Subject Section

MAGUS: Multiple Sequence Alignment using Graph Clustering

Vladimir Smirnov1,* and Tandy Warnow1,*

1Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, 61801, US
*To whom correspondence should be addressed.

Abstract

Motivation: The estimation of large multiple sequence alignments (MSAs) is a basic bioinformatics challenge. Divide-and-conquer is a useful approach that has been shown to improve the scalability and accuracy of MSA estimation in established methods such as SATé and PASTA. In these divide-and-conquer strategies, a sequence dataset is divided into disjoint subsets, alignments are computed on the subsets using base MSA methods (e.g., MAFFT), and then merged together into an alignment on the full dataset.

Results: We present MAGUS, Multiple sequence Alignment using Graph cluSTERing, a new technique for computing large-scale alignments. MAGUS is similar to PASTA in that it uses nearly the same initial steps (starting tree, similar decomposition strategy, and MAFFT to compute subset alignments), but then merges the subset alignments using the Graph Clustering Merger (GCM), a new method for combining disjoint alignments that we present in this study. Our study, on a heterogeneous collection of biological and simulated datasets, shows that MAGUS produces improved accuracy and is faster than PASTA on large datasets, and matches it on smaller datasets.

Availability: MAGUS: https://github.com/vlasmirnov/MAGUS

Supplementary information: Supplementary data are available at Bioinformatics online.

1 Introduction

Multiple sequence alignment (MSA) is a basic step in many bioinformatic pipelines, but accurate estimation of MSAs on large sequence datasets is challenging, especially for datasets characterized by low sequence identity and high rates of insertions and deletions (indels). Methods that scale to large datasets have been developed (Sievers et al., 2011; Lassmann, 2019; Garriga et al., 2019; Liu et al., 2009, 2012; Mirarab et al., 2015; Nguyen et al., 2015), some of which (e.g., SATé (Liu et al., 2009), SATé-II (Liu et al., 2012), and PASTA (Mirarab et al., 2015)) use divide-and-conquer: they decompose the input sequence dataset into subsets, align each subset using a base alignment method of choice (such as MAFFT (Katoh et al., 2005)), and merge the alignments together (thus treating them as constraints) into an alignment on the full dataset.

There are multiple benefits to divide-and-conquer: scalability to large datasets is one advantage, but improved accuracy is the main draw. The reason for this improvement is that most alignment methods, including top-performing methods like MAFFT, either degrade in accuracy with high rates of evolution and number of sequences, or simply cannot run on large datasets. Divide-and-conquer avoids this by only using these methods on smaller, more closely related subsets of sequences. Because the alignments on such subsets tend to be very accurate, they are treated as constraints. Hence, the accuracy of the final alignment significantly depends on the ability to merge constraint alignments. A tree can then be estimated on the alignment, and then the process can be iterated. By default, SATé-I, SATé-II, and PASTA each run for three iterations.

SATé-II improved on SATé-I by changing the decomposition strategy and PASTA improved on SATé-II by changing the merger step. Each of these changes improved accuracy and scalability, so that PASTA is the state-of-the-art for this family of divide-and-conquer MSA pipelines. Furthermore, UPP (Nguyen et al., 2015), which is designed to produce good alignments in the presence of sequence length heterogeneity, uses PASTA to compute a “backbone” alignment on sequences that are deemed “full length”, represents the backbone alignment using an ensemble of profile Hidden Markov Models, and then aligns the remaining sequences to the backbone alignment using the ensemble. Thus, while PASTA (like most other methods) is suitable for datasets without substantial sequence length heterogeneity, it is also used in methods for aligning heterogeneous sequence length datasets.
Here we present MAGUS, a new method to compute large-scale MSAs. MAGUS follows the same initial steps as the first iteration of PASTA, which has the following structure. PASTA computes an alignment using a fast technique (similar to UPP, but using a random selection of sequences for the backbone without consideration for sequence length) and FastTree2 (Price et al., 2010) to compute a tree on the alignment, divides the sequences into disjoint subsets by deleting edges until all the subsets are below the maximum allowed (default 200), computes alignments on the subsets (default MAFFT), and then merges the alignments together (merging pairs of constraint alignments using Opal (Wheeler and Keciccioglu, 2007) or Muscle (Edgar, 2004) and then completing the merger using transitivity). PASTA then computes a tree on the merged alignment using FastTree, and can continue by iterating; by default, PASTA iterates three times. The two main ways that MAGUS differs from PASTA is that it combines the disjoint alignments using the Graph Clustering Merger (i.e., GCM, a new technique we present here) and it does not need to iterate.

