Adaptive local realignment of protein sequences

Dan DeBlasio * John Kececioglu †

Abstract

While mutation rates can vary markedly over the residues of a protein, multiple sequence alignment tools typically use the same values for their scoring-function parameters across a protein's entire length. We present a new approach, called *adaptive local realignment*, that in contrast automatically adapts to the diversity of mutation rates along protein sequences. This builds upon a recent technique known as parameter advising, that finds global parameter settings for an aligner, to now adaptively find local settings. Our approach in essence identifies local regions with low estimated accuracy, constructs a set of candidate realignments using a carefully-chosen collection of parameter settings, and replaces the region if a realignment has higher estimated accuracy. This new method of *local parameter advising*, when combined with prior methods for global advising, boosts alignment accuracy as much as 26% over the best default setting on hard-to-align protein benchmarks, and by 6.4% over global advising alone. Adaptive local realignment, implemented within the `Opal` aligner using the `Facet` accuracy estimator, is available at `facet.cs.arizona.edu`.

*Corresponding author. Computational Biology Department, Carnegie Mellon University, Pittsburgh PA 15213, USA. `deblasio@cmu.edu`. Work performed while at the University of Arizona.

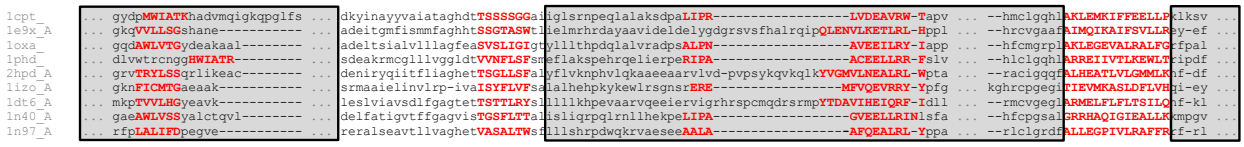
†Department of Computer Science, The University of Arizona, Tucson AZ 85721, USA. `kece@cs.arizona.edu`

Keywords Multiple sequence alignment; iterative refinement; local mutation rates; alignment accuracy; parameter advising.

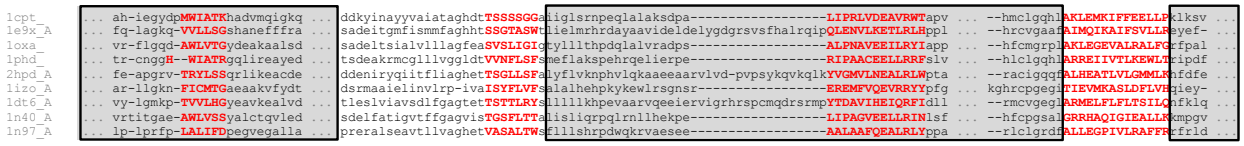
1 Introduction

Ever since the 1960s, it has been known that proteins can have distinct mutation rates at different locations along the molecule (Fitch and Margoliash, 1967). The amino acids at some positions in a protein may stay unmutated for long periods of time, while other regions change a great deal (sometimes referred to as "hypermutable regions"). This has led to methods in phylogeny construction that take variable mutation rates into account when building trees from sequences (Yang, 1993). In multiple sequence alignment, however, variation in mutation rates across sequences to our knowledge has yet to be successfully exploited to improve alignment accuracy. Multiple sequence alignments are typically computed using a single setting of values for the parameters of the alignment scoring function. This single parameter setting affects how residues across a protein are aligned, and implicitly assumes uniform mutation rates. In contrast, the approach of this paper identifies alignment regions that may be misaligned under a single parameter setting, and finds alternate settings that may more closely match the local mutation rate of the sequences.

We present a method that takes a given alignment and attempts to improve its overall accuracy by replacing sections of it with better subalignments, as demonstrated in Figure 1. The top alignment of the figure was computed using a single parameter setting: the optimal default setting of the `Opal` aligner (Wheeler and Kececioglu, 2007). The bottom alignment is obtained by our new method, taking the top alignment, automatically identifing the sections



(a) Using the optimal default parameter settings



(b) After adaptive local realignment

Figure 1: *Effect of adaptive local realignment.* Two alignments of the same region of benchmark `BB11007` from the `BAliBASE` suite, where the amino acids highlighted in red uppercase are from the so-called core columns of the reference alignment, which should be aligned in a correct alignment. (a) The alignment computed by `Opal` using its optimal default parameter setting (`VTML200`, 45, 11, 42, 40) across the sequences, with an accuracy of 89.6%. The regions of the alignment in gray boxes are automatically selected for realignment. (b) The outcome of adaptive local realignment, with an improved accuracy of 99.6%, that uses different parameters settings in each region. The realignments of the three regions use alternate parameter settings (`BLOSUM62`, 45, 2, 45, 42), (`BLOSUM62`, 95, 38, 40, 40), and (`VTML200`, 45, 18, 45, 45), respectively.

in gray boxes, and realigning them using alternate parameter settings, as described later in Section 3. This increases the overall alignment accuracy by 10%, as most of the misaligned core blocks (highlighted in red uppercase) are now corrected.

Related work

Methods that partition a set of sequences to align or realign them can be grouped in two categories, based on the orientation of their partition. *Vertical* realigners cut the input sequences into substrings, and once these shorter substrings are realigned, they stitch their alignments together. *Horizontal* realigners split an alignment into groups of whole sequences, which are then merged together by realigning between groups, possibly using the induced subalignment of each group. Realignment is occasionally called alignment *polishing*.

`Crumble` and `Prune` (Roskin *et al.*, 2011) are a pair of algorithms for performing both vertical (`Crumble`) and horizontal (`Prune`) splits on an input set of sequences. During the `Crumble` stage, a set of constraints is found that anchor the input sequences together, and the substrings or blocks between these anchor points are aligned. Once the disjoint blocks of the sequences are aligned, they are then fused by aligning their overlapping anchor regions. During the `Prune` stage, smaller groups of sequences are aligned that correspond to subtrees of the progressive aligner's guide tree. The subset of sequences in a subtree is then replaced by their alignment's consensus sequence in the remaining steps of progressive alignment. The original subalignments of the groups are finally reinserted to form the output alignment. Replacing a group of sequences by their consensus sequence during alignment reduces the number of sequences that are aligned at any one time. The objective, however, for splitting

sequences both vertically and horizontally within `Crumble` and `Prune` is not to improve accuracy, but to reduce running time and memory consumption, in order to make aligning a large number of long sequences feasible.

An early example of horizontal realignment is `ReAligner` (Anson and Myers, 1997) which improves DNA sequence assembly by removing and then realigning sequencing reads. If a read is initially misaligned in the assembly it may be corrected when realigned. This process is repeated over all reads to continually refine the assembly.

Gotoh (1993) presented several horizontal methods for heuristically aligning two multiple sequence alignments, which he called "group-to-group" alignment. This can be used for alignment construction in a progressive aligner, proceeding bottom-up over the guide tree and applying group-to-group alignment at each node, or for polishing an existing alignment by assigning sequences to two groups and realigning between the groups.

