Boosting alignment accuracy by adaptive local realignment

Dan DeBlasio[1⋆] and John Kececioglu[2]

[1] Computational Biology Department, Carnegie Mellon University, USA
deblasio@cmu.edu
[2] Department of Computer Science, The University of Arizona, USA
kece@cs.arizona.edu

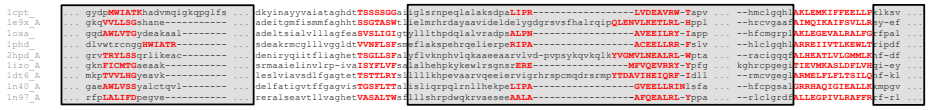
Abstract. While mutation rates can vary markedly over the residues of a protein, multiple sequence alignment tools typically use the same values for their scoring-function parameters across a protein's entire length. We present a new approach, called *adaptive local realignment*, that in contrast automatically 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 aligners, to 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`.

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 [11]. 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 called "hypermutable regions"). This has led to methods in phylogeny construction that take variable mutation rates into account when building trees from sequences [26]. 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

⋆ Corresponding author; work performed while at the University of Arizona.



(a) Using the optimal default parameter setting

(b) After adaptive local realignment

Fig. 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.

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 [25]. The bottom alignment is obtained by our new method, taking the top alignment, automatically identifing the sections 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 into two categories, based on the orientation of their partitions. *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` [21] is a pair of algorithms for performing both vertical (`Crumble`) and horizontal (`Prune`) splits on an input set of sequences. 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, making aligning a large number of long sequences feasible.

Gotoh [12] 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 using it to realign the groups.

`AlignAlign` [16] 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 [25].

The standard aligners `MUSCLE` [10], `MAFFT` [14], and `ProbCons` [8] also include a polishing step that performs horizontal realignment similar to Gotoh.

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` [17] uses a multi-level HMM that effectively chooses the alignment scoring function at each position. `T-Coffee` [19] uses consistency between pairwise alignments to create position-specific substitution scores. In fact, even the early tool `ClustalW` [23] 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 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 accuracy.

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 [7].)

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.

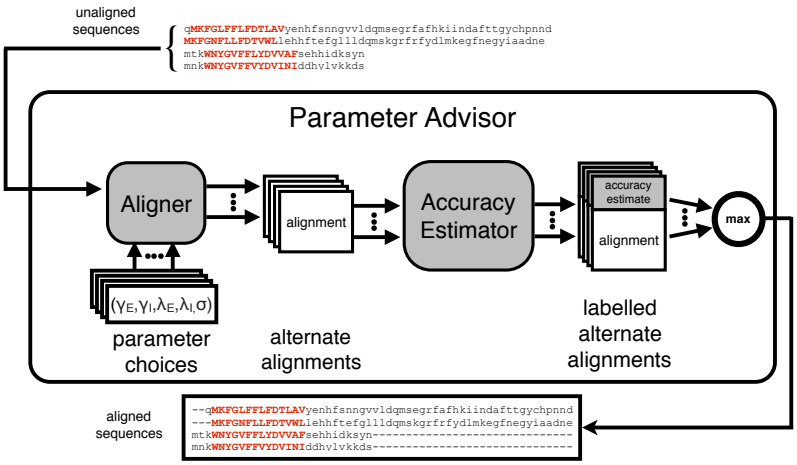


Fig. 2. *The parameter advising process.* For an input set of sequences, a parameter advisor first invokes the aligner for each assignment of parameter values in a collection of parameter choices. Each parameter choice when used with the aligner produces an alternate alignment of the sequences. An accuracy 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.

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 a different, 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. (So a computed alignment of 100% accuracy completely agrees with the reference on its core blocks, though 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 our 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 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 [6, 15] 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" [4]), which is currently the most accurate estimator for parameter advising [15, 5].