We explore MAGUS on a heterogeneous collection of sequence datasets, using both simulated and biological datasets. We find that just one iteration of MAGUS produces more accurate alignments than default PASTA (which uses three iterations) on nearly all datasets we explore; furthermore, MAGUS is faster, since it only needs one iteration.

2 Approach

We begin with a description of the overall strategy, and then discuss the merger step (where we ran GCM) in detail.

2.1 Divide-and-conquer strategy

Given an input set $S$ of unaligned sequences:

- Step 1: Compute a starting tree for $S$. In PASTA, 100 sequences are selected at random and aligned; the remaining sequences are added into that alignment, and a tree is computed on the final alignment with FastTree. In MAGUS, we follow nearly the same technique, except we select 300 sequences whose lengths are within 25% of 75th percentile; the hope is that longer sequences will reduce the negative impact of short sequences in the input, and using more sequences will improve the accuracy of the starting alignment and tree.

- Step 2: Decompose the sequence dataset into disjoint subsets using the centroid edge decomposition (an option within PASTA), modified to enable the maximum number of subsets to not exceed a specified amount.

- Step 3: Compute alignments on the subsets; we use MAFFT -L-ins-i, the same technique as in PASTA.

- Step 4: Merge the alignments together using GCM, as described in Section 2.2.

As PASTA enables the user to specify algorithmic parameter values for each of these steps, we expose them as well. Our experimental study (see below) presents results for different settings.

2.2 GCM

The main difference between PASTA and MAGUS is the use of GCM, which is the subject of this section. We begin by defining the MSA Merger problem and basic terminology that we will use throughout the paper.

The MSA Merging problem

- Input: a set $S = \{S_1, S_2, \ldots, S_k\}$ of MSAs, with $S_i$ an MSA on a set $X_i$ of sequences, $X_i \cap X_j = \emptyset$ if $i \neq j$, and $X = \cup_i X_i$.

- Output: An alignment $A$ on the full set $X$ of sequences that agrees with each of the input alignments; thus, the subalignment of $A$ induced on $X_i$ is identical to $S_i$ (after removing columns containing only gaps).

We will refer to the input set of alignments as “constraint alignments,” to emphasize that the input set is treated as absolute constraints that must be induced by the merged alignment. This is always possible, as the constraint alignments are disjoint. The MSA Merging problem is a straightforward generalization of the well-studied problem of aligning two alignments (Wheeler and Keciccioglu, 2007).

High-level description of the GCM algorithm.

The Graph Clustering Merger (GCM) can be used with nucleotide sequences or with amino acid sequences, and we will describe and test GCM for both types of input; hence, when we say “letter” in a sequence, we allow this to refer to either a nucleotide or an amino acid.

The basic idea of GCM is to merge our constraint alignments by using a library of “backbone alignments” (also called “backbones”), which we compute using a selected “base MSA method” (e.g., MAFFT) after sampling representative sequences from each constraint alignment. This approach is related to the MSA methods in Garriga et al. (2019) and Pei and Grishin (2007), each of which merges a set of disjoint alignments by picking one representative sequence from each subset, aligning the set of selected sequences, and then using that new alignment to construct the merged alignment. However, in these methods, exactly one representative sequence per subset is selected and only one “backbone” alignment of representative sequences is computed, so that there’s a trivial merged alignment that induces all the subset alignments.

![Diagram](https://example.com/gcm_diagram.png)

Fig. 1. Overview of the GCM algorithm. The input is a set of constraint alignments. 1) We compute a set of backbone alignments with the same number of sequences from each constraint alignment. 2) We construct the alignment graph. Each node represents a column from a constraint alignment, and weighted edges are derived from homologies inferred in the backbone alignments: thicker lines represent edges with higher weight. 3) We cluster the alignment graph with MCL. This example shows two violations: the pink-outlined cluster contains two columns from the same (blue) constraint alignment, and there is no valid ordering between the pink- and green-outlined clusters (“crisscrossing”). 4) We resolve the violations and produce a valid trace, where each connected component is a column in our final alignment.