The standard alignment tools `MUSCLE` (Edgar, 2004), `MAFFT` (Katoh *et al.*, 2005), and `ProbCons` (Do *et al.*, 2005) also include a polishing step that performs horizontal realignment using ideas similar to Gotoh.

`AlignAlign` (Kececioglu and Starrett, 2004) is unique as a horizontal method in that it implements an exact algorithm for optimally aligning two multiple sequence alignments under the sum-of-pairs scoring function with affine gap costs. This optimal group-to-group alignment algorithm, used for both alignment construction and alignment polishing, forms the basis of the `Opal` aligner (Wheeler and Kececioglu, 2007).

While realignment attempts to *correct* errors in existing alignments that were made during the alignment process, several tools attempt to *avoid* making these errors in the first place by adjusting parameter values along the sequences during alignment construction. For

example, PRANK (Löytynoja and Goldman, 2008) uses a multi-level HMM that effectively chooses the alignment scoring function at each position. T-Coffee (Notredame *et al.*, 2000) and M-Coffee (Wallace *et al.*, 2006) use consistency between pairwise alignments to create position-specific substitution scores. M-Coffee extracts these pairwise alignments from a set of multiple sequence alignments, while T-Coffee generates optimal pairwise alignments. In fact, even the early tool ClustalW (Thompson *et al.*, 1994) adjusted positional gap-penalties based on pairwise sequence characteristics. Nevertheless, these tools which adjust positional alignment scores all attain lower accuracies on protein benchmarks than the aligners which make no positional adjustments that we compare against in our later computational experiments.

Adaptive local realignment, in contrast, is a vertical approach that aims to improve alignment accuracy, and a meta-method that can be applied to any aligner with tunable parameters. To our knowledge, this is the first realignment approach that automatically adapts to varying mutation rates along a protein and successfully achieves a demonstrable improvement in alignment accuracy.

Plan of the paper

The next section provides the necessary background on parameter advising, which is basic to our adaptive local realignment technique. A parameter advisor selects a parameter setting for an aligner from a small set of choices drawn from a larger universe of all possible settings, using an alignment accuracy estimator to select its choice. Section 3 then describes our adaptive local realignment method, which can be viewed as a form of local parameter

advising, and discusses how it interacts with global parameter advising. Section 4 experimentally evaluates our approach, and compares it with prior methods for advising. Finally, Section 5 gives conclusions and offers directions for further research.

2 Background on parameter advising

To make the paper self-contained, we briefly review our prior work on parameter advising. We first review the concept of a parameter advisor, which requires an estimator of alignment accuracy and a set of parameter choices for the advisor, and then summarize our prior techniques for learning both an estimator and an advisor set. An extensive discussion of parameter advising for multiple sequence alignment is in DeBlasio and Kececioglu (2017c).

We emphasize that while this section describes how to find an accuracy estimator and advisor set based on training examples, in practice a user of parameter advising will simply apply an advisor with a precomputed accuracy estimator and advisor set, and will not invoke the training procedures described here.

2.1 Global parameter advising

The goal of parameter advising is to find the parameter setting for an aligner that yields the most accurate alignment of a given set of input sequences. The accuracy of a computed alignment is measured with respect to the "correct" alignment of the sequences (which often is not known). For special benchmark sets of protein sequences, the gold-standard alignment of the proteins, called their *reference alignment*, is usually obtained through structural alignment by finding the best superposition of the known three-dimensional structures of

the proteins. Columns of the reference alignment that contain a residue from every protein in the set (where a *residue* is the amino acid at a particular position in a protein), and for which the residues in the column are all mutually close in space in the superposition of the structures, are called *core columns*. Runs of consecutive core columns are called *core blocks*, which represent the regions of the structural alignment with the highest confidence of being correct. Given such a reference alignment with identified core blocks, the *accuracy* of an alternate computed alignment is the fraction of the pairs of residues aligned in the core blocks of the reference alignment that are also aligned in the computed alignment. (Note that while a computed alignment of 100% accuracy must completely agree with the reference on its core blocks, it may disagree elsewhere.) The best computed alignment is one of highest accuracy, and the task of a parameter advisor is to find a setting of the tunable parameters of an aligner that yields an accurate output alignment. This setting can be highly input dependent, as the best choice of parameter values for an aligner can vary for different sets of input sequences.

When aligning sequences in practice, a reference alignment is almost never known, in which case the true accuracy of a computed alignment cannot be measured. Instead a parameter advisor relies on an accuracy *estimator* E that for an alignment A, gives a value $E(A)$ in the range $[0, 1]$ that estimates the true accuracy of alignment A. An estimator should be efficiently computable and positively correlated with true accuracy.

To choose a parameter setting, an advisor takes a set of choices P, where each *parameter choice* $p \in P$ is a vector that assigns values to all the tunable parameters of an aligner, and picks the choice that yields a computed alignment of highest estimated accuracy.

Formally, given an accuracy estimator E and a set P of parameter choices, a *parameter*

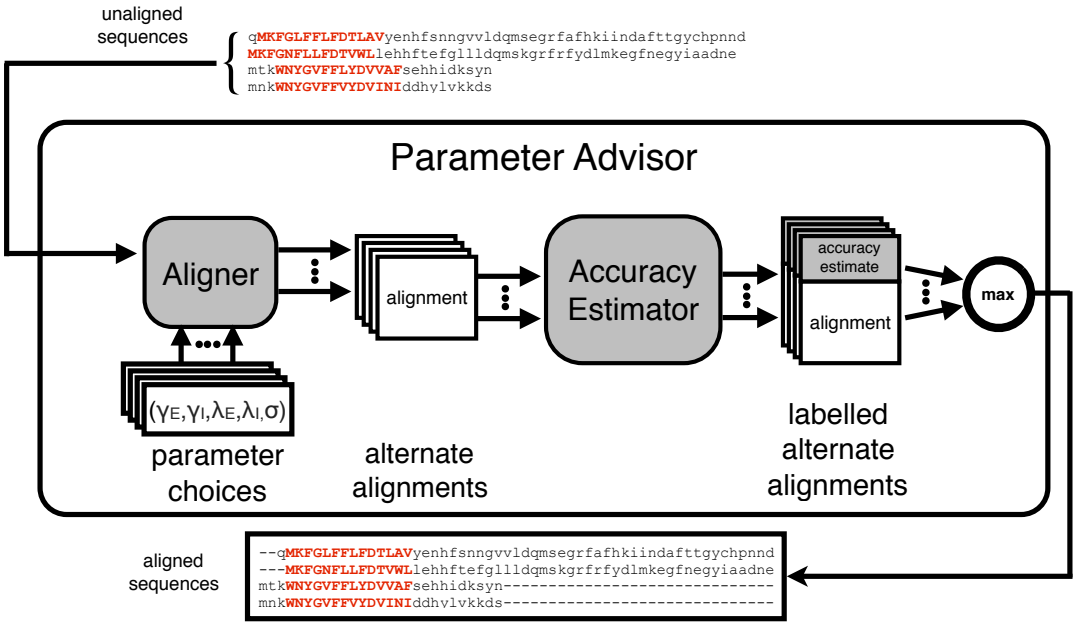


Figure 2: *The parameter advising process.* For an input set of sequences, a parameter advisor first invokes the aligner for each assignment of parameter values in a collection of parameter choices. Each parameter choice when used with the aligner produces an alternate alignment of the sequences. An accuracy estimator is then used to label each of the alternate alignments with an accuracy estimate. The advisor then returns the alignment with the highest accuracy estimate.

advisor tries each parameter choice $p \in P$, invokes an aligner to compute an alignment A_p using choice p, and then selects the parameter choice p^* that has maximum estimated accuracy: $p^* \in \text{argmax}_{p \in P} \{E(A_p)\}$. Figure 2 shows a diagram of parameter advising. Since the advisor runs the aligner $|P|$ times on a given set of input sequences, a crucial aspect of parameter advising is finding a small set P for which the true accuracy of the output alignment A_{p^*} is high.