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, $E(A) = \sum_{1 \le i \le d} c_i \, g_i(A)$, 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

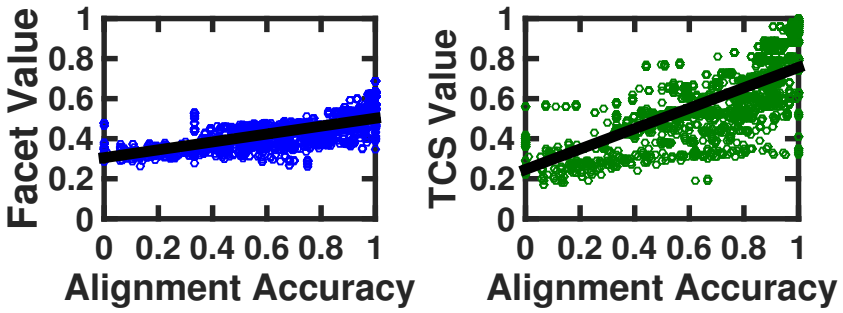


Fig. 3. *Relationship of estimators to true accuracy.* Each point in a scatterplot corresponds to an alignment whose true accuracy is on the horizontal axis, and whose value under a given estimator is on the vertical axis. Both scatterplots show the same set of 3,000 alignments under the accuracy estimators `Facet` [15] and `TCS` [3].

make use of predicted secondary structure for the protein sequences [15]. Figure 3 shows the correlation of `Facet` and `TCS` [3] (the next best estimator in our tests) 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 [15].

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 [15].

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 [25] (a perfect advisor that has access to the true accuracy of an alignment) to find 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.

We have shown that while finding an optimal oracle set is NP-complete, it can be formulated as an integer linear programming problem [15]. Learning an optimal oracle set of cardinality k, for a universe of u parameter choices and a training set of t benchmarks, involves solving an integer linear program with $\Theta(ut)$ variables and $\Theta(ut)$ inequalities. Using the `CPLEX` 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 [5]. 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.

This prior work focused on using parameter advising to choose the parameter setting for an entire alignment, which we call here *global* parameter advising. The next section presents adaptive local realignment, which leverages these ideas to in essence achieve *local* parameter advising.

3 Adaptive local realignment

To overcome the issue of protein sequences being non-homogeneous and having regions that may require different alignment parameter settings we have developed a method 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 can be broken down into two 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.