We depart from these approaches by using more than one backbone, with each having more than one representative sequence from each constraint alignment. The backbone alignments are not constrained to be consistent with the input constraint alignments or with each other. Thus, in GCM there is much more information to use in merging the constraint alignments, but GCM must also deal with conflicts among the backbone and constraint alignments. This use of multiple backbone alignments introduces a whole new layer of challenges compared to these earlier approaches.
Phase 1 computes a library of backbone MSAs that is then used to form a graph, where the vertices correspond to columns within the input constraint MSAs. In keeping with terminology introduced in previous literature (Kececioglu, 1993), we refer to this as the “alignment graph.” In Kececioglu (1993), each node in an alignment graph corresponds to one letter from one sequence; however, since we are “aligning alignments”, the nodes in our alignment graphs will represent columns of letters from our constraint alignments instead.

Phase 2 clusters the vertices using the Markov Clustering (MCL) technique from Van Dongen (2000), which can be seen as extending the consistency principle in the MSA literature to longer paths in a way that is similar to the technique in Probcons (Do et al., 2005) (please refer to the Supplementary Materials for a more detailed explanation). This results in a set of clusters, each of which is a set of nodes (vertices) in the graph, where the nodes correspond to columns from the constraint alignments. In order for this clustering to define a valid MSA (also called a “trace” in Kececioglu (1993)), for each cluster, the set of associated columns must contain at most one column from any constraint alignment, and it must be possible to order the clusters (from left-to-right) so that there are no “crossings”. Hence, the clustering obtained using MCL is only an initial guess for a final “trace” that will give us a legal merged MSA. Consequently, Phase 3 “fixes and orders” the clusters (breaking them apart, if needed) to form a legal MSA.

**Phase 1: Alignment Graph Construction.** We pick a number \( M \) of backbone alignments and the target size \( L \) per backbone alignment. For each backbone, we randomly select the same number \( \frac{L}{2} \) of sequences from each constraint alignment (where \( q \) is the number of constraint alignments), to achieve approximately the desired target size \( L \). We then compute alignments on each backbone set of sequences, thus producing the backbone alignment for that set, and the full set of such backbone alignments forms the backbone library.

Given the backbone library, we construct the “alignment graph”, where the vertices correspond to columns (i.e., sites) in the constraint alignments, and edges are derived from the backbone alignments, as we now describe. Given a column \( j \) in constraint alignment \( i \), we create vertex \( S_{ij} \). The edge between vertices \( S_{ij} \) and \( S_{ik} \) is included if and only if there is at least one pair of letters from constraint alignment columns \( S_{ij} \) and \( S_{ik} \) in the same column in a backbone alignment, and the weight of each such edge is the number of such pairs. Self-loops are retained, as well as edges between different vertices from the same constraint alignment.

**Phase 2: Clustering the Graph.** The next phase takes the undirected graph \( G \) from Phase 1 and partitions the vertices into disjoint clusters. In essence, the purpose of this clustering phase is two-fold: (1) to make analysis of the alignment graph more tractable through stochastic flow, or the chief algorithmic parameter in MCL is the inflation factor \( I \), which roughly controls the granularity of the clustering in GCM; the guidelines for choosing \( I \) in Von Dongen (2012) suggest four values between 1.4 and 6.

**Phase 3: Clean-up to produce a MSA.** The final phase is meant to modify the clustering from the previous step to yield a valid MSA on the full set \( X \) of sequences. Our objective is to transform our given set of clusters (connected components) into an ordered set of (possibly different) clusters that form a valid trace. Thus, our task is somewhat similar to the Maximum Weight Trace problem (Kececioglu, 1993). We need our modified set of clusters to satisfy two trace conditions. The first condition is the absence of vertical violations, where some \( S_{ij} \) and \( S_{ik} \) (i.e., two columns from the same constraint alignment) appear together in a cluster (if the clustering algorithm permits overlapping clusters, we must also deal with “horizontal” violations, where the same \( S_{ij} \) appears in more than one cluster). The second condition requires a valid ordering, so that if cluster \( C \) comes before cluster \( C' \), then \( j < k \) for every \( S_{ij} \in C \) and \( S_{ik} \in C' \).