To construct a good advisor, we need to find a good estimator E and a good set P. The estimator and advisor set are learned on training data consisting of benchmark sets of protein sequences for which a reference alignment is known. The learning procedure tries to find an estimator E and set P that maximize the true accuracy of the resulting advisor on this training data, which we subsequently assess on separate testing data.

Note that the process of advising is fast: for a set P of k parameter choices, advising involves computing k alignments under these choices, which can be done in parallel, evaluating the estimator on these k alignments, and taking a max. The separate process of training an advisor, by learning an estimator and advisor set as we review next, is done once, off-line, before any advising is done.

2.2 Learning an accuracy estimator

Our previous work (DeBlasio *et al.*, 2012; Kececioglu and DeBlasio, 2013) presented an efficient approach for learning an accuracy estimator that is a linear combination of real-valued alignment feature functions, based on solving a large-scale linear programming problem. This approach resulted in `Facet` (short for "feature-based accuracy estimator"; DeBlasio

and Kececioglu, 2015b), which is currently the most accurate estimator for parameter advising (Kececioglu and DeBlasio, 2013; DeBlasio and Kececioglu, 2017b).

This approach assumes we have a collection of d real-valued feature functions $g_1(A), \ldots, g_d(A)$ on alignments A, where these functions g_i are positively correlated with true accuracy. The alignment accuracy estimator E is a linear combination of these functions, $E(A) = \sum_{1 \leq i \leq d} c_i \, g_i(A)$, where the coefficents c_i specify the estimator E. When the feature functions have range $[0, 1]$ and the coefficients form a convex combination, the resulting estimator E will also have range $[0, 1]$. `Facet` uses a collection of five feature functions, many of which make use of predicted secondary structure for the protein sequences (Kececioglu and DeBlasio, 2013). Figure 3 shows the correlation of `Facet` and `TCS` (the next best estimator in our tests; Chang *et al.*, 2014) to true accuracy. To be able to distinguish good from bad alignments an estimator should have a steep slope and very little spread. While the `TCS` estimator has high slope, it has quite a bit of spread. In contrast, the `Facet` estimator has much less spread but a less steep slope, and we have found this to be more effective in ranking alignments for parameter advising.

The features we use in `Facet` are a mixture of canonical measures of alignment quality, such as Amino Acid Identity, and novel non-local features of an alignment that correlate with true accuracy. Many of the most accurate features use predicted protein secondary structure. For instance, the Secondary Structure Blockiness feature finds an optimal packing of blocks of aligned amino acids that have the same predicted structure type. The other feature functions used in the `Facet` estimator are: Secondary Structure Identity, Secondary Structure Agreement, Gap Open Density, and Core Column Percentage. A full description of all features is in Kececioglu and DeBlasio (2013).

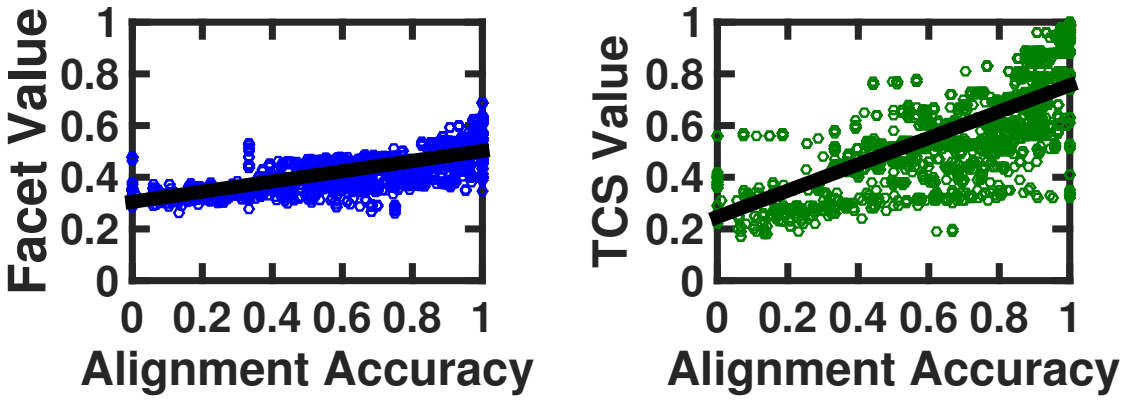


Figure 3: *Relationship of estimators to true accuracy.* Each point in a scatterplot corresponds to an alignment whose true accuracy is on the horizontal axis, and whose value under a given estimator is on the vertical axis. Both scatterplots show the same set of 3,000 alignments under the accuracy estimators `Facet` (Kececioglu and DeBlasio, 2013) and `TCS` (Chang *et al.*, 2014).

A parameter advisor uses the estimator to effectively rank alignments, so an estimator just needs to be monotonic in true accuracy. The *difference-fitting* approach learns the coefficients of an estimator that is close to monotonic by fitting the estimator to differences in true accuracy for pairs of training alignments. We can formulate the problem of coefficient finding using difference-fitting as a linear program; the details of this approach are in Kececioglu and DeBlasio (2013).

2.3 Learning an advisor set

The size of the parameter set used for advising should be small, since the aligner is run for each parameter setting. We utilize the concept of an oracle (a perfect advisor that has access to the true accuracy of an alignment; see Wheeler and Kececioglu, 2007) to

compute sets that we use in practice. For a given advisor set P, an *oracle* selects parameter choice $\text{argmax}_{p \in P}\{F(A_p)\}$, where again function F gives the true accuracy of an alignment. (Equivalently, an oracle is an advisor that uses the perfect estimator F.) An oracle always picks the parameter choice that yields the highest accuracy alignment.

While an oracle is impossible to construct in practice, it gives a theoretical limit on the accuracy achievable by advising with a given set. Furthermore, if we find the optimal advisor set for an oracle for a given cardinality bound k, which we call an *oracle set*, then the performance of an oracle on an oracle set gives a theoretical limit on how well advising can perform for a given bound k on the number of parameter choices. In practice, oracle sets are used with `Facet` to construct an advisor.

While finding an optimal oracle set is NP-complete, it can be formulated as an integer linear programming problem (Kececioglu and DeBlasio, 2013). Learning an optimal oracle set of cardinality k, for a universe of m parameter choices and a training set of n benchmarks, involves solving an integer linear program with $\Theta(mn)$ variables and $\Theta(mn)$ inequalities. Using the `CPLEX` (`IBM` Corporation, 2015) integer linear programming solver, this formulation permits finding optimal oracle sets in practice even for cardinalities up to $k = 25$.