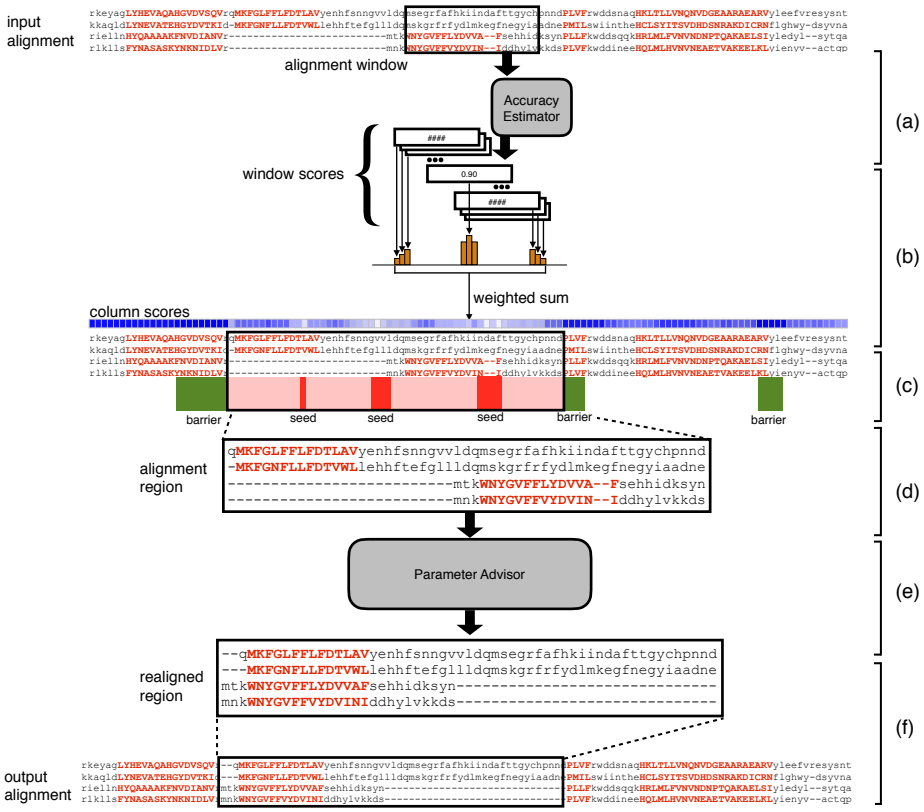


Fig. 4. *The adaptive local realignment process.* (a) Estimate the accuracy for sliding windows across the input alignment using `Facet`. (b) Calculate a score for each column as the weighted sum of the accuracies of all windows that overlap the column. (c) Label columns that are above τ_S or below τ_B 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 is more accurate.

3.1 Identifying local realignment regions

When selecting alignment columns that should be saved we cannot simply identify correctly recovered columns in a computed alignment since just as with global alignments we do not have a known reference in practice. But we can identify these regions using an accuracy estimator E which we defined earlier. To partition the input alignment we first calculate the estimated accuracy of a sliding window across the alignment (Figure 4a). The window size is a fraction $w \leq 1$ of the total length of the alignment. The value of w must be chosen carefully because the accuracy estimator has features that reflect global proper-

ties of an alignment. A larger sliding window will provide more context at each position and should provide a better estimate of accuracy. At the same time, if the window is too large there will not be fine enough granularity to identify the transition points between correctly- and incorrectly-aligned columns. Additionally, we define upper and lower bounds on the absolute window size to account for very short and very long alignments.

A score is assigned to each column as the sum of the approximately $\frac{1}{w}$ window scores that overlap that column weighted proportionally to the distance to the center column of the window (Figure 4b). A geometric distribution, with a decay rate $d < 1$, centered on the middle column is used to determine the contribution of a window to the scores of each of the columns it covers. As d approaches 1, a column gets equal weight from all covering windows; as it approaches 0, the score is dependent only on the window centered at that column.

From the column scores we generate a partitioning by labeling columns for which there is the most evidence of being correctly (or incorrectly) aligned. Given the minimum percentage of columns we would like to retain from the input alignment, T_B, and the minimum percentage of columns we would like to replace, T_S, we calculate two threshold values τ_B and τ_S such that the number of columns with score greater than τ_B is at least $\lceil \ell\, T_B \rceil$, and the number of columns with score less than τ_S is at least $\lceil \ell\, T_B \rceil$. We can then label all columns with score at least τ_B as *barriers*—these columns are guaranteed to be retained—and those with column score at most τ_S are labeled as *seeds*—these columns are guaranteed to be realigned (Figure 4c). Finally, we define realignment regions by extending each seed in both directions until a barrier column is reached (or the first or last column of the alignment). Note that a realignment region may contain more than one seed column but will never include one of the barriers. Using this method we ensure that: at least ℓT_B 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 have the potential to be improved and we will use parameter advising to produce better alignments of these regions and replace them in the input alignment. We extract the subalignment from the input identified by the columns in each alignment region (Figure 4d). Removing the gaps from this subalignment yields a set of unaligned sequences which becomes the input to a slightly modified version of the parameter advising method described earlier (Section 2.1, Figure 2) which considers the location of the realignment region within the alignment scoring scheme (Figure 4e). The `Opal` aligner scores terminal and internal gaps separately but for the case of adaptive local realignment we only apply terminal gap scores when the terminal column in the context of the subalignment is also the terminal column in the context of the global alignment. As mentioned earlier an alignment region will never include both terminals.

Once we have obtained the new alignment via parameter advising the final step is to replace the original region in the input (Figure 4f) if the `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 last advising decision between the new alignment and the input alignment. The more accurate global alignment of the two is returned.