Producing this new set of ordered clusters may be achievable just through ordering the original set of clusters (produced in Phase 1), or may require breaking some of the original clusters into two or more parts. Our optimization criterion is to satisfy our requirements using the smallest number of cluster breaks. In other words, we want to retain the integrity of our constraint alignment column matching as much as possible, while teasing a valid trace out of it.

**Step 1: Eliminate cluster violations.** We use a greedy approach, which can require breaking clusters apart.

- While violations exist:
  - Find the pair \( C, C' \), where \( C \) is a cluster and \( C' \) is a horizontally or vertically violating element in \( C \) that has the smallest sum-of-weights between \( S_{ij} \) and the other elements in \( C \) (excluding elements from the same constraint alignment).
  - Remove \( S_{ij} \) from \( C \).

The removal of \( S_{ij} \) from the cluster \( C \) creates a new singleton cluster. However, singleton clusters (which define single columns in some constraint alignment) are not needed in the next step, so we will throw out all singleton clusters as we proceed to Step 2. Afterwards, we can reinsert all of the removed columns in their proper locations (and not homologous to any other columns) using the ordering inferred in Step 2.

**Step 2: Order (and possibly fragment) the clusters.** With these cluster violations removed, it remains to produce a legal ordering, which can also require breaking clusters apart; this proves to be the most non-trivial component of the algorithm. We formulate this task as a search problem, as we now describe. Each state in the search space is defined by two sets of clusters—a set of clusters that have already been ordered, and a set of remaining unordered clusters. Then, our starting state has an empty ordered set, and all given clusters in the unordered set. An end state is any state with an empty unordered set, meaning that the ordered set contains a complete, valid ordering. This state space has an exponential number of states, and so we do not construct the space explicitly, but rather perform an implicit search from the starting state.

We transition from one state to another by removing a cluster from the unordered set and adding it to the ordered set, if such a move is legal. A cluster \( X \) in the unordered set can only be legally moved into the ordered set if it comes strictly “before” every other cluster in the unordered set, element-wise. More formally, we say a move from state \( A = (C_1, U_1) \) to \( B = (C_2, U_2) \) (where \( C_1 \) is a set of ordered clusters, and \( U_1 \) is a set of unordered clusters) is legal if \( C_2 = C_1 \cup \{X\} \) and \( U_2 = U_1 \setminus \{X\} \), and for all \( S_{ij} \in X \), there is no \( S_{ik} \) with \( k < j \) belonging to any cluster in \( U_2 \). If no legal moves exist, we can transition to another state by breaking one of the unordered clusters apart into two unordered clusters. We want to travel from our starting state to an end state with the smallest number of cluster breaks, which is our measure of distance.

We use the A* algorithm (Hart et al., 1968), a mainstay in pathfinding and graph search applications, to perform the search. A* is an extension of Dijkstra’s algorithm that uses both distance and heuristic measures to
guide the search; in our case, the distance measure is simply the size of the ordered set, and the heuristic measure is the size of the unordered set (thus being a lower bound on the remaining distance to travel). This suffices to guarantee that A* will eventually find an optimal path.

Unfortunately, if the given clusters are poorly formed and extensively "tangled", A* will not be able to find an optimal path in a reasonable amount of time. We handle such situations by weighting the heuristic more heavily if progress is not being made (i.e. we switch to Weighted A* (Pearl, 1984)). This revokes the optimality guarantee, but allows A* to finish more quickly while still finding a reasonable solution.

Once A* finishes, we order the clusters on the clusters to construct the MSA: each cluster contains a set of constraint alignment columns, which we stack together to form a column in the final MSA. The ordering of the columns follows the ordering of the clusters, with gaps added wherever needed. Finally, as noted above, the singleton clusters that were deleted in Step 1 are then inserted into the correct location (defined by the ordering), without making them homologous to any other columns.