It is possible to use a greedy procedure to find advisor sets tuned to a concrete estimator rather than the oracle (DeBlasio and Kececioglu, 2017b). While using these sets on global parameter advising increased advising accuracy over oracle sets, this increase did not transfer to adaptive local realignment. For the results in later sections, we will construct an advisor using the `Facet` accuracy estimator learned using difference fitting, along with oracle sets. Note this is *not* an oracle advisor, since it uses the `Facet` estimator.

3 Adaptive local realignment

To handle heterogeneity in protein sequences that have regions requiring distinct alignment parameter settings, we introduce a method that we call *adaptive local realignment*. Adaptive local realignment uses some of the same basic principles that have been shown to work well for global parameter advising. We apply the techniques described in the previous section locally to choose the best alignment parameters over an interval of columns in an alignment.

The adaptive local realignment method has two major steps: (1) discerning regions of the alignment that are well-aligned, which should be retained; and (2) producing a new alignment for regions that are poorly aligned, using parameter advising.

3.1 Identifying local realignment regions

When selecting alignment columns that should be retained, we cannot just identify the correctly-recovered columns in a computed alignment, since in practice we do not have a known reference alignment against which to compare. We can, however, attempt to identify these regions using an accuracy estimator E, as defined earlier. To partition the input alignment, we first evaluate the accuracy estimator within a window of a fixed width that we slide across the alignment (Figure 4a). The window width is given by a fraction $\omega < 1$ of the total length ℓ of the alignment. The value of ω must be carefully chosen, as the accuracy estimator has features that reflect global properties of an alignment. While a larger sliding window provides more context at each position, and should yield a better accuracy estimate, if the window is too large, the granularity may not be sufficiently fine to precisely identify the transition points between correctly- and incorrectly-aligned columns. Hence we also specify

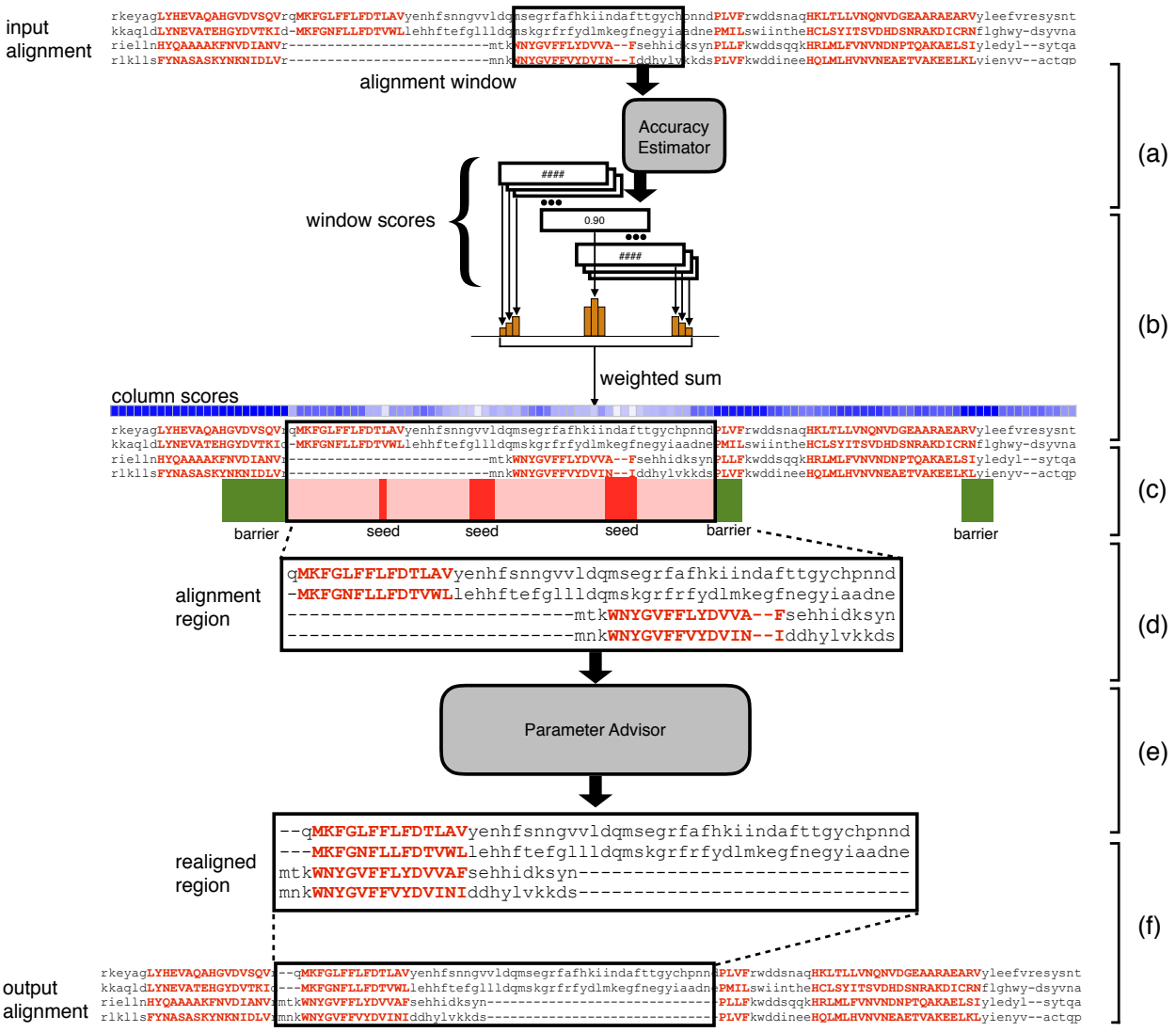


Figure 4: *The adaptive local realignment process.* (a) Estimate the accuracy for sliding windows across the input alignment using `Facet`. (b) Calculate a score for each column as the weighted sum of the accuracy estimates of all windows that overlap the column. (c) Label columns that are above τ_S or below τ_B as seeds or barriers, respectively. (d) Define realignment regions that will be extracted from the alignment by extending seeds in both directions until they reach a barrier. (e) Use parameter advising to find a new alignment of each realignment region. (f) Replace the original realignment region if the new alignment has higher accuracy estimate.

upper and lower bounds on the absolute window width to account for very short and very long alignments.

A score is assigned to each column as a weighted sum of the estimated accuracies of windows that overlap that column, weighted according to how far away the center column of the window is (Figure 4b). A geometric distribution with decay rate $\lambda < 1$, centered at the given alignment column, weights the contributions to the column's score by windows that overlap it. As λ approaches 1, a column gets equal weight from all overlapping windows; as λ approaches 0, the score only depends on the window centered at that column.