3.3 Iterative local realignment

The adaptive local realignment process corrects misalignments in the input, but after performing the procedure there may still be some regions of the alignment that can be improved. These remaining regions may not have been identified originally because they are subregions within 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 case, it would be beneficial to repeat adaptive local realignment to further increase accuracy. Therefore, 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 alignment input to the adaptive local realignment process is critical since adaptive local realignment is only making local improvements. Therefore, we would like to use global parameter advising, which has been shown to improve accuracy [15], to identify the best initial alignment. Local and global advising can then be combined in two ways:

(1) local advising on *all* global alignments, using adaptive local realignment on each of the alternate alignments produced within global parameter advising, and then choosing among all $2|P|$ alternate alignments (for $|P|$ unaltered global alignments, and $|P|$ locally-advised alignments); and

(2) local advising on the *best* global alignment, which chooses the best global alignment, and then uses adaptive local realignment to boost its accuracy.

We compare both ways of combining local and global advising, as well as local advising on the default alignment, 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 universe of parameters used for parameter advising can be found in [15] 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 structure. In particular, we use the `BENCH` suite of Edgar [9] (which is a combination of the `BAliBASE` [1], `PREFAB` [10], `OxBench` [20], and `SABRE` [24] databases), supplemented by a selection from the `PALI` suite of Balaji et al. [2]. The full benchmark collection 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 *hardness*, which we measured by the true accuracy of `Opal` using the 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 hardness interval $\big((b-1)/10, b/10\big]$, and has 12, 12, 20, 34, 26, 50, 62, 74, 137, and 434 benchmarks respectively. We report the average accuracy across *bins* rather than across benchmarks. This means that the average accuracy of alignments using the `Opal` default parameter settings is near 50%. Even though the binning is based on the `Opal` default alignments, most other standard aligners have default accuracy near 50% as well: `Clustal Omega` [22], 47.3%; `MUSCLE` [10], 48.4%; `MAFFT` [14], 51.0%. The methodology presented here is general and can be implemented for any other aligner.

We developed a universe of alignment parameter settings by enumerating reasonable values of each of the tunable alignment parameters for the `Opal` aligner. In particular the tunable parameters for `Opal` are represented as a 5-tuple $(\sigma, \lambda_I, \lambda_T, \gamma_I, \gamma_T)$ which represent the replacement matrix (σ) and the internal and terminal gap open (λ) and extension costs (γ). For the substitution matrix we selected 3 matrices form the `BLOSUM` [13] and `VTML` [18] families, two choices of terminal gap extension costs, and three choices each of internal gap extension, terminal gap open and internal gap open costs. In total we generate a universe of 162 parameter settings.

We use 12-fold *cross validation* to examine the increase in accuracy gained using adaptive local realignment. We first evenly and randomly distributed benchmarks into twelve groups for each hardness bin; the 12 independent folds are 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 generate an alignment for each benchmark in the training or testing set of each fold using each of the parameters in our universe and the `Opal` aligner. The results we reported are averages over these twelve folds. (Note that across twelve folds, every example 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 that there was very little change in coefficients between the training folds so for simplicity we use the estimator coefficients that are release with the newest version `Facet` which were trained on all available benchmarks. We also use the `TCS` estimator for adaptive local realignment, these results are in Section 4.3.

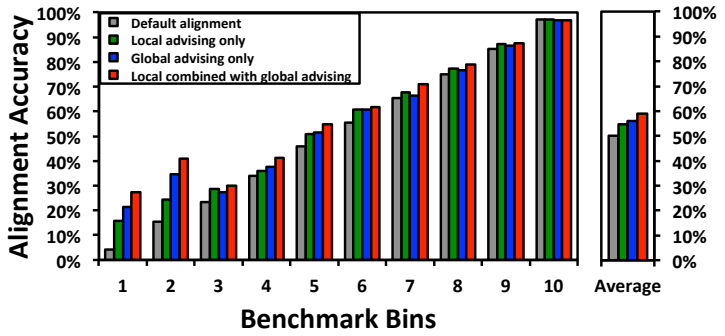


Fig. 5. *Accuracy of the default alignment, and different advising methods, within difficulty bins.* The horizontal axis shows all ten benchmarks bins. The vertical axis shows the accuracy averaged over just the benchmarks in that bin using default parameter settings, local advising only, global advising 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.

To choose the parameters for adaptive local realignment, we tested the cross product of reasonable values for the tunable parameters. We used the performance on *training* benchmarks described 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 the tunable parameters, the range of values over which we tested and the value we selected for use in our experiments. Details on the iteration count selection are in Section 4.4.

