence to each of the HMMs. This is done through the derivation first presented in [13]. The transition probabilities, insertion probabilities, and emission probabilities from each HMM are averaged with their weights derived from the adjusted bit-scores.

The result is a hidden Markov model that contains all the edges from the input profile hidden Markov models with the transition, insertion, and emission probabilities weighted according to the adjusted bit-scores.

4.4 Matrix Representation

In order to represent an HMM with matrices, we need a mapping from the states to a matrix index. Suppose that B is the number of columns in the backbone alignment. Since every column has a match state, insertion state, and a deletion state, the HMM has $3B$ states that are associated with a backbone alignment column. In addition, there are beginning and end states which adds 2 to the

number, and an insertion state connected to the beginning state which adds 1 to the number of states.

In total, there are $3B + 3$ number of states in our HMM. See Figure 2 for an example HMM. We can map the states in such a way where the ordering is [start, I_0, M_1, I_1, D_1, M_2, I_2, D_2, ..., M_B, I_B, D_B, end]. For the match state corresponding to the ith column (M_i), the index in the ordering is $2 + 3(i - 1)$. For the insertion state corresponding to the ith column (I_i), the index in the ordering is $2 + 3(i - 1) + 1$. For the deletion state corresponding to the ith column (D_i), the index in the ordering is $2 + 3(i - 1) + 2$.

4.5 Alignment of Sequence to Merged HMM

Here, we describe global alignment and glocal alignment. We will also explain how global and glocal alignment are implemented in this paper.

4.5.1 Global alignment (Viterbi Algorithm)

Given two sequences, a global alignment is a type of alignment that aligns the ends of one sequence to the ends of another sequence. In other words, global alignment aligns the sequences "end-to-end". This is implemented via the Viterbi algorithm. We followed the standard Viterbi algorithm with additional cares to ensure that states that do not emit characters are handled properly.

Inputs We define a table VT for ViTerbi. Suppose the input sequence S has length L. The parameters to our function are i and k where VT[i,k] represents the maximum probability of the path that emits the letters $S[0..i]$ and ending at state k. Given a matrix where the rows correspond to the states k and the columns correspond to the indices i, since every VT entry only depends on VT entries where either the row index is less than i, or the column index is less than k, the evaluation order can simply be a column-major ordering. We can start filling in the entries of our table by starting at the first column and going down the rows.

Base cases:

- $VT[0, 0] = 1$ Here, we are assuming that 0 is the index of the start state. This is the top left corner of the matrix if we imagine the VT table to have row indices going from top to bottom and the column indices going from left to right. This base case and the first case in the subproblem together fill out the first row of the VT table.
- $VT[0, j] = 0$ if j is an emission state. This base case and the second case in the subproblem together fill out the first column of the VT table.

Subproblems: State j is an emission state if $\Sigma_{k \in \Sigma} P[j, k] > 0$

$$VT[i, j] = \begin{cases} P[j, S[i]] \cdot \max\{VT[i-1, x] \cdot T[x, j]; x <= j\} & \text{j is an emission state} \\ \max\{VT[i, x] \cdot T[x, j]; x < j\} & \text{j is not an emission state} \end{cases}$$

For all nodes that have an outgoing edge to our current state j, we multiply the probability of emitting the current character $S[i]$ at our current state j with the maximum probability of ending up at our current state.

Log-Likelihoods In order to save computation time and achieve better numerical stability, we implemented a log-likelihood version of the Viterbi algorithm described above. Now, VT[i,k] represents the maximum log probability of the path that emits the letters $S[0..i]$ and ending at state k. We quickly redefine the base cases and the subproblems here.

Log-Likelihoods Base cases:

- $VT[0,0] = 0$ Since $\log_2 1 = 0$.
- $VT[0,j] = -\inf$ if j is an emission state since $\log_2 0 = -\inf$. This base case and the second case in the subproblem together fill out the first column of the VT table.

Log-Likelihoods Subproblems: State j is an emission state if $\Sigma_{k \in \Sigma} P[j,k] > 0$

$$VT[i,j] = \begin{cases} \log_2 P[j, S[i]] + \max\{VT[i-1,x] + \log_2 T[x,j]; x <= j\} & \text{j is an emission state} \\ \max\{VT[i,x] + \log_2 T[x,j]; x < j\} & \text{j is not an emission state} \end{cases}$$

4.5.2 Glocal alignment

A glocal alignment is a hybrid approach in which the entirety of one sequence is aligned to a segment from another sequence. An example where this may be useful is aligning a short sequence to a long sequence. In this case, we would like to align the entirety of the short sequence to a specific region in the longer sequence.

In order to allow for glocal alignment without incurring any penalties, i.e., aligning a query sequence to a specific region of the backbone alignment, we need to allow for our HMM algorithm to be able to start at any match state and end at any match state.