Using these column scores, we partition the alignment into regions by first labeling the columns for which there is the most evidence of being correctly—or incorrectly—aligned. Given a minimum percentage of columns we would like to retain from the input alignment, ρ_B (for "barriers"), and a minimum percentage of columns we would like to replace, ρ_S (for "seeds"), we determine two threshold values, τ_B and τ_S, such that the number of columns with score greater than τ_B is at least $\lceil \rho_B \ \ell \rceil$ for an alignment of length ℓ, and the number of columns with score less than τ_S is at least $\lceil \rho_S \ \ell \rceil$. Then all columns with score at least τ_B are labeled *barriers*—these columns are guaranteed to be retained—and those with column score at most τ_S are labeled *seeds*—these columns will be realigned (Figure 4c). Finally, we define realignment regions by extending each seed in both directions until a barrier column (or the first or last column of the alignment) is reached. Note that a realignment region may contain more than one seed column, but will never include one of the barriers. With this method we ensure: at least $\rho_B \ \ell$ columns from the original alignment will be in the final alignment, there will always be at least one realignment region, and there will never be a realignment region that covers all columns of the input.

3.2 Local parameter advising on a region

The realignment regions defined above identify subalignments that may potentially be improved, and we use parameter advising to search for better alignments of these regions that may replace them in the initial alignment. We extract the subalignment given by the columns in each realignment region (Figure 4d). Removing the gaps from this subalignment yields a set of unaligned sequences, which become the input to a slightly modified version of the parameter advising approach described earlier (Section 2.1, Figure 2), that now takes the location of the realignment region into account in the alignment scoring scheme (Figure 4e). The `Opal` aligner uses different scores for terminal and internal gaps. For adaptive local realignment, we only apply terminal gap scores when the terminal column in the context of the subalignment is also a terminal column in the context of the global alignment. As indicated earlier, an alignment region will not have terminal columns at both ends.

Once we have obtained the new subalignment via parameter advising, the last step is to replace the corresponding region in the original alignment (Figure 4f) if its new `Facet` score is higher than that of the original subalignment for the realignment region.

After all realignment regions have been updated by local advising, we make one final comparison between the new resulting alignment and the original initial alignment: of these two global alignments, the one with higher estimated accuracy is returned.

3.3 Iterative local realignment

While adaptive local realignment can correct errors, after performing the procedure there may still be regions that can be further improved. Such regions may not have been identified

because they are subregions of a newly-included alignment, or because the threshold was too low for a seed to be identified due to the very low quality of other another region. In either situation, it can be beneficial to repeat adaptive local realignment to further increase accuracy. Accordingly, we iterate the whole process (Figure 4) until a user-defined maximum number of iterations is reached, or no further improvements are made.

3.4 Combining local and global advising

The quality of the initial alignment that is input to this process is critical, since realignment only makes local improvements. To identify the best initial alignment, it may be advantageous to use global parameter advising, which is known to aid accuracy (Kececioglu and DeBlasio, 2013). Local and global parameter advising can be combined in two ways:

(1) local advising on *all* global alignments, using adaptive local realignment on each of the alternate alignments considered by global parameter advising with advisor set P, and then choosing among all $2|P|$ alternate alignments (for $|P|$ original global alignments, and $|P|$ locally-advised realignments); and

(2) local advising on the *best* global alignment, which first selects the global alignment of highest estimated accuracy, and then only on this selection, performs adaptive local realignment.

We compare both these ways of combining local and global advising, as well as simply local advising on the single alignment produced by the default parameter setting, in the next section.

4 Assessing adaptive local realignment

We evaluate the performance of adaptive local realignment, and its use in combination with global advising, through experiments on a collection of protein multiple sequence alignment benchmarks. A full description of the benchmarks and the universe of parameter settings used for parameter advising can be found in Kececioglu and DeBlasio (2013), and is briefly described here.

The benchmark suites used in our experiments consist of reference alignments of proteins that are largely induced by structurally aligning their known three-dimensional structures. In particular, we use the `BENCH` suite of Edgar (2009) supplemented by a selection from the `PALI` suite of Balaji *et al.* (2001). `BENCH` is in turn is a combination of the `BAliBASE` (Bahr *et al.*, 2001), `PREFAB` (Edgar, 2004), `OxBench` (Raghava *et al.*, 2003), and `SABRE` (Van Walle *et al.*, 2005) suites of benchmarks. The full collection of benchmarks we use consists of 861 reference alignments.

As is common in benchmark suites, easy-to-align benchmarks are highly over-represented in this collection. To correct for this bias towards easy-to-align benchmarks when evaluating average advising accuracy, we binned the 861 benchmarks by *difficulty*, which we measured by the true accuracy of `Opal` using its default parameter setting. We then divided the the full range $[0, 1]$ of accuracies into 10 bins, where bin b for $b = 1, ..., 10$ contains difficulty interval $((b-1)/10, b/10]$, which have 12, 12, 20, 34, 26, 50, 62, 74, 137, and 434 benchmarks, respectively. We report "average accuracy" uniformly-averaged across *bins* (rather than uniformly-averaged across benchmarks). This means that the average alignment accuracy of `Opal` using its default parameter setting will be near 50%. Even though the binning is based

on `Opal` default alignments, most other standard aligners have default accuracy near 50% as well: `Clustal Omega` (Sievers *et al.*, 2011), 47.3%; `MUSCLE` (Edgar, 2004), 48.4%; and `MAFFT` (Katoh *et al.*, 2005), 51.0%. (Note these numbers do not imply that `MAFFT`, for instance, is more accurate than `Clustal Omega`, since if you bin based on an aligner other than `Opal`, you will again get a different ranking of these aligners.) We have previously shown that for the task of global parameter advising, many of the top aligners perform almost equally well; we chose `Opal` for local parameter advising as it had the highest global advising accuracy in prior tests (DeBlasio and Kececioglu, 2017b). We emphasize that the methodology presented here is general, and can be applied to any other aligner.

We developed a universe of alignment parameter settings by enumerating reasonable values for each of the tunable parameters for the `Opal` aligner. In particular, the tunable parameters for `Opal` can be written as a 5-tuple $(\sigma, \gamma_I, \gamma_T, \lambda_I, \lambda_T)$, which represents the substitution scoring matrix (σ), as well as the the internal (I) and terminal (T) gap-open (γ) and gap-extension (λ) penalties. We considered three choices of substitution matrices from the `BLOSUM` (Henikoff and Henikoff, 1992) and `VTML` (Müller *et al.*, 2002) families, two choices of terminal gap-extension penalties, and three choices each of internal gap-extension, terminal gap-open, and internal gap-open penalties. In total, we generated a universe of 162 parameter settings.

We used 12-fold *cross validation* to assess the increase in accuracy gained by adaptive local realignment. We first evenly and randomly distributed benchmarks into twelve groups for each hardness bin; the twelve independent folds were generated by choosing one group from each bin to be in the *testing set*, and the other eleven to be in the *training set*. Finally, we generated an example alignment for each benchmark in the training or testing set for

Table 1: Adaptive Local Realignment Hyper-Parameter Selection

Hyper-parameter	Range of values	Selection
window-width fraction ω	$0.05, 0.1, 0.2, 0.3, \ldots, 0.7$	0.3
window-width lower-bound	$5, 10, 20, 30$	10
window-width upper-bound	$30, 50, 75, 100, 125$	30
barrier percentage ρ_B	$5\%, 10\%, 20\%, 30\%, \ldots, 70\%$	10%
seed percentage ρ_S	$5\%, 10\%, 20\%, 30\%, \ldots, 70\%$	30%
geometric decay rate λ	$0.5, 0.66, 0.9, 0.99$	0.9
number of iterations	$1, \ldots, 5, 10, 15, 25$	5

each fold using each of the parameters in our universe with the `Opal` aligner. The results we report are averages over these twelve folds. (Note that across the twelve folds, every example alignment is tested on exactly once.)