4.1 Effect of local realignment within difficulty bins

Figure 5 shows the alignment accuracy across difficulty bins for default alignments from `Opal`, local advising on these default alignments, global advising alone, and local combined with global alignment. Here the combination method uses local advising on all alternate alignments within global advising. The oracle set of cardinality $k = 10$ was used for both global and local advising.

Table 1. Adaptive Local Realignment Parameter Selection

Parameter	Range of Values	Chosen
window length fraction, w	$0.05, 0.1, 0.2, 0.3, \ldots, 0.7$	0.3
window length lower bound	$5, 10, 20, 30$	10
window length upper bound	$30, 50, 75, 100, 125$	30
barrier label percentages, T_B	$5\%, 10\%, 20\%, 30\%, \ldots, 70\%$	10%
seed label percentages, T_B	$5\%, 10\%, 20\%, 30\%, \ldots, 70\%$	30%
geometric decay rate, d	$0.5, 0.66, 0.9, 0.99$	0.9
iterations	$1, \ldots, 5, 10, 15, 25$	5

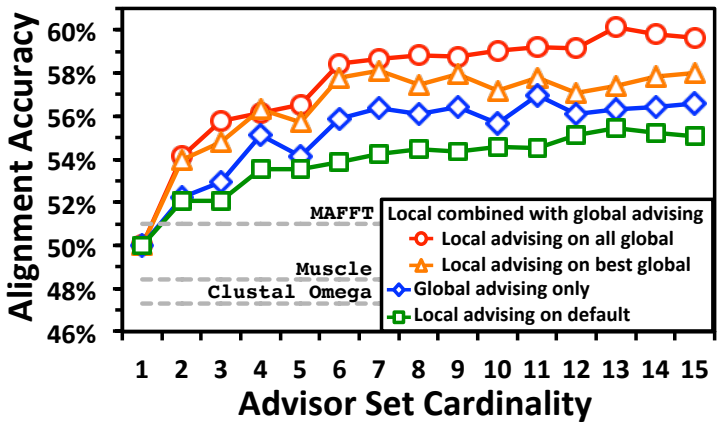


Fig. 6. *Advising accuracy versus advisor set cardinality.* The horizontal axis is the cardinality of the advisor set used by the advising methods. The vertical axis shows the advising accuracy of the default parameter setting, local advising, global advising, and the combined advising method, averaged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

The improvement gained by using adaptive local realignment over the default parameter setting is most evident in the two most difficult benchmark bins using local advising increases the average accuracy by 11.5% and 9.1% respectively, but the accuracy increases on all bins. Overall using local advising increases the accuracy of the default alignments by and average of 4.5% across bins.

Combining local and global advising substantially improves the accuracy over either of the methods individually. This is most pronounced for the hardest to align benchmarks. For the bottom two bins using both parameter advising and adaptive local realignment increases the accuracy by 23.0% and 25.6% accuracy over using just the default parameter choices. Additionally, using adaptive local realignment increases the accuracy by 5.9% and 6.4% accuracy on the bottom most bins over using parameter advising alone. On average thats an 8.9% increase in accuracy over all bins by using the combined procedure over using just the default parameter choice and a 3.1% increase over using only parameter advising.

4.2 Varying advisor set cardinality

Since an alignment is produced for each region of local realignment for each parameter choice in the advisor set, and the running time is dependent on the size of the advisor set, it may be desirable to use a smaller set than used in the previous section to reduce the running time of local (and global) advising. We produced oracle advisor sets for cardinalities $k = 2, \ldots, 15$ and used them to test the effect of local advising both alone and in combination with global advising. Figure 6 shows the average advising accuracies of using advisor sets of increasing cardinalities for adaptive local realignment applied to the `Opal`