**Runtime Complexity.** The runtime of MAGUS is dominated by building backbone alignments, which we employ using MAFFT-L-ins-i. This portion is \(O(B_n^2 B_m^2 l^2)\), where \(B_n\) is the number of backbone alignments, \(B_m\) is their average length, and \(B_m^2\) is the maximum number of sequences in each backbone, and follows from the runtime complexity of MAFFT-L-ins-i (Katoh and Toh, 2008). For additional details, see the Supplementary Materials.

### 3 Evaluation

**Overview.** We performed three experiments. Experiment 1: We set default algorithmic parameters for a fast variant and a slower variant of MAGUS. Experiment 2: We compare both the fast and slow variants of MAGUS to PASTA run for varying numbers of iterations. Experiment 3: We evaluate the impact of following a PASTA analysis with a single iteration where GCM replaces Opal or Muscle for the alignment merger technique. We perform our evaluation study on the UIUC Campus Cluster. Our analyses are run on nodes with 16 cores, 64GB of memory, and a walltime limit of 4 hours (except for 16S.B.ALL, where we use high memory nodes with a limit of 48 hours). We provide an overview of representative results here. Complete and additional results are presented in the Supplementary Materials.

**Datasets.** Our evaluation explores a heterogeneous collection of simulated and biological datasets from prior studies, available in public websites (see Smirnov (2020)). For simulated datasets, we use ten 1000-sequence model conditions from Liu et al. (2009), ranging from 1000M1 (the most difficult) to 1000M4 (relatively easy). We also use 1,000 and 10,000-sequence subsets of the RNASim dataset from Mirarab et al. (2015), which evolve under a non-standard model with positive selection. For the biological datasets, we include four large nucleotide datasets from Cannone et al. (2002) and eight protein datasets from Balibase (Thompson et al., 1999), where reference alignments are available based on structural features. Since PASTA (and most other alignment methods, other than UPP) is restricted to datasets without sequence length heterogeneity, these biological datasets were first modified to remove the outlier sequences (i.e., the ones that deviate in length from the median by more than 20%). The modified biological datasets range in size from 195 sequences to 24,246 sequences, and vary in the average pairwise sequence identity, percent of the matrix occupied by gaps, and other empirical properties. See Table 1 for additional details.

**Criteria** We use SPFN and SPPF, computed using FastSP (Mirarab and Warnow, 2011). SPFN denotes the Sum-of-Pairs False Negative rate, which is the fraction of pairs of homologous letters in the true alignment that are not aligned in the estimated alignment. SPPF is the corresponding Sum-of-Pairs False Positive rate, the fraction of pairs of aligned letters in the estimated alignment that are not in the true alignment. SPFN and SPPF are related to the SP-Score and Modeler scores (used sometimes in protein alignment evaluation) by the formulae: \(\text{SP-score} = 1-\text{SPFN}\) and \(\text{Modeler score} = 1-\text{SPPF}\). Due to space constraints, we show SPFN and SPPF in the supplementary materials, and report their average in the main paper.

**Runtime.** Runtimes are reported for each process running start-to-finish on a single cluster node, with both PASTA and MAGUS exploiting multi-threading in a similar way (i.e. the MAFFT-L-ins-i runs are parallelized).

### 4 Results

**4.1 Experiment 1: Parameter tuning for MAGUS**

MAGUS has multiple algorithmic parameters: MCL inflation parameter for GCM, starting tree, number of constraint subsets, number and size of backbone alignments, and whether the backbone alignments are extended to full alignments using HMMER (Eddy, 2020) (see Supplementary Materials). We performed extensive experiments on the 1000M1 model condition to evaluate the impact of how we set these parameters; we excluded one of the 20 replicates as a persistent outlier, and used performance on the other replicates for this experiment. This experiment determined default settings for two modes of MAGUS: a fast version, which does not include the HMM-extension technique, and a slow version that does. Both versions use the same settings for the other parameters: MCL inflation parameter 4, a starting tree that we have developed, 100 backbones of size 200, and divide the input sequence set into 25 subsets for constraint alignment creation. Here we show how changes to these selected parameters impact accuracy and running time. (The results for different inflation factors are provided in the supplementary materials, and are less interesting as they follow a standard recommendation in any event.)
In this experiment we show the impact of changing the number of constraint alignments. We vary the number of constraint alignments from 5 to 300, and the number of subsets from 5 to 100. The error rates are the average of SPFN and SPFP, and are averaged over the 19 replicates of the 1000M1 model condition. Error bars indicate standard error. The y-axis shows the impact of changing the number of subsets. The x-axis shows the impact of changing the number of subsets.

