BIOINFORMATICS

Vol. 23 ISMB/ECCB 2007, pages i559–i568
doi:10.1093/bioinformatics/btm226

Multiple alignment by aligning alignments

Travis J. Wheeler* and John D. Kececioglu
Department of Computer Science, The University of Arizona, Tucson AZ 85721, USA

ABSTRACT

Motivation: Multiple sequence alignment is a fundamental task in bioinformatics. Current tools typically form an initial alignment by merging subalignments, and then polish this alignment by repeated splitting and merging of subalignments to obtain an improved final alignment. In general this *form-and-polish* strategy consists of several stages, and a profusion of methods have been tried at every stage. We carefully investigate: (1) how to utilize a new algorithm for aligning alignments that optimally solves the common subproblem of merging subalignments, and (2) what is the best choice of method for each stage to obtain the highest quality alignment.

Results: We study six stages in the form-and-polish strategy for multiple alignment: parameter choice, distance estimation, merge-tree construction, sequence-pair weighting, alignment merging, and polishing. For each stage, we consider novel approaches as well as standard ones. Interestingly, the greatest gains in alignment quality come from (i) estimating distances by a new approach using normalized alignment costs, and (ii) polishing by a new approach using 3-cuts. Experiments with a parameter-value oracle suggest large gains in quality may be possible through an input-dependent choice of alignment parameters, and we present a promising approach for building such an oracle. Combining the best approaches to each stage yields a new tool we call `Opal` that on benchmark alignments matches the quality of the top tools, without employing alignment consistency or hydrophobic gap penalties.

Availability: `Opal`, a multiple alignment tool that implements the best methods in our study, is freely available at http://opal.cs.arizona.edu

Contact: twheeler@cs.arizona.edu

1 INTRODUCTION

Alignments of multiple biological sequences play an important role in a wide array of bioinformatics applications, including inference of phylogenetic trees and identification of conserved and divergent regions of biomolecules. All widely used tools for multiple sequence alignment at essence seek an alignment that maximizes the *sum-of-pairs score*: the weighted sum of the scores of all pairwise alignments induced by the multiple alignment. Optimal multiple alignment with sum-of-pairs scoring is NP-complete (Wang and Jiang, 1994), which motivates, the search for good heuristics.

Most tools use a heuristic called progressive alignment (Feng and Doolittle, 1987) which has two steps: (1) construct a binary *merge tree* whose leaves are the input sequences and whose internal nodes arrange the sequences into groups, and (2) merge these groups bottom-up over the tree by combining the alignments at the two children of a node into one alignment at their parent. The alignments for two groups may be combined by aligning profiles (Gotoh, 1994; Kececioglu and Zhang, 1998) or aligning alignments (Gotoh, 1993; Kececioglu and Starrett, 2004). When this merging process reaches the root, it has formed an alignment of all the input sequences.

With this heuristic, the alignment between two sequences in a group is not altered when new sequences are added to the group. Consequently, errors made in early merges remain in the final alignment, and may lead to further misalignment in later merges. One approach to correcting such errors is to apply a second phase we call *polishing* (Berger and Munson, 1991; Hirosawa *et al.*, 1995), which refines the alignment by repeatedly splitting its sequences into subsets and realigning their induced subalignments. Another strategy is to reduce early errors using an approach called *consistency* (Notredame *et al.*, 2000; Do *et al.*, 2005). Consistency approaches assign position-specific substitution scores for a pair of sequences A and B and a pair of positions i and j that depend on the support for the substitution between i and j from the pairwise alignments of sequences A and B to all other sequences C.

1.1 Related work

The most widely-used multiple alignment tool, `ClustalW` (Thompson *et al.*, 1994) relies on a complex scoring scheme in which substitution scores and gap penalties are adjusted according to features of the aligned sequences, including divergence, length, hydrophobicity of amino acids and proximity of neighboring gaps.

`T-Coffee` (Notredame *et al.*, 2000) and its predecessor `Coffee` (Notredame *et al.*, 1998) introduced alignment consistency. `T-Coffee` assigns position-specific substitution scores based on a mixture of information from global and local alignments.