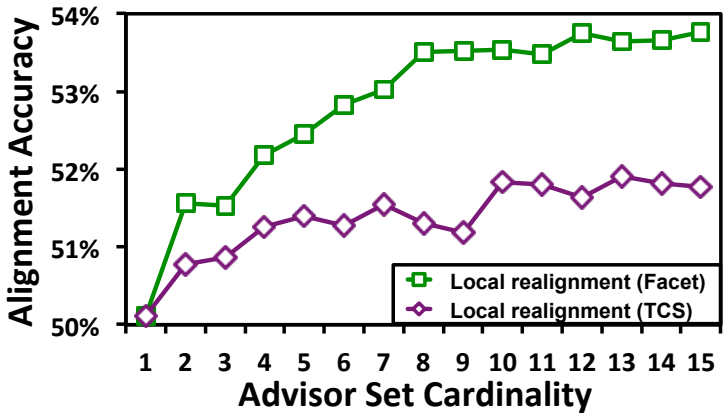


Fig. 7. *Accuracy of the default alignment and local realignment using* `TCS` *and* `Facet` *with various advisor set cardinalities.* This figure compares the accuracy of alignments produced by the `Opal` default parameter settings applying local realignment using either the `TCS` or `Facet` estimator. The horizontal axis is the cardinality of the oracle advising set used for local realignment. The vertical axis shows the accuracy of the alignments produced by each of the advising methods, averaged across difficulty bins.

default alignment, global advising alone, and global combined with local advising. The figure shows the accuracy of both combined local and global advising strategies described in Section 3.4: adaptive local realignment applied to best global alignment found though advising and adaptive local realignment applied to all global alignments. The cardinality of the set used for both global and local advising is shown on the horizontal axis, while the vertical axis shows alignment accuracy uniformly averaged across bins.

The accuracy of alignments produced by all four methods shown eventually reaches a plateau where adding additional parameters to the advisor set no longer increases the alignment accuracy. This plateau is reached at cardinality $k = 10$ when local realignment is applied to the default alignments and at $k = 6$ for parameter advising with and without local realignment, but this plateau is higher for the combined methods. Across all cardinalities using local combined with global advising improves alignment accuracy by nearly 4% on average. Note that when local realignment is applied to all global alignments the advisor is now choosing from a set of alignments which have higher accuracy than their corresponding original alignments.

The results above uniformly average advising accuracy across *bins*. In contrast, if we report advising accuracy uniformly averaged across *benchmarks*, `Opal` on its default parameter choice 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 $k=10$). Other aligners have accuracies: `Clustal Omega`, 77.3%; `MUSCLE`, 78.1%; `MAFFT`, 79.4%.

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 [15, 5]). 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`.

4.4 Effect of iterating local realignment

As discussed in Section 3.3, iterating local advising should eventually reach a state where the alignment is no longer improving, or even worse, begins deteriorating due to noise in the accuracy estimator. Table 2 shows the average accuracy of using local adaptive realignment on the default alignment as the number of iterations is increased. The training accuracy reaches a plateau at 5 iterations; we use this number of iterations 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 had at least one realignment region in the alignment 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

Table 2. Accuracy of Adaptive Local Realignment Across Iterations

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

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%

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. Notice that while the percentage of columns covered by realignment regions stays roughly the same in the easiest-to-align benchmark bin, only 47% of the alignment columns were altered, while in the rest of the bins over 60% of the alignment columns improved.

4.6 Running time

The running time of `Opal` with adaptive local realignment, averaged across all benchmarks, increases to 110 seconds when using an advisor set of cardinality $k = 10$, and 5 iterations. This is up from 36 seconds for one iteration, and about 8 seconds for the default parameter settings. This high increase in wall-clock time is mainly due to the fact that, as currently implemented, adaptive local realignment does not exploit parallelism in advising. In contrast, *global* advising has been parallelized, so the average running time of global advising on the same advisor set of size $k = 10$ 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 the two methods are combined, the average running time increases to 68 and 178 seconds for local advising on the best global alignment, and local advising on all global alignments, respectively.