We now define two new probabilities, p_{entry} and p_{exit}, which represent the transition probabilities from the start state to each match state and from each match and delete states to the end state, respectively. See Figure 2 for a visualization of how these entry probabilities override the old edge weights and insert new edges. Let T_0 be the old transition probability matrix, T_1 be the new transition probability matrix, n is the number of states as stated before, and S be the number of match states.

For the entry probabilities, we do the following modification in order to re-normalize old edge weight such that outgoing edge weights sum to 1.

$$T_1[0,i] = \begin{cases} p_{entry} & \text{if } i \text{ is a match state} \\ (1 - S \cdot p_{entry}) \cdot \frac{T_0[0,i]}{\Sigma T_0[0,\cdot]} & \text{where } i \text{ is not a match state} \end{cases}$$

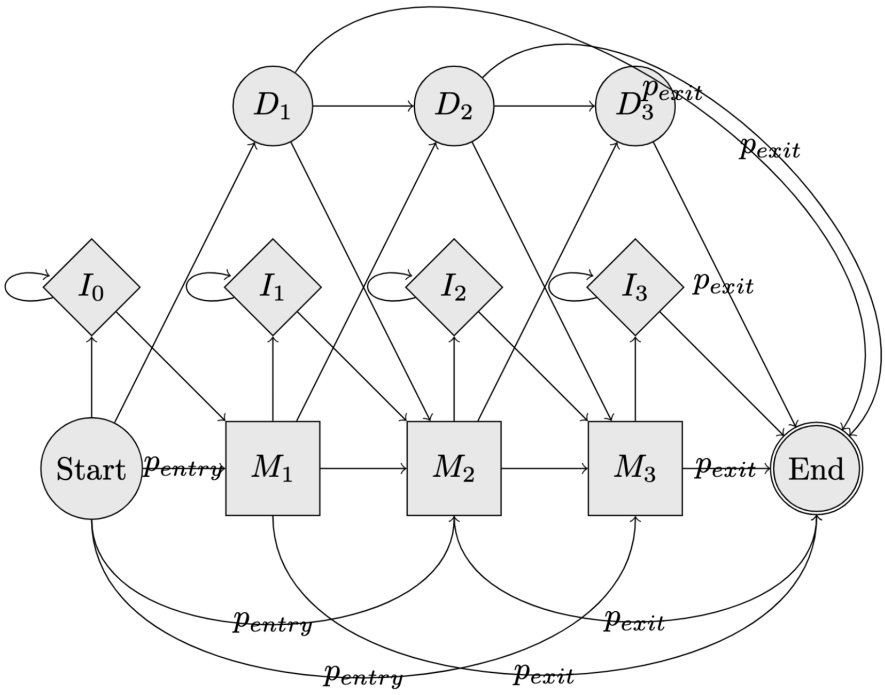

Figure 2: **Equal Entry Exit HMM Example** Here, we show a visualization of an example HMM with equal entry and exit probability edges. P_{entry} and P_{exit} represent the probabilities of a path taking that edge. M_i stands for the ith match state, I_i stands for the ith insertion state, and D_i stands for the ith deletion state. The start and end states do not emit any characters.

For the exit probabilities, we do the following modification in order to renormalize old edge weight such that outgoing egde weights sum to 1.

$$T_1[i,j] = \begin{cases} p_{exit} & \text{if } i \text{ is not an insertion state and } j = n \\ (1 - p_{exit}) \cdot \frac{T_0[i,j]}{\Sigma T_0[i,\cdot]} & \text{if } i \text{ is not an insertion state and } j \neq n \end{cases}$$

4.6 Runtime Analysis

S is the number of sequences in the backbone alignment, B is the length of the backbone alignment, d is the decomposition size, q is the number of query sequences, $O(SB)$ to parse the input sequences and create a mapping of the backbone alignment characters to input decomposed alignment characters, $O(\frac{S}{d})$ calls to *hmmbuild* to build the input HMMs, $O(\frac{S}{d})$ calls to *hmmsearch* to obtain the bit-scores of each query sequence for every HMM, $O(q \cdot \frac{S}{d})$ to parse the *hmmsearch* outputs, $O(B \cdot \frac{S}{d})$ to read the *hmmbuild* outputs, $O(qB^2 \cdot \frac{S}{d})$ to calculate the new probabilities for the merged HMM, $O(B^2)$ to represent the HMM in probability matrices, $O(B^2)$ to add new edges to enable glocal alignment, and $O(q \cdot B^3)$ to run the Viterbi algorithm for each query sequence.

In total, the big O runtime is $O(SB + \frac{qB^2S}{d} + qB^3)$ excluding the $O(\frac{S}{d})$ calls to *hmmbulid* and *hmmsearch*.