Early versions of `MAFFT` (Katoh *et al.*, 2002) substantially increased alignment speed without sacrificing accuracy through a host of ideas including scoring system modifications, use of the fast Fourier transform to speed up profile alignment, and a fast two-stage method for building an initial alignment based on k-mer frequencies. Recent versions of `MAFFT` (Katoh *et al.*, 2005) incorporate `T-Coffee`-like consistency, resulting in a substantial improvement in accuracy.

`Muscle` (Edgar, 2004) further improved speed and accuracy, using reduced terminal gap costs and a measure called log-expectation to score alignments of profiles.

`ProbCons` (Do *et al.*, 2005) introduced probabilistic consistency, which assigns position-specific substitution scores based on a measure of expected accuracy derived from a hidden

*To whom correspondence should be addressed.

Markov model. It also offers a column reliability score that estimates the likelihood that each column represents a correct alignment of residues.

To summarize general techniques: MAFFT, Muscle and ProbCons employ polishing; ClustalW, T-Coffee and Muscle use hydrophobic gap penalties; T-Coffee, MAFFT and ProbCons use consistency.

1.2 Contributions

While these tools use diverse techniques, they can all be viewed as following a common series of *stages* in what we call the form-and-polish strategy for multiple alignment: parameter choice, distance estimation, merge-tree construction, sequence-pair weighting, alignment merging, and polishing. Current tools use a profusion of methods for each stage, and it can be difficult to determine which methods are contributing to their accuracy, and should be in a best-of-breed tool.

We carefully study the impact of the standard methods for each stage, and offer several new methods that substantially improve accuracy. Surprisingly, the greatest gains in accuracy come from two simple and new ideas: (i) estimating distances between sequences for merge tree construction by normalized alignment costs, and (ii) polishing the final alignment using 3-partitions of the input sequences induced by cutting pairs of edges in the merge tree.

By combining the best methods for each stage of the form-and-polish strategy, we obtain a new tool we call Opal whose accuracy matches the state-of-the-art, without employing elaborate approaches like hydrophobicity or consistency. In a sense, this shows that some of the more involved ideas these tools have contributed are not required to attain their accuracy. This does not imply, however, that ideas such as hydrophobicity and consistency are not valuable for multiple alignment. Indeed, adding them to Opal may boost its accuracy further.

1.3 Methodology

The standard practice for evaluating multiple alignment tools is to use benchmark datasets of reference alignments that are usually based on structural alignment of proteins. When comparing methods for a stage, and comparing alignment tools, we evaluate accuracy by measuring the recovery of reference alignments from three standard suites of protein alignment benchmarks: BAliBASE 3.0 (Thompson *et al*., 1999; Bahr *et al*., 2001), SABmark 1.65 (Van Walle *et al*., 2004) and PALI 2.5 (Balaji *et al*., 2001). These suites, which have been used by many comparative studies, of course represent only a sample of the types of inputs biologists face.

BAliBASE is a collection of 218 reference alignments based on structural alignments with manually-arranged gaps, exhibiting a variety of phylogenetic and structural characteristics. We limited our tests to the 163 alignments with no more than 40 sequences, as our focus is measuring accuracy and not speed.

SABmark contains 627 benchmarks with at most 25 sequences that cover the space of folds in the SCOP classification (Murzin *et al*., 1995) of protein families. Each benchmark is a collection of pairwise structural alignments that are not necessarily consistent with one multiple alignment.

PALI contains 1655 alignments of all SCOP families constructed by structural multiple alignment without hand curation. We used a subset of 102 alignments consisting of all reference alignments with at least 7 sequences that have non-trivial gap structure.

In our experiments, the measure of accuracy is mainly the so-called *SPS score* (Bahr *et al*., 2001): the percentage of pairs of aligned positions from the reference alignment that were correctly recovered. In some cases, we also report the so-called *TC score* (Bahr *et al*., 2001): the percentage of columns from the reference alignment that were completely recovered; this score is appropriate for BAliBASE and PALI, but not SABmark. For BAliBASE and PALI, both scores are measured on their *core blocks*: those columns in the reference alignment that are deemed reliable by the benchmark (typically through strong support from a structural alignment). Experiments were run on a 3.0 GHz Pentium IV with 2 GB of RAM.