We trained the estimator coefficients for `Facet` on the training example sets for each fold, using the difference-fitting method described in Section 2.2. We found there was very little change in coefficients between the training folds, so for simplicity we use the estimator coefficients that are released with the latest version of `Facet`, which were trained on all available benchmarks. We also consider the `TCS` estimator for adaptive local realignment, and show these results in Section 4.3.

To choose values for the hyper-parameters for adaptive local realignment (such as ω, ρ_B, and ρ_S, as described earlier in Section 3.1), we enumerated the cross product of reasonable values for these parameters. We used the performance on *training* benchmarks described

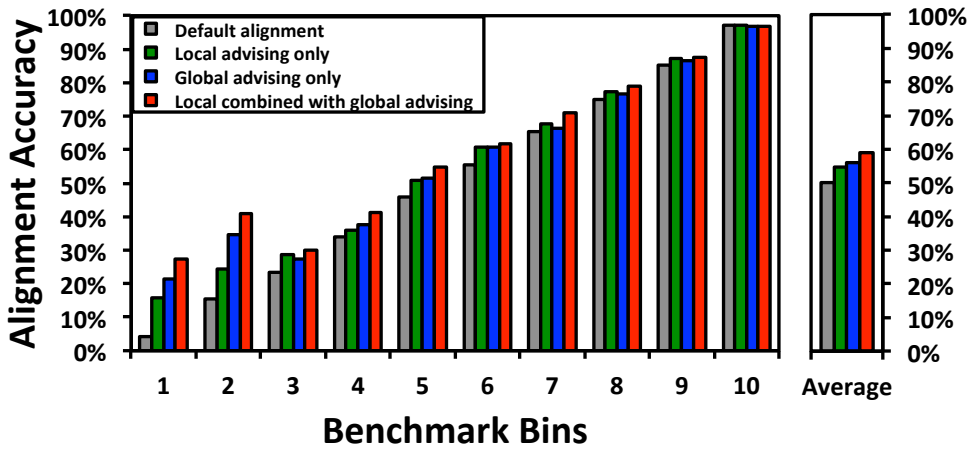


Figure 5: *Accuracy of the default alignment, and different advising methods, within difficulty bins.* The horizontal axis shows all ten benchmarks bins. The vertical axis shows the accuracy averaged over just the benchmarks in that bin using default parameter settings, local advising only, global advising only, and the combined advising method using an oracle set of cardinality $k = 10$. The bar chart on the right shows the accuracy uniformly averaged over the bins.

above to find the combination of these settings that gave the highest improvement in accuracy when local advising was applied to the default alignments from `Opal`. Table 1 summarizes these hyper-parameters, the range of values that we considered, and the value that was selected for use in our experiments. Details on selecting the number of iterations are given later in Section 4.4.

4.1 Effect of local realignment within difficulty bins

Figure 5 shows alignment accuracy within difficulty bins for default alignments from `Opal`, local advising on these default alignments, global advising alone, and local combined with

global advising (which uses local advising on all alternate alignments considered by global advising). The optimal oracle set of cardinality $k = 10$ was used for both local and global advising.

The improvement gained by adaptive local realignment over the default parameter setting is most evident in the two most difficult benchmark bins. On these hardest bins, local advising increases average accuracy by 11.5% and 9.1%, respectively. Furthermore, local advising boosts accuracy in all ten bins. On average, local advising increases the accuracy of the default alignments by 4.5% across the bins.

Combining local and global advising together substantially improves accuracy over using either of these approaches alone. This is again most pronounced on the hardest-to-align benchmarks. On the bottom two bins, combined local and global advising increases the accuracy by 23.0% and 25.6% over the default parameter choice. On these bottom-most bins, local advising increases the accuracy by 5.9% and 6.4% over global advising alone. On average across bins, combined local and global advising increases accuracy by 8.9% over the default parameter choice, and local advising by 3.1% over global advising alone.

4.2 Varying advisor set cardinality

Since a new alignment is recomputed for each realignment region and each parameter choice in the advisor set, the running time grows with the cardinality of the advisor set, so it may be desirable to use a smaller cardinality to reduce the running time for advising. We constructed optimal oracle advisor sets for cardinalities $k = 2, \ldots, 15$, and examined their effect on local advising both alone and in combination with global advising. Figure 6 shows

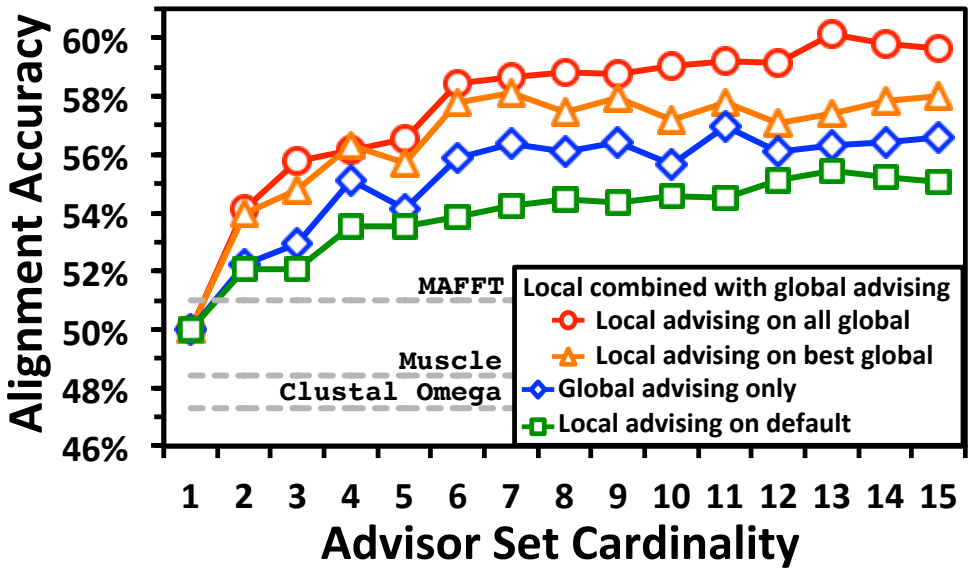


Figure 6: *Advising accuracy versus advisor set cardinality.* The horizontal axis is the cardinality of the advisor set used by the advising methods. The vertical axis shows the advising accuracy of the default parameter setting, local advising, global advising, and the combined advising method, averaged across difficulty bins.

average advising accuracies with advisor sets of increasing cardinalities, for local advising on `Opal` default alignments, global advising alone, and local combined with global advising. The figure shows accuracies for both of the strategies described in Section 3.4 for combining local and global advising: local advising applied to the best alignment from global advising, and local advising applied to all alignments from global advising. The cardinality of the advisor set used for both global and local advising is on the horizontal axis, while alignment accuracy uniformly averaged across bins is on the vertical axis.