5 Experimental Study Design

Since HMMerge can be used within an UPP-like framework in which HMMerge is substituted for the second stage of UPP, it is possible to directly compare the effectiveness of HMMerge and UPP for aligning query sequences given identical inputs. Our first experiment directly explored this on a variety of simulated and biological datasets. Our second experiment focused on evaluating the ability of the full HMMerge pipeline to align full datasets compared to currently existing standard methods.

Datasets We created high fragmentary versions of the ROSE simulated datasets, first studied in [6], using the same technique and parameters as used in [18], meaning that each sequence has been made fragmentary by taking half of the sequences and trimming them to be on average 25% of the median length with a standard deviation of 60 base pairs. These datasets consist of 500 full-length sequences and 500 fragmentary sequences. There are 15 different model conditions we explored, where each model condition varies by the average length of gaps ("S" for Short, "M" for Medium, or "L" for Long) and by their varying evolutionary rates (numbered 1 through 5 the rate of evolution decreases as the number increases). Specifically, we used the high fragmentary versions of 1000S1-1000S5, 1000M1-1000M5, and 1000L1-1000L5.

We also used the CRW biological datasets from the Comparative RNA Website published by the Gutell lab [1]. Specifically, we chose 23S.C, 23S.A, 5S.E, 5S.3, and 5S.T for our study. In order to split the datasets into backbone sequences and query sequences, we split the sequences manually by visual inspection of their sequence length histograms. As a result, 23S.C and 23S.A were both split at 1250, and 5S.E, 5S.3, and 5S.T were all split at 100. That is to

say that on the 23S datasets, any sequence with length greater than 1250 was chosen as the backbone while the remaining sequences were chosen to be query sequences.

For both simulated and biological datasets, we summarize the empirical properties in Table 1.

Name	Sim/Bio	# Sequences	# Full	# Frag	avg. p-dist.	max. p-dist.
1000S1	Sim	1000	500	500	0.694	0.768
1000S2	Sim	1000	500	500	0.693	0.768
1000S3	Sim	1000	500	500	0.686	0.763
1000S4	Sim	1000	500	500	0.501	0.608
1000S5	Sim	1000	500	500	0.498	0.611
1000M1	Sim	1000	500	500	0.695	0.769
1000M2	Sim	1000	500	500	0.684	0.762
1000M3	Sim	1000	500	500	0.660	0.741
1000M4	Sim	1000	500	500	0.495	0.606
1000M5	Sim	1000	500	500	0.499	0.602
1000L1	Sim	1000	500	500	0.695	0.769
1000L2	Sim	1000	500	500	0.696	0.769
1000L3	Sim	1000	500	500	0.687	0.763
1000L4	Sim	1000	500	500	0.500	0.608
1000L5	Sim	1000	500	500	0.496	0.606
1000L5	Sim	1000	500	500	0.496	0.606
23S.A	Bio	214	133	81	0.293	0.667
23S.C	Bio	374	274	100	0.143	0.750
5S.3	Bio	5507	4439	1068	0.417	1.000
5S.E	Bio	2774	1886	888	0.305	1.000
5S.T	Bio	5751	4677	1074	0.425	1.000

Table 1: **Dataset Overview** Here, we show the basic empirical statistics about the datasets used in this study. P-distances refer to the normalized Hamming distance such that two identical sequences have a p-distance of 0 while two sequences with no shared characters have a p-distance of 1. For the simulated datasets, the p-distances shown are prior to fragmentation. For the biological datasets, the backbone sequences were chosen by partitioning the datasets at length 1250 for 23S.C and 23S.A and at length 100 for 5S.3, 5S.E, and 5S.T

MSA Methods We compare our new MSA pipeline, which we refer to as HMMerge for short, to a collection of leading MSA methods. Specifically, we compare to: UPP, MAGUS, PASTA, Clustal Omega [16], MUSCLE [3], MAFFT [5], and T-COFFEE [4]. The only methods that were not run in their default modes were MAFFT, which was run using the L-INS-i algorithm (to maximize accuracy), and T-COFFEE, which was run using the regressive mode (to enable it to run on datasets of these sizes). The exact commands and versions for each of the methods are supplied in the Appendix.

Computational Resources All methods were run on the Illinois Campus Cluster with 16 cores available for parallelism and a default memory limit of

64GB, which none of the methods surpassed. All methods were limited to a wall-time of 4 hours.

Evaluation Criteria We evaluated the methods on both alignment accuracy and runtime. Alignment accuracy was calculated using FastSP [9]. FastSP is a tool that can calculate the SPFN and SPFP scores of an estimated alignment given a reference alignment, where SPFN and SPFP measure the proportion of false negative and false positive pair-wise homologies in the estimated alignment with respect to the reference alignment. Specifically, SPFN, or sum-of-pairs false negatives, is obtained by dividing the number of homologies found in the reference alignment but not in the estimated alignment by the number of homologies in the reference alignment, and SPFP, or sum-of-pairs false positives, is obtained by dividing the number of homologies found in the estimated alignment but not in the reference alignment by the homologies in the reference alignment. The runtime was measured using the elapsed wall-time.