Experiment 1(a): Varying the size and number of backbone alignments and number of constraint alignments. We vary the number of constraint alignments, and the number and size of our backbones (Figure 2). We use the MAGUS starting tree and fix the inflation parameter to 4. Note that the best accuracy is obtained using 25 constraint alignments and 15 backbones of size 200 or 300. Given the use of 25 constraint alignments and 15 backbones, there is no difference in accuracy between backbones of size 200 and 300, but that change does impact the running time. However, smaller backbones definitely increase error, and fewer than 10 backbones also increase error noticeably. There is only a small decrease in accuracy when using 10 instead of 15 backbone alignments, and so if running time is considered, the best default setting seems to be 25 constraint alignments and 10 backbones of size 200. If time permits, then 15 or even 20 backbones can be used, and such a setting might reasonably be used on much larger datasets, where the computation of constraint alignments would dominate the runtime anyway.

Experiment 1(b): Varying the starting tree, number of constraint alignment subsets, and HMM-extension. In this experiment we show the impact of the choice of starting tree and the use of HMM-extension (see Supplementary Materials for commands), while allowing the number of constraint alignments to change; all analyses use inflation factor 4 and we use 10 backbones of size 200.

There is an advantage in alignment accuracy to using the MAGUS starting tree instead of the PASTA starting tree, and this advantage is present at every number of constraint alignments we examine (Fig. 3). The impact of using the HMM-extension to complete the backbone alignments ("slow" mode) is generally positive—improving accuracy with 25 or more constraint alignments and neutral on fewer constraint alignments. However, using the HMM-extension increases the runtime (e.g., the median runtime here increases by as much as ten minutes). However, of perhaps greater concern is the high variance in the runtime when using the HMM-extension, which suggests that it may become very computationally expensive to use on large datasets.

4.2 Experiment 2: MAGUS vs. PASTA

The central experiment of our study is the comparison between MAGUS and PASTA. MAGUS(Fast) and MAGUS(Slow) differ in the use of the HMM-extension, but otherwise both use 25 constraint alignments and 10 backbones of size 200, and the same starting tree. Of the ten model conditions with 1000 sequences (Fig. 4), both versions of MAGUS have better accuracy on nine conditions and tie with PASTA on one condition. The largest differences are on the hardest model condition, 1000M3, where MAGUS (both versions) leads PASTA by about 6.7% (10–11% vs. 17–18%). The difference narrows, but persists, under easier model conditions. The use of MAGUS(Slow) provides an accuracy benefit under the harder model conditions, and otherwise has very similar accuracy as MAGUS(Fast). Moreover, MAGUS(Slow) and MAGUS(Fast) are both much faster than PASTA, generally completing in about half the time (e.g., a reduction from about 40 minutes for PASTA to about 15 minutes for MAGUS on 1000M2, and from about 20 minutes to about 10 minutes on 1000M3). Lastly, the error bars indicate that MAGUS tends to have noticeably lower variability in its accuracy and runtime than PASTA under the harder model conditions.

On the larger datasets, ranging from 5,489 to 24,246 sequences (Fig. 5), MAGUS’s accuracy is tied with PASTA on the 165.3 dataset, but is about 1–2% better on the other three datasets. MAGUS is also much faster, often finishing in half the time as PASTA (e.g., on the 165.8 ALL dataset with over 24,000 sequences, MAGUS completes in 3 hours to process.
PASTA’s 6). The differences between MAGUS and PASTA on small datasets (see Supplementary Materials, Figures 2–4) are generally small, but also show an advantage for MAGUS: out of the 9 datasets, there are 3 with a noticeable accuracy advantage for MAGUS (both Slow and Fast). PASTA and MAGUS(Fast) are effectively tied on the other 6 datasets, and MAGUS(Slow) is slightly worse on one of them. Notably, MAGUS(Slow) doesn’t enjoy any accuracy advantage over MAGUS(Fast). In terms of runtime, MAGUS is slower than PASTA on the smallest dataset (17 min vs. 9 min), but this difference reverses by the time we reach the largest of the small datasets (10 min vs. 18 min).