We use the above suites of benchmarks in experiments to determine the best method for each of the six stages of the form-and-polish strategy for multiple alignment. While our presentation may appear to ignore interactions between methods for different stages, results omitted due to page limits show the best method for a given stage is independent of the choices at other stages. The best method is also generally independent of which suite of benchmarks is considered.

1.4 Overview

The sections of the paper are organized roughly according to the stages of the form-and-polish strategy. In the next section we study methods for constructing the merge tree. Section 3 considers methods for merging alignments. Section 4 assesses the effect of weighted sum-of-pairs. Section 5 explores methods for polishing alignments. Section 6 examines the impact of parameter choice. Finally Section 7 compares the combined approach to current tools.

2 CONSTRUCTING THE MERGE TREE

Constructing a merge tree involves (i) grouping sequences hierarchically, based on (ii) a measure of distance between sequences. We study these two aspects in turn.

2.1 Grouping sequences

Common approaches to constructing a merge tree maintain a partition of the input sequences into groups. Initially every sequence is in its own group, which forms a leaf of the tree. The general step merges the pair of groups that are closest based on a measure of distance between groups, which defines an internal node of a binary tree. Merging continues until one group remains, which is the root of the tree. The tree may be rerooted at a different node, for instance to balance root-to-leaf path lengths, but in our experiments (not shown) this gives no improvement.

We study five grouping methods, the last of which is new.

2.1.1 NJ Neighbor joining (Saitou and Nei, 1987), or NJ, is the method used by ClustalW and T-Coffee. NJ merges the groups that are closest according to an estimate of evolutionary distance, and is generally regarded as the best of the distance-based methods at producing the true evolutionary tree for the sequences.

Let G be the current set of groups during the merging process, and $d_{a|b}$ be the current distance between a pair a, b of groups. NJ merges the groups a and b that minimize

$$d_{a|b} \; - \; \frac{1}{|G-\{a,b\}|} \sum_{c \in G-\{a,b\}} \Big(d_{a|c} \; + \; d_{b|c} \Big).$$

The new distance $d_{ab|c}$ between the merged group ab and all other groups c is

$$d_{ab|c} \; := \; \frac{1}{2} \Big(d_{a|c} \; + \; d_{b|c} \; - \; d_{a|b} \Big).$$

Neighbor joining takes $O(k^3)$ time for k sequences.

2.1.2 UPGMA and MST The unweighted-pair group method with arithmetic mean (Sneath and Sokal, 1973), or UPGMA, and minimum spanning tree, or MST, are simpler approaches that run in $O(k^2)$ time. Both merge the pair a, b of groups with minimum distance $d_{a|b}$, but differ in how they define the distance $d_{ab|c}$ from the merged group ab to all other groups c. UPGMA sets $d_{ab|c}$ to the average $\frac{1}{2}(d_{a|c} + d_{b|c})$, while MST uses $\min\{d_{a|c}, d_{b|c}\}$. The multiple alignment literature (Edgar, 2004; Do *et al.*, 2005) suggests that using a merge tree that is similar to the correct evolutionary tree is less important than using a tree that groups similar sequences first, which may explain the superiority (confirmed below) of these simpler methods over NJ.

We also considered the mixture UPGMA + MST used by MAFFT, which for $0 \le \alpha \le 1$, sets $d_{ab|c}$ to the convex combination

$$\alpha \, \min\{d_{a|c}, d_{b|c}\} \; + \; (1-\alpha) \, \frac{1}{2} \, (d_{a|c} + d_{b|c}).$$

2.1.3 DAD A shortcoming of the above methods is that distances are derived from sequences only at initialization. When group ab is formed, the new distances $d_{ab|c}$ are calculated from original sequence distances, which ignores the constraints on sequence pairs across groups ab and c imposed by the alignments for these groups.

We evaluated several new methods that take such constraints into account. The best of these, which we call dynamic alignment distance, or DAD, computes $d_{ab|c}$ by aligning the current alignments for ab and c to obtain an alignment $\mathcal{A}$ for group abc, and then taking for $d_{ab|c}$ the *minimum* of the distances measured on the pairwise alignments in $\mathcal{A}$ induced by sequence pairs across groups ab and c (analogous to MST). The distance measure on pairwise alignments can be any of those from Section 2.2.