HMMerge and UPP Decomposition Parameters For this study, we chose to do a centroid edge decomposition which is when the backbone tree is split at a centroid edge, an edge that divides the leaves into two as balanced as possible halves, recursively until a stopping criterion is met. In all experiments, the stopping criteria for HMMerge was when the subtree reached size 50 or smaller while for UPP it was when the smallest subtree reached size 2 or smaller. In addition, HMMerge used a disjoint decomposition that only kept the leaf partitions while UPP, as is default, used a hierarchical decomposition that keeps every level of partition in the decomposition. The decision to vary the decomposition parameters was due to the runtime constraints. For UPP, decomposing hierarchically down to size 2 has shown to improve alignment accuracy [13], and it may help the accuracy of HMMerge as well. However, the slow runtime of HMMerge made it not viable to hierarchically decompose down to the granularity of UPP since that would have created significantly more HMMs than keeping just the leaf HMMs. Therefore, we chose to stop the decomposition at size 50 and only keep the leaf HMMs.

5.1 Experiment 1: Comparison of HMMerge and UPP for adding query sequences

In Experiment 1, we explored the effectiveness of HMMerge and UPP for aligning query sequences to an existing backbone alignment. On all datasets tested, HMMerge and UPP used the same backbone alignments computed by MAGUS and backbone trees computed by FastTree [12]. HMMerge, by default, runs in glocal mode, which is the mode we used for all of our experiments. In this experiment, we only looked at the alignment error and runtime when restricted to the query sequences.

5.2 Experiment 2: Comparison of HMMerge pipeline to standard methods on aligning full datasets

In Experiment 2, we explored the effectiveness of the HMMerge pipeline and currently available standard methods such as UPP, MAGUS, PASTA, Clustal

Omega, MUSCLE, MAFFT, and T-Coffee. Similar to Experiment 1, both HMMerge and UPP shared the same MAGUS backbone alignment and FastTree backbone tree. In this experiment, we looked at the alignment error and runtime for aligning the full dataset, not just restricted to the query sequences.

6 Results

6.1 Experiment 1: Comparison of HMMerge and UPP for adding query sequences

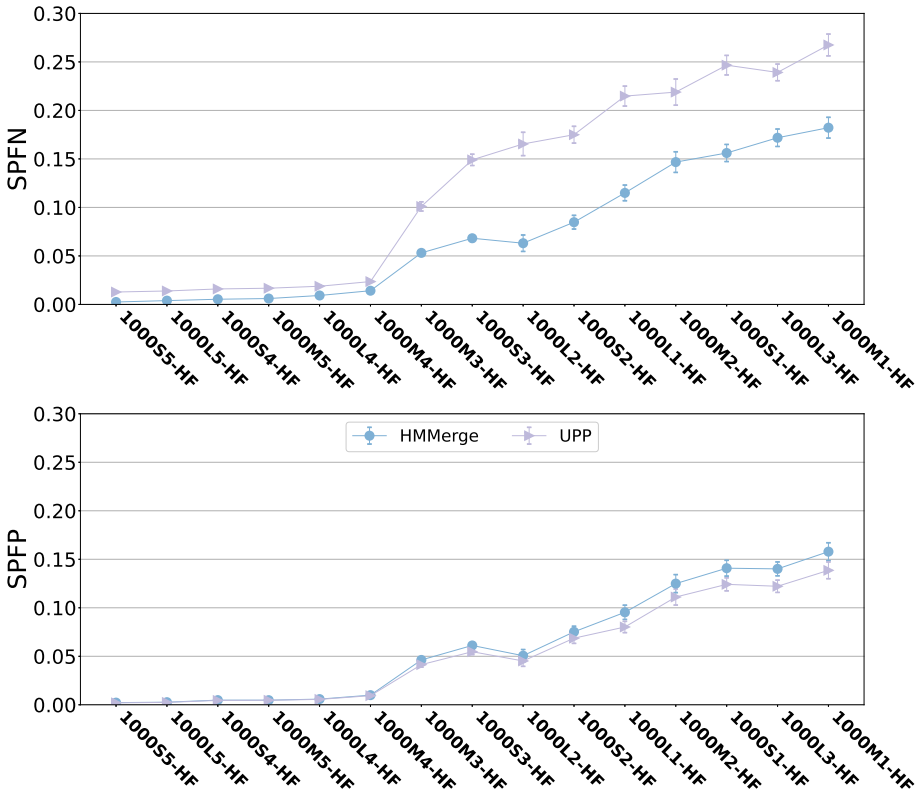

Figure 3: **UPP and HMMerge query sequence alignment error on simulated datasets** We show the SPFN and SPFP error restricted to the query sequences of the alignment produced by HMMerge and UPP. The error bars indicate standard error over 19 replicates.