### 4.3 Experiment 3: Evaluating PASTA+GCM

Recall that default PASTA runs for three iterations; here we explore the impact of using GCM within PASTA instead of the standard options in PASTA for merging sets of alignments. Thus, we compare three pipelines: PASTA(3) (i.e., default PASTA, which runs for three iterations), PASTA(4) (i.e., running PASTA in default mode but allowing an additional iteration), and PASTA(3)+GCM (i.e., following PASTA(3) by an iteration where we replace the default merger strategy in PASTA by GCM).

PASTA(3)+GCM outperforms PASTA(3) and PASTA(4) on the ten 1000-sequence model conditions (Fig. 6 (top)). The improvement is largest for the most difficult model condition (1000L3) and this difference decreases as the rates of evolution decrease, so that PASTA(3)+GCM is strictly better on the nine hardest conditions and marginally worse on the easiest dataset. As expected, PASTA(3) and PASTA(4) are very close in accuracy, indicating that there is practically no accuracy benefit to running more than three PASTA iterations. Results on the largest datasets (Fig. 6 (bottom)) are nearly identical to their counterparts from Experiment 2: we see a tie between PASTA+GCM and PASTA on the 16S.3 dataset, and a 1–2% advantage to running a PASTA+GCM iteration on the other three datasets. As before, an additional PASTA iteration hardly makes a difference. Results on the smaller biological datasets (Supplementary materials, Figures 7 and 8) show consistent trends: the PASTA+GCM iteration yields a sizeable benefit on one dataset and small gains on three others, with the remaining five being essentially tied. Thus, dataset size influences, but does not determine, whether using GCM within PASTA improves accuracy.

### 5 Discussion

Both versions of MAGUS are generally at least as accurate as PASTA, with reliable improvements on the large, challenging datasets and a small advantage on the smaller datasets (i.e., MAGUS provides a small accuracy advantage for about half the smaller datasets and matched PASTA on the other datasets, with MAGUS(Slow) coming off slightly worse on one), and differences in running time are often relatively small. MAGUS is also faster than PASTA on the larger datasets (with 1000 or more sequences), and the running time advantage can result in a savings of several hours on the larger datasets. The improvement in running time is easily understood, since MAGUS requires only one iteration, whereas PASTA default uses three iterations. What is more noteworthy, therefore, is the improvement in accuracy.

We did not expect to see any improvements on the small datasets, and so the general superior accuracy obtained by MAGUS compared to PASTA on these datasets is noteworthy. It is not clear why MAGUS has better accuracy on these data, and this is a question for future research. On the other hand, the improvement of MAGUS over PASTA on the larger datasets makes sense, since PASTA’s merger strategy is inherently limited: it uses Opal or Muscle to merge pairs of constraint alignments and then completes the merger using transitivity. While this approach ensures scalability, there...
is room for improvement by merging all the constraint alignments at once. Therefore, the improvements we see of MAGUS over PASTA on the larger datasets, especially when the alignments are complicated by high rates of substitutions and indels, seem fairly understandable.

6 Conclusion

MAGUS is a new method based on PASTA's divide-and-conquer strategy for large-scale multiple sequence alignment. Our study, which evaluated MAGUS and PASTA under a heterogeneous collection of datasets, shows that MAGUS typically improves on the accuracy of default PASTA (which was optimized for accuracy), particularly under more difficult model conditions. MAGUS also runs for a single iteration, as opposed to PASTA's three, which dramatically reduces the runtime. Thus, MAGUS's ability to produce more accurate alignments in a much shorter amount of time offers a convincing improvement over PASTA, a method that has been surprisingly difficult to improve. Furthermore, following PASTA with a single iteration using GCM also yields improvements (and more than a single additional PASTA iteration), thus showing that GCM itself provides an advantage over the merging strategy in PASTA.

Our study suggests several directions for future work. We only compared MAGUS to PASTA, but other methods have been developed that are designed for large datasets (e.g., Garriga et al. 2019; Lassmann 2012). Therefore, improvements we see of MAGUS over PASTA on the larger datasets, especially when the alignments are complicated by high rates of substitutions and indels, seem fairly understandable.