The average accuracy for all four approaches eventually reaches a plateau, where adding further parameter choices to the advisor set no longer improves accuracy. This plateau is reached at cardinality $k=10$ for local advising applied to default alignments, and at $k=6$ for global advising with or without local advising, but plateaus at a higher accuracy for combined advising. Across all cardinalities, combined local and global advising improves accuracy by nearly 4% on average. Note that when local advising is applied to all alignments from global advising, the combined advisor is now choosing from an expanded set of alternate alignments whose best accuracy can only be higher than the original set.

These results above again give advising accuracy uniformly-averaged across *bins*. In contrast, if we report advising accuracy uniformly-averaged across *benchmarks*, `Opal` with its default parameter setting achieves accuracy 80.4%; local or global advising alone increases this accuracy to 82.1% and 81.8%, respectively; and combining both methods increases the accuracy to 83.1% (all at cardinality $k=10$). By comparison, the corresponding average accuracies of other standard aligners, using their default parameter settings, are: `Clustal Omega`, 77.3%; `MUSCLE`, 78.1%; and `MAFFT`, 79.4%.

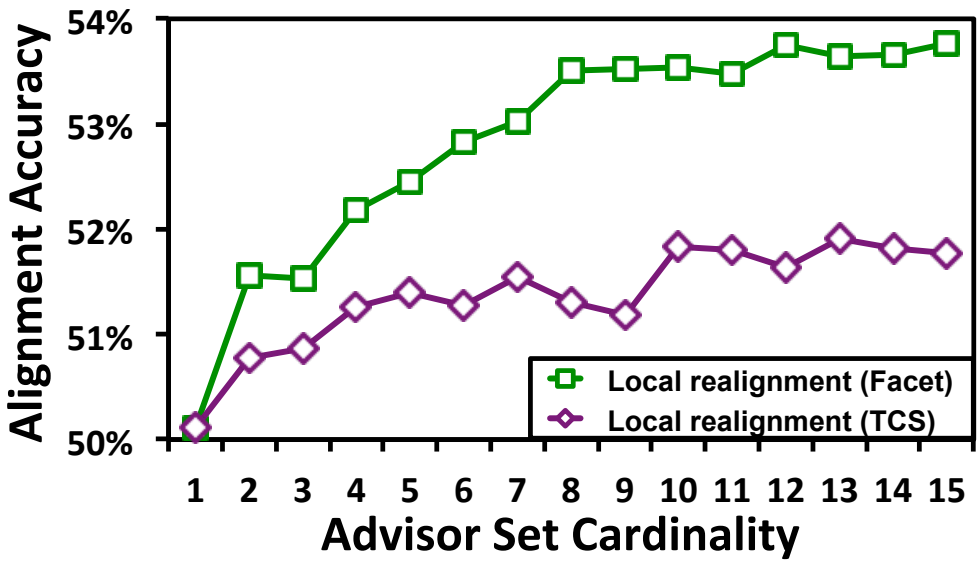


Figure 7: *Accuracy of local realignment using the* `Facet` *and* `TCS` *estimators, for varying advisor set cardinality.* The curves show the accuracy of local realignment, using either the `Facet` or `TCS` estimators, on initial alignments produced by `Opal` with its default parameter setting. The horizontal axis is the cardinality of the advisor set for local realignment, where oracle sets are used for advising. The vertical axis shows alignment accuracy, averaged across difficulty bins.

4.3 Comparing estimators for local advising

Figure 7 shows the average accuracy of local advising on default alignments using both `Facet` and `TCS` (the next-best estimator for advising; see Kececioglu and DeBlasio (2013) and DeBlasio and Kececioglu (2017b)). These results used only a single iteration of adaptive local realignment for both estimators, due to the large increase in running time caused by calls to the external `TCS` program. Using `TCS` for local advising does increase accuracy over the default alignment, but the increase is less than half that of `Facet`.

Table 2: Accuracy of Adaptive Local Realignment Across Iterations

Iteration	1	2	3	4	5	10	15	25
Training	53.5%	53.9%	54.5%	54.6%	54.8%	54.8%	54.9%	54.9%
Testing	53.5%	53.7%	54.1%	54.4%	54.5%	54.5%	54.5%	54.5%

Table 3: Summary of Adaptive Local Realignment on Default Alignments

Bin	1	2	3	4	5	6	7	8	9	10	Overall
Number of benchmarks	12	12	20	34	26	50	61	74	137	434	861
Number modified	8	7	16	27	19	34	46	61	115	352	685
Percentage modified	67%	58%	80%	79%	73%	68%	74%	82%	84%	81%	80%
Regions per benchmark	1.92	2.17	2.50	1.88	2.23	2.14	2.31	2.16	2.48	2.19	2.23
Columns realigned	75%	73%	76%	70%	75%	77%	74%	73%	75%	72%	73%
Columns replaced	64%	60%	68%	60%	66%	72%	65%	63%	64%	47%	57%

4.4 Effect of iterating local realignment

As discussed in Section 3.3, iterating adaptive local realignment should eventually reach a state where the alignment no longer improves, or even worse, begins deteriorating due to noise in the accuracy estimator. Table 2 shows the average accuracy of adaptive local realignment on default alignments as the number of iterations increases. Both the training and testing accuracy reach a plateau at around 5 iterations, and this is the number we used in Sections 4.1 and 4.2.

4.5 Summarizing the effect of adaptive local realignment

Table 3 summarizes how adaptive local realignment behaves across difficulty bins, during the first iteration of improving `Opal` default alignments. The columns are average values for each of the 10 benchmark bins, and average values across all benchmarks. The first three rows show how many of the 861 benchmarks are in each bin, as well as the number and percentage of those with at least one realignment region that was replaced. The last three rows summarize how much of each alignment changed. The fourth row shows the average number of realignment regions found for each benchmark; on average about 2 regions were realigned for each default alignment. The last two rows summarize the percentage of the original columns that were in realignment regions, and how many of the columns from the original alignment were replaced. Note that while the percentage of columns covered by realignment regions stays roughly the same across bins, in the easiest-to-align bin only 47% of alignment columns were replaced, while in the rest of the bins over 60% of the columns changed.

4.6 Running time

To give a sense of running time, `Opal` with adaptive local realignment, averaged across all benchmarks, takes 110 seconds using an advisor set of cardinality 10 and 5 iterations. This is up from 36 seconds for 1 iteration, and about 8 seconds using just the default parameter setting. This high increase in wall-clock time is mainly due to adaptive local realignment, as currently implemented, not exploiting parallelism in advising. In contrast, *global* advising is parallelized, and the average running time of global advising for the same advisor set of

cardinality 10 is only around 33 seconds. (Note that the number of columns being repeatedly aligned by global advising is about a factor 1.25 more than for local advising.) When these two approaches are combined, the average running time increases to 68 and 178 seconds, respectively, for local advising on the best alignment, and on all alignments, from global advising.

5 Conclusion

We have presented *adaptive local realignment*, to our knowledge the first approach that demonstrably boosts protein multiple sequence alignment accuracy by adaptively realigning regions with local parameter settings. Applying this new method to alignments initially computed using an optimal default parameter setting already improves accuracy significantly, and when combined with global parameter advising to select an initial parameter setting, this new approach to local parameter advising boosts alignment accuracy greatly.