We observed that HMMerge was able to achieve a much lower SPFN error than UPP at a slight increase in SPFP error on simulated datasets as shown in Figure 3. On the other hand, we observed that on biological datasets as shown in Figure 4, in all but one dataset (23S.C), the SPFN score slightly improved when using HMMerge over UPP, but the SPFP score was worse in all cases.

We also observed that HMMerge was consistently slower than UPP on various simulated and biological datasets as seen in Figure 5 and 6. On the simulated datasets, the greatest runtime difference came from the harder model

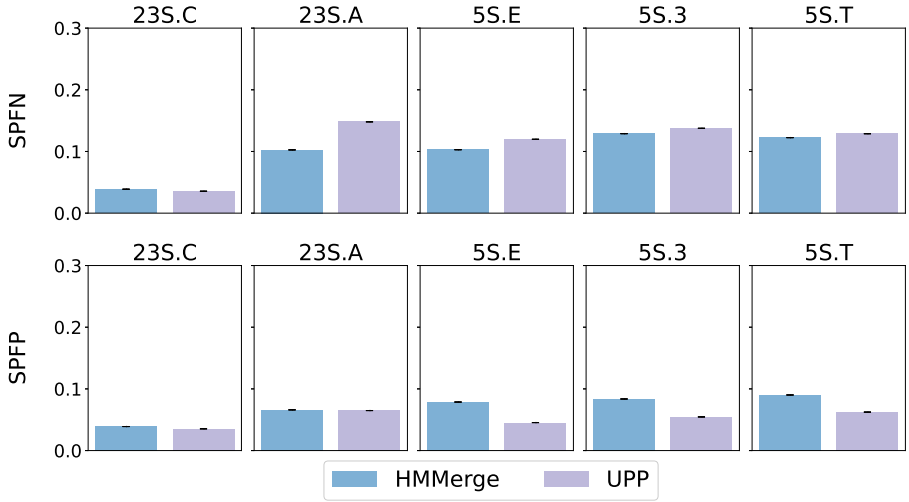

Figure 4: **UPP and HMMerge query sequence alignment error on biological datasets** Here, we show the SPFN and SPFP error on just the query sequences of the alignment produced by UPP and HMMerge.

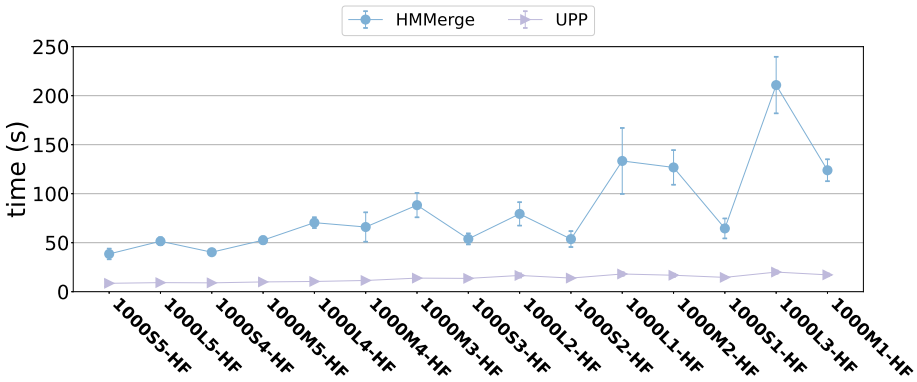

Figure 5: **Total Runtime for HMMerge and UPP pipelines on simulated datasets** Here, we show the runtime of obtaining the final alignment by each method. For HMMerge and UPP, the time includes the time required for building the backbone alignment with MAGUS and tree with FastTree 2. The error bars indicate standard deviation over 19 replicates.

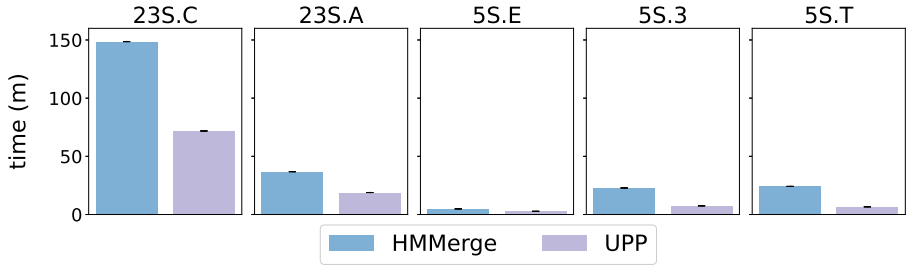

Figure 6: **Total runtime for HMMerge and UPP pipelines on biological datasets** Here, we show the runtime of obtaining the final alignment by HMMerge and UPP; this includes the time required for building the backbone alignment with MAGUS and tree with FastTree.