Funding

This work was supported in part by NSF grant ABI-1458652 to TW.

References

Camorre, J. J., Subramanian, S., Schnarr, M. N., Collet, J. R., D’Souza, L. M., Du, Y., Feng, B., Lin, N., Matubai, L. V., Müller, K. M., Pande, N., Shang, Z., Yu, N., and Guttell, R. R. (2002). The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. BMC Bioinf., 3(1).

Do, C. B., Mahabhashyam, M. S., Budru, M., and Batzoglou, S. (2005). Probcons: Probabilistic consistency-based multiple sequence alignment. Genome Res., 15(2), 330–340.

Eddy, S. R. (2020). HMMER website. https://hmmer.org/, last accessed September 25, 2020.

Edgar, R. C. (2004). Muscle: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinf., 5(1), 113.

Garriga, E., DiTommaso, P., Magis, C., Erb, I., Mansouri, L., Balitsis, A., Laayouni, H., Kondrashov, F., Foden, E., and Notredame, C. (2019). Large multiple sequence alignments with a root-to-leaf regressive method. Nature Biotech., 37(12), 1466–1470.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1986). A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cyber., 4(2), 100–107.

Katoh, K. and Toh, H. (2008). Recent developments in the MAFFT multiple sequence alignment program. Brief Bioinf., 9, 286–298.

Katoh, K., Kuma, K.-i., Toh, H., and Miyata, T. (2005). MAFFT version 5: improvement in accuracy of multiple sequence alignment. Nucleic Acids Res., 33(21), 511–518.

Kececioglu, J. (1993). The maximum weight trace problem in multiple sequence alignment. In Aori Symp Combinatorial Pattern Matching, pages 106–119. Springer.

Lassmann, T. (2019). Kalign 3: multiple sequence alignment of large datasets. Bioinf., 36(6), 1926–1929.

Li, L., Stoeckert, C. J., and Roos, D. S. (2003). OrthoMCL-identification of ortholog groups for eukaryotic genomes. Genome Res., 13(9), 2178–2189.

Liu, K., Raghavan, S., Nelsen, S., Lindner, C. R., and Warnow, T. (2013). Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. Science, 342(5493), 1561–1564.

Liu, K., Warnow, T., J, Holder, M. T., Nelsen, S. M., Yu, J., Stamatakis, A. P., and Lindner, C. R. (2012). SANE-II: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. Syst Biol., 61(1), 90.

Mirarab, S., and Warnow, T. (2011). FastSP: linear time calculation of alignment accuracy. Bioinf., 27(23), 3250–3258.

Mirarab, S., Nguyen, N., Guo, S., Wang, L.-S., Kim, J., and Warnow, T. (2015). PASTA: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. J Comp Biol., 22(3), 377–386.

Nguyen, N.-p., Mirarab, S., Kumar, K., and Warnow, T. (2015). Ultra-large alignments using phylogeny-aware profiles. Genome Biol., 16(1), 124.

Notredame, C., Higgins, D. G., and Heringa, J. (2000). T-Coffee: a novel method for fast and accurate multiple sequence alignment. J Mol Biol., 302(1), 205–217.

Pearl, J. (1984). Intelligent search strategies for computer problems solving. Addison Wesley.

Poi, J. and Grishin, N. V. (2007). PROMALS: towards accurate multiple sequence alignments of distantly related proteins. Bioinf., 23(7), 802–808.

Price, M. N., Dehal, P. S., and Arkin, A. P. (2010). FastTree 2–approximately maximum-likelihood trees for large alignments. PLoS One, 5(3), e9490.

Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karpilow, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Soding, J., et al. (2011). Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. Mol Syst Biol., 7(1), 539.

Smirnov, V. (2020). Datasets for the MAGUS study. Illinois Data Bank website, https://doi.org/10.13012/B2IDB-2643961_V1, last accessed September 25, 2020.

Thompson, J. D., Plewniak, F., and Poch, O. (1999). BAliBASE: a benchmark database of comparative sequence and structure information for ribosomal, intron, and other RNAs. BMC Bioinf., 1(1).

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.

Wesley.