5 Conclusion

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

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

References

1. Bahr, A., Thompson, J.D., Thierry, J.C., Poch, O.: `BAliBASE` (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. Nucleic Acids Research 29(1), 323–326 (2001)
2. Balaji, S., Sujatha, S., Kumar, S., Srinivasan, N.: `PALI`—a database of Phylogeny and ALIgnment of homologous protein structures. NAR 29(1), 61–65 (2001)
3. Chang, J., Tommaso, P., Notredame, C.: `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 (2014)

4. DeBlasio, D., Kececioglu, J.: `Facet`: software for accuracy estimation of protein multiple sequence alignments. `facet.cs.arizona.edu` (2014)
5. DeBlasio, D., Kececioglu, J.: Learning parameter-advising sets for multiple sequence alignment. IEEE/ACM Trans. on Comp. Biology and Bioinformatics (2015)
6. DeBlasio, D.F., Wheeler, T.J., Kececioglu, J.D.: Estimating the accuracy of multiple alignments and its use in parameter advising. Proceedings of the 16th *Conf. on Research in Computational Molecular Biology* (RECOMB) pp. 45–59 (2012)
7. DeBlasio, D.F.: Parameter Advising for Multiple Sequence Alignment. PhD dissertation, Department of Computer Science, The University of Arizona (May 2016)
8. Do, C., Mahabhashyam, M., Brudno, M., Batzoglou, S.: `ProbCons`: probabilistic consistency-based multiple sequence alignment. Gen. Res. 15(2), 330–340 (2005)
9. Edgar, R.C.: `BENCH`. `drive5.com/bench` (2009)
10. Edgar, R.: `MUSCLE`: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research 32(5), 1792–1797 (2004)
11. Fitch, W.M., Margoliash, E.: 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 (1967)
12. Gotoh, O.: Optimal alignment between groups of sequences and its application to multiple sequence alignment. Comp. Appl. in the Biosciences 9(3), 361–370 (1993)
13. Henikoff, S., Henikoff, J.: Amino acid substitution matrices from protein blocks. Proc. of the National Academy of Sciences of the USA 89(22), 10915–10919 (1992)
14. Katoh, K., Kuma, K.i., Toh, H., Miyata, T.: `MAFFT` version 5: improvement in accuracy of multiple sequence alignment. Nucl. Acids Res. 33(2), 511–518 (2005)
15. Kececioglu, J., DeBlasio, D.: Accuracy estimation and parameter advising for protein multiple sequence alignment. J. of Comp. Biology 20(4), 259–279 (2013)
16. Kececioglu, J., Starrett, D.: Aligning alignments exactly. In: Proceedings of the 8th *Conf. on Research in Comp. Molec. Biology* (RECOMB). pp. 85–96. ACM (2004)
17. Löytynoja, A., Goldman, N.: Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. Science 320(5883), 1632–1635 (2008)
18. Müller, T., Spang, R., Vingron, M.: 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 (2002)
19. Notredame, C., Higgins, D., Heringa, J.: `T-Coffee`: A novel method for fast and accurate multiple sequence alignment. J. of Molecular Bio. 302(1), 205–217 (2000)
20. Raghava, G., et al.: `OXBench`: A benchmark for evaluation of protein multiple sequence alignment accuracy. BMC Bioinformatics 4(1), 1–23 (2003)
21. Roskin, K.M., Paten, B., Haussler, D.: Meta-alignment with `Crumble` and `Prune`: Partitioning very large alignment problems for performance and parallelization. BMC Bioinformatics 12(1), 1–12 (2011)
22. Sievers, F., et al.: Fast, scalable generation of high-quality protein multiple sequence alignments using `Clustal Omega`. Mol. Sys. Bio. 7(1), 539–539 (2011)
23. Thompson, J., Higgins, D., Gibson, T.: `ClustalW`: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucl. Acids Res. 22(22), 4673–4680 (1994)
24. Van Walle, I., Lasters, I., Wyns, L.: `SABmark`: a benchmark for sequence alignment that covers the entire known fold space. Bioinformatics 21(7), 1267–1268 (2005)
25. Wheeler, T.J., Kececioglu, J.D.: Multiple alignment by aligning alignments. Bioinformatics 23(13), i559–68 (2007)
26. Yang, Z.: Maximum-likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. Mol. Bio. Evol. 10(6), 1396–1401 (1993)