conditions. On the biological datasets, the greatest runtime difference came from 23S.C (left most dataset in Figure 6)

6.2 Experiment 2: Comparison of HMMerge pipeline to standard methods on aligning full datasets

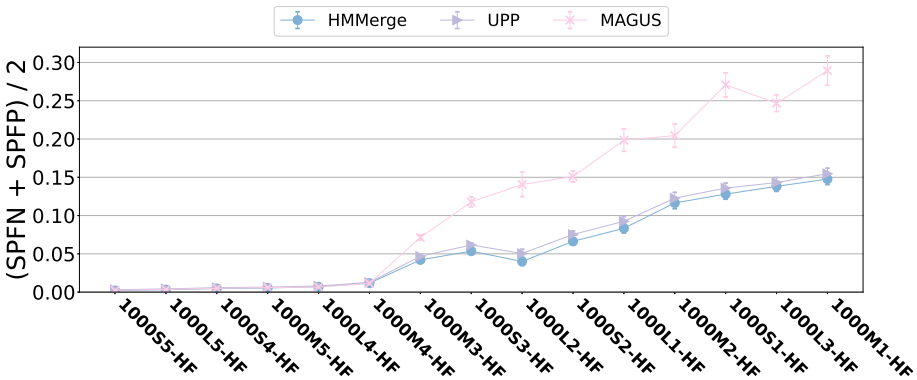

Figure 7: **Alignment error for HMMerge, UPP, and MAGUS on simulated datasets** We show the alignment error (average of SPFN and SPFP) on the final alignment produced by each of the top three methods (HMMerge, UPP, and MAGUS). The error bars indicate standard error over 19 replicates.

As shown earlier in Figure 1, UPP and MAGUS are the two leading methods across all model conditions of the ROSE simulated datasets. Therefore, we only show the top three methods in this section. We observed that HMMerge was consistently more accurate than UPP, which was consistently more accurate than MAGUS.

On the CRW biological datasets (Figure 8), HMMerge, UPP, MAGUS, PASTA, and MAFFT were consistently the most accurate methods, while Clustal Omega and T-COFFEE performed the worst. MUSCLE, although achieving low alignment error on some model conditions, had the highest alignment error

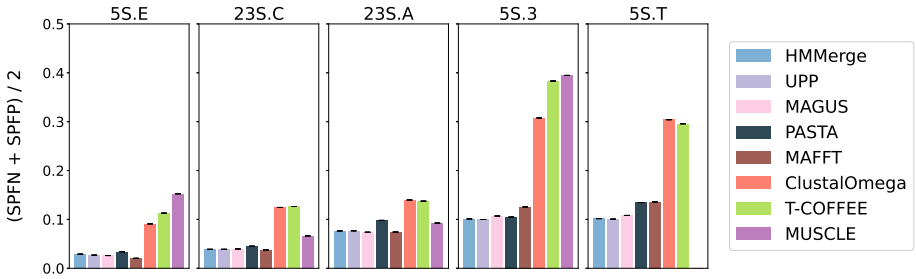

Figure 8: **Alignment error of all methods on five biological datasets** Alignment error (average of SPFN and SPFP) of the full set of MSA methods on the five biological datasets from the CRW [1].

in two of the model conditions (5S.E and 5S.3) and even failed to run in the allotted time limit on one of the model conditions (5S.T).

7 Discussion

The HMMerge pipeline is very closely related to UPP, differing from UPP only in how sequences are added into the backbone alignment. These are the two most accurate methods across all the simulation conditions we explored, achieving nearly identical accuracy to each other, with substantial improvements over the next best method, which is typically MAGUS.

A direct comparison between HMMerge and UPP is best made only with respect to the alignments they produce on the query sequences, since they are identical on the backbone sequences. Here we see that HMMerge produces more accurate alignments than UPP, with the biggest differences the conditions with high rates of evolution, which are also the conditions where alignment error tends to increase.

Overall, the strength of HMMerge in improving overall alignment error comes from its ability to improve on SPFN scores, which can make up for the increased error on SPFP on some model conditions. In general, a lower SPFN score paired with a higher SPFP score relative to another method is a sign of over-alignment. Since HMMerge aims to align the entirety of the query sequence to a segment of the backbone alignment, a low SPFN score with a high SPFP score indicates that HMMerge is emitting a lot of characters through the match states rather than its insertion states whose emissions are considered non-homologous and hence do not count as true homologies for SPFN and SPFP score calculations.