A new tool that performs both adaptive local realignment and global parameter advising using the `Opal` aligner is available at `facet.cs.arizona.edu`.

Further research

Many directions remain open for further research in local parameter advising. In the context of global parameter advising, *greedy advisor sets*, which are designed to work well with a given accuracy estimator, are known to perform better than estimator-independent oracle sets (DeBlasio and Kececioglu, 2017b). In the context of local parameter advising, however, the greedy sets found for global advising performed worse than oracle sets for local

advising (hence the use of oracle sets in this study). Greedy sets may possibly have underperformed due to the smaller universe of parameter settings explored here, or because the known tendency of greedy sets to not generalize well may possibly be exacerbated when they are applied to local advising. While improving the generalization of greedy sets may require more fundamental changes to our approach for learning advisor sets, perhaps by simply exploring a much larger universe of parameter settings and by learning greedy sets specifically for local parameter advising, advisor sets might be found that perform even better than the oracle sets used here.

Finally, combining local parameter advising with *aligner advising* (DeBlasio and Kececioglu, 2015a), which takes in addition a set of aligners and selects both an aligner and its parameter setting—effectively yielding a method for *local ensemble alignment*—also seems promising. Just as a given alignment tool may be particularly well-suited for aligning a class of proteins with high accuracy (through the developer tailoring the aligner's models and methods), so an aligner might for instance have a class of protein structural motifs at which it excels. Incorporating aligner advising could potentially leverage each aligner's particular strength within adaptive local realignment.

Acknowledgements

Research of JK and DD at the University of Arizona was supported by NSF Grant IIS-1217886 to JK. DD was also partially supported at Carnegie Mellon University by NSF Grant CCF-1256087, NSF Grant CCF-131999, NIH Grant R01HG007104, and Gordon and Betty Moore Foundation Grant GBMF4554, to Carl Kingsford.

This paper is an extended version of the conference publication, DeBlasio and Kececioglu (2017a).

Author Disclosure Statement

No competing financial interests exist.

References

Anson, E. L. and Myers, E. W. 1997. ReAligner: a program for refining DNA sequence multi-alignments. *Journal of Computational Biology*, **4**(3), 369–383.

Bahr, A., Thompson, J. D., Thierry, J. C., *et al.* 2001. `BAliBASE` (Benchmark alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Research*, **29**(1), 323–326.

Balaji, S., Sujatha, S., Kumar, S., *et al.* 2001. `PALI`—a database of phylogeny and ALIgnment of homologous protein structures. *Nucleic Acids Research*, **29**(1), 61–65.

Chang, J., Tommaso, P., and Notredame, C. 2014. `TCS`: A new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. *Molecular Biology and Evolution*, **31**(6), 1625–1637.

DeBlasio, D. and Kececioglu, J. 2015a. Ensemble multiple sequence alignment via advising. In the proceedings of *6th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM-BCB)*, pages 452–461.

DeBlasio, D. and Kececioglu, J. 2015b. `Facet`: software for accuracy estimation of protein multiple sequence alignments. `http://facet.cs.arizona.edu`.

DeBlasio, D. and Kececioglu, J. 2017a. Boosting alignment accuracy by adaptive local realignment. In the proceedings of *21st Conference on Research in Computational Molecular Biology (RECOMB)*, pages 1–17.

DeBlasio, D. and Kececioglu, J. 2017b. Learning parameter-advising sets for multiple sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, **14**(5), 1028 – 1041.

DeBlasio, D. and Kececioglu, J. 2017c. *Parameter Advising for Multiple Sequence Alignment*, volume 26 of Computational Biology Series. Springer International, Cham, Switzerland.

DeBlasio, D. F., Wheeler, T. J., and Kececioglu, J. D. 2012. Estimating the accuracy of multiple alignments and its use in parameter advising. In the proceedings of *16th Conference on Research in Computational Molecular Biology (RECOMB)*, pages 45–59.

Do, C., Mahabhashyam, M., Brudno, M., *et al.* 2005. `ProbCons`: probabilistic consistency-based multiple sequence alignment. *Genome Research*, **15**(2), 330–340.

Edgar, R. 2004. `MUSCLE`: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, **32**(5), 1792–1797.

Edgar, R. C. 2009. `BENCH`. `http://drive5.com/bench`.

Fitch, W. M. and Margoliash, E. 1967. A method for estimating the number of invariant

amino acid coding positions in a gene using cytochrome c as a model case. *Biochemical Genetics*, **1**(1), 65–71.

Gotoh, O. 1993. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Computer Applications in the Biosciences*, **9**(3), 361–370.

Henikoff, S. and Henikoff, J. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the USA*, **89**(22), 10915–10919.

Katoh, K., Kuma, K.-i., Toh, H., *et al.* 2005. `MAFFT` version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research*, **33**(2), 511–518.

Kececioglu, J. and DeBlasio, D. 2013. Accuracy estimation and parameter advising for protein multiple sequence alignment. *Journal of Computational Biology*, **20**(4), 259–279.

Kececioglu, J. and Starrett, D. 2004. Aligning alignments exactly. In the proceedings of *8th Conference on Research in Computational Molecular Biology (RECOMB)*, pages 85–96.

Löytynoja, A. and Goldman, N. 2008. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, **320**(5883), 1632–1635.

Müller, T., Spang, R., and Vingron, M. 2002. Estimating amino acid substitution models: a comparison of dayhoff's estimator, the resolvent approach and a maximum likelihood method. *Molecular Biology and Evolution*, **19**(1), 8–13.

Notredame, C., Higgins, D., and Heringa, J. 2000. `T-Coffee`: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, **302**(1), 205–217.

Raghava, G., Searle, S. M., Audley, P. C., *et al.* 2003. `OXBench`: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, **4**(1), 1–23.

Roskin, K. M., Paten, B., and Haussler, D. 2011. Meta-alignment with `crumble` and `prune`: partitioning very large alignment problems for performance and parallelization. *BMC Bioinformatics*, **12**(1), 1–12.

Sievers, F., Wilm, A., Dineen, D., *et al.* 2011. Fast, scalable generation of high-quality protein multiple sequence alignments using `Clustal omega`. *Molecular Systems Biology*, **7**(1), 539–539.

`IBM` Corporation 2015. `CPLEX`: high-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming (version 12.6.2.0). `http://www.ilog.com/products/cplex`.

Thompson, J., Higgins, D., and Gibson, T. 1994. `ClustalW`: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, **22**(22), 4673–4680.

Van Walle, I., Lasters, I., and Wyns, L. 2005. `SABmark`: a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, **21**(7), 1267–1268.

Wallace, I. M., O'Sullivan, O., Higgins, D. G., *et al.* 2006. `M-Coffee`: combining multiple sequence alignment methods with `T-Coffee`. *Nucleic Acids Research*, **34**(6), 1692–1699.

Wheeler, T. J. and Kececioglu, J. D. 2007. Multiple alignment by aligning alignments. *Bioinformatics*, **23**(13), i559–68.

Yang, Z. 1993. Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. *Molecular Biology and Evolution*, **10**(6), 1396–1401.