It is worth noting that on biological datasets tested, HMMerge was only able to improve on the SPFN scores by a small margin except on one dataset and had noticeably worse SPFP scores on three of the five datasets. One reason for the observed difference in results is that the biological datasets all had relatively low average p-distances. In addition, the biological datasets did not undergo an extreme kind of sequence length fragmentation such as the one that the simulated datasets underwent. This leads to an observation that HMMerge is most effective when used with datasets exhibiting high degree of fragmentation as well as high rates of evolution.

HMMerge is a method that can align query sequences which, when used in an UPP-like pipeline, shows great potential in improving the accuracy of alignments. On various simulated datasets we have tried, HMMerge was able to achieve lower overall error than any of the standard methods we tried. The improvement in alignment error was more apparent in the harder model conditions where the rate of evolution was high.

The ability of HMMerge to improve on alignment accuracy is especially noticeable when compared to the second stage of UPP. The main difference in this step is that HMMerge utilizes the information from all of the HMMs constructed from the backbone and uses the Viterbi algorithm whereas the second stage of UPP heuristically picks one HMM from the backbone and aligns using *hmmalign*, which uses a variant of the forward-backward algorithm. This difference is potentially what gives HMMerge its edge since the Viterbi algorithm optimizes the probability of an entire query sequence being emitted instead of optimizing for the maximum likelihood of individual characters being emitted at independent positions as in the forward-backward algorithm.

One important theoretical difference between HMMerge and the second stage of UPP is that although both methods rank the input HMMs, HMMerge ranks the HMMs through adjusted bit-scores, which accurately captures the probability of a query sequence being emitted by an HMM, while UPP ranks the HMMs with the raw bit-scores, which UPP uses as a heuristic to assess the importance of an HMM.

8 Conclusion

HMMerge is a multiple sequence alignment method that can be used to add query sequences to backbone alignments. When used in a full pipeline, HMMerge can be used to align datasets with sequence length heterogeneity and achieve competitive, if not better, accuracy than current leading alignment methods, albeit often slower than other methods. HMMerge is especially effective under fast evolving and highly fragmentary sequence length conditions where it excels in improving on SPFN scores, meaning that the true homologies represented by HMMerge are more reliable than other methods.

Even though we used a subset set size of 50 for the decomposition and used a centroid-edge disjoint decomposition, meaning that we recursively split at a centroid edge and only kept the leaves of the subsets, the HMMerge algorithm naturally works with any kind of decomposition sizes and types. Namely, it is able to handle arbitrary sized subsets that do not have to be disjoint and can be hierarchical as is the default in UPP. It is possible that varying the decomposition size and type could lead to accuracy and runtime improvements.

9 Acknowledgements

This work was funded in part by a subaward from Sandia National Labs (LDRD to Kelly Williams) and by the US National Science Foundation 2006069.

References

[1] Jamie J Cannone, Sankar Subramanian, Murray N Schnare, James R Collett, Lisa M D'Souza, Yushi Du, Brian Feng, Nan Lin, Lakshmi V Madabusi, Kirsten M Müller, et al. The Comparative RNA Web (CRW) Site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC Bioinformatics*, 3(1):1–31, 2002.

[2] Sean R Eddy. Accelerated Profile HMM searches. *PLoS Computational Biology*, 7(10):e1002195, 2011.

[3] Robert C Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[4] Edgar Garriga, Paolo Di Tommaso, Cedrik Magis, Ionas Erb, Leila Mansouri, Athanasios Baltzis, Hafid Laayouni, Fyodor Kondrashov, Evan Floden, and Cedric Notredame. Large multiple sequence alignments with a root-to-leaf regressive method. *Nature Biotechnology*, 37(12):1466–1470, 2019.

[5] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14):3059–3066, 2002.

[6] Kevin Liu, Sindhu Raghavan, Serita Nelesen, C Randal Linder, and Tandy Warnow. Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science*, 324(5934):1561–1564, 2009.

[7] Frederick A Matsen, Robin B Kodner, and E Virginia Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC bioinformatics*, 11(1):1–16, 2010.

[8] Siavash Mirarab, Nam Nguyen, Sheng Guo, Li-San Wang, Junhyong Kim, and Tandy Warnow. PASTA: Ultra-Large Multiple Sequence Alignment for Nucleotide and Amino-Acid Sequences. *Journal of Computational Biology*, 22(5):377–386, 2015.

[9] Siavash Mirarab and Tandy Warnow. FastSP: linear time calculation of alignment accuracy. *Bioinformatics*, 27(23):3250–3258, 2011.

[10] Lam-Tung Nguyen, Heiko A. Schmidt, Arndt von Haeseler, and Bui Quang Minh. IQ-TREE: A Fast and Effective Stochastic Algorithm for Estimating Maximum-Likelihood Phylogenies. *Molecular Biology and Evolution*, 32(1):268–274, 11 2014. `arXiv:https://academic.oup.com/mbe/article-pdf/32/1/268/13171186/msu300.pdf`, `doi:10.1093/molbev/msu300`.

[11] Nam-phuong D Nguyen, Siavash Mirarab, Keerthana Kumar, and Tandy Warnow. Ultra-large alignments using phylogeny-aware profiles. *Genome Biology*, 16(1):1–15, 2015.

[12] Morgan N. Price, Paramvir S. Dehal, and Adam P. Arkin. FastTree 2 – Approximately Maximum-Likelihood Trees for Large Alignments. *PLOS ONE*, 5(3):1–10, 03 2010. `doi:10.1371/journal.pone.0009490`.

[13] Chengze Shen, Minhyuk Park, and Tandy Warnow. WITCH: improved multiple sequence alignment through weighted consensus HMM alignment. *Journal of Computational Biology*, 2022.

[14] Chengze Shen, Paul Zaharias, and Tandy Warnow. MAGUS+eHMMs: improved multiple sequence alignment accuracy for fragmentary sequences. *Bioinformatics*, 38(4):918–924, 11 2021. `arXiv:https://academic.oup.com/bioinformatics/article-pdf/38/4/918/42319144/btab788.pdf`, `doi:10.1093/bioinformatics/btab788`.

[15] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[16] Fabian Sievers, Andreas Wilm, David Dineen, Toby J Gibson, Kevin Karplus, Weizhong Li, Rodrigo Lopez, Hamish McWilliam, Michael Remmert, Johannes Söding, et al. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(1):539, 2011.

[17] Vladimir Smirnov and Tandy Warnow. MAGUS: Multiple sequence Alignment using Graph clUStering. *Bioinformatics*, 37(12):1666–1672, 2021.

[18] Vladimir Smirnov and Tandy Warnow. Phylogeny estimation given sequence length heterogeneity. *Systematic Biology*, 70(2):268–282, 2021.

[19] Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 01 2014. `arXiv:https://academic.oup.com/bioinformatics/article-pdf/30/9/1312/17345185/btu033.pdf`, `doi:10.1093/bioinformatics/btu033`.

A Method Versions and Commands

UPP Version: 4.5.2
Availability: https://github.com/smirarab/sepp

Listing 1: UPP Command

```
python run_upp.py -c <Config file>
```

Listing 2: UPP Config File

```
[commandline]
tree=<FastTree backbone tree>
alignment=<MAGUS backbone alignment>
sequence_file=<Unaligned query sequence file>
backboneSize=<backbone size>
alignmentSize=2
molecule=dna
cpu=16
outdir=<Output directory>
tempdir=<Temporary output directori>
```

MAGUS Version: 0.1.0b2
Availability: https://github.com/vlasmirnov/MAGUS

Listing 3: MAGUS Command

```
python <git root>/magus.py -d <Output directory> -i \
<Unaligned sequences> -o <Output filename>
```

PASTA Version: 1.9.0
Availability: https://github.com/smirarab/pasta

Listing 4: PASTA Command

```
python run_pasta.py -i <Unaligned sequences> --num-cpus 16 \
-o <Output directory> --temporaries <Temporary output directory>
```

MAFFT Version: 7.487
Availability: https://mafft.cbrc.jp/alignment/software/linux.html

Listing 5: MAFFT Command

```
linsi --thread 16 <Unaligned sequences> 1> <Output filename>
```

Clustal Omega Version: 1.2.4
Availability: http://www.clustal.org/omega/#Download

Listing 6: Clustal Omega Command

```
clustalo --threads=16 --in <Unaligned sequences> --out \
<Output filenanme>
```

T-COFFEE Version: 13.45.0.4846264
Availability: https://www.tcoffee.org/Packages/Stable/Latest/

Listing 7: T-COFFEE Command

```
t_coffee -thread=16 -reg -seq <Unaligned sequences> -outfile \
<Output filename>
```

MUSCLE Version: 3.8.31
Availability: https://drive5.com/muscle/downloads_v3.htm

Listing 8: MUSCLE Command

```
muscle -in <Unaligned sequences> -out <Output filename>
```

FastSP Version: 1.7.1
Availability: https://github.com/smirarab/FastSP
Note: When evaluating MAFFT estimated alignments, the ml and mlr flags should be omitted.

Listing 9: FastSP Command

```
java -jar FastSP.jar -r <Reference alignment> -e \
<Estimated alignment> -ml -mlr
```

HMMerge Commit ID: 2e0ff12c3a311673e8bea2647ee9d1b694bd203d
Availability: https://github.com/MinhyukPark/HMMerge

Listing 10: HMMerge Command

```
python <git root>/main.py --input-dir <Directory with \
partitioned backbone alignments> --backbone-alignment \
<Backbone alignment> --query-sequence-file \
<Query sequences> --output-prefix <Output directory> \
--num-processes 16 --model DNA
```