We also considered variants that use the *average* of the distances of the induced pairwise alignments between ab and c (analogous to UPGMA), and variants that add in the average distances between abc and all *other* groups d (similar to consistency), but the above method performed the best.

DAD does not perform as well as the seemingly less-informed methods NJ, UPGMA and MST. While there is more information in the aligned sequences, alignments against large groups are more constrained than those against small groups, so larger groups tend to have higher distances. This causes smaller groups to be merged first, which leads to a balanced merge tree even when this is undesirable.

2.1.4 Comparison Below, we compare the *accuracy* of these methods on three suites of benchmarks in terms of their SPS score. As our baseline for the other stages, we measure sequence distances using percent identity over a compressed alphabet (Section 2.2), merge alignments using pessimistic gap counts (Section 3), use unweighted sum-of-pairs (Section 4), no polishing (Section 5), and default parameters (Section 6). For UPGMA + MST, we use $\alpha = 0.9$ as in MAFFT. The best accuracies are in bold.

Tree method	BAliBASE	SABmark	PALI	Average
MST	**79.4**	**44.1**	79.8	**67.8**
UPGMA + MST	79.2	44.0	80.2	**67.8**
UPGMA	78.0	42.7	**80.5**	67.1
NJ	77.4	42.1	77.2	65.6
DAD	78.2	43.5	73.0	64.9

Generally, the methods based on minimum spanning trees outperform the others. For the merge tree method in the rest of the paper, we chose the simpler method MST.

2.2 Measuring distances

The standard measure of distance between two sequences is based on *percent identity*: the percentage of matched positions in an optimal pairwise alignment that are identities. (The actual distance used is the complement of percent identity.) Many tools modify percent identity by the Kimura correction for multiple substitutions at a locus (Kimura, 1983), and measure it over a compressed alphabet that groups amino acids with similar characteristics into equivalence classes (Dayhoff *et al.*, 1978), which we call *compressed identity*.

Percent identity is a coarse measure of similarity, while alignment cost is a more refined measure that can be obtained with no overhead. We also tested a new distance measure, which we call *normalized alignment cost*, that simply normalizes the cost of an optimal pairwise alignment by dividing by its number of columns (where alignment cost uses affine gap penalties, as discussed in Section 3). In a sense, this generalizes percent identity and compressed identity to the full spectrum of substitutions while also taking into account gaps.

2.2.1 Comparison Below, we compare the accuracy of these three distance methods. We use an MST merge tree, pessimistic gap counts, no weighting, no polishing, and default parameters. Identity measures have Kimura correction.

Distance method	BAliBASE	SABmark	PALI	Average
Normalized cost	**81.6**	**48.2**	**83.0**	**70.9**
Compressed identity	79.4	44.1	79.8	67.8
Percent identity	78.5	43.5	79.9	67.3

Normalized cost gives a striking boost in accuracy, which persists even after polishing (not shown due to page limits). The rest of the paper measures distances by normalized costs.

3 MERGING ALIGNMENTS

Merging two multiple alignments of disjoint groups of sequences into one alignment of all the sequences is central to both forming an initial multiple alignment and polishing a final alignment. During the initial alignment phase, subalignments $\mathcal{A}$ and $\mathcal{B}$ at the children of a merge tree node are combined into one alignment $\mathcal{C}$ of all the sequences in the subtree. During the polishing phase, the final alignment is repeatedly split into subalignments $\mathcal{A}$ and $\mathcal{B}$ that are recombined into an updated alignment $\mathcal{C}$. Clearly, the quality of the final alignment strongly depends on how subalignments are merged in both phases.

When merging alignments $\mathcal{A}$ and $\mathcal{B}$ to form $\mathcal{C}$, we take as our objective to optimize the *sum-of-pairs score* (Carillo and Lipman, 1988) of alignment $\mathcal{C}$, which is the sum of the scores of all induced pairwise alignments in $\mathcal{C}$. The sum-of-pairs score may be weighted, as discussed in Section 4. When merging $\mathcal{A}$ and $\mathcal{B}$ to form $\mathcal{C}$, subalignments $\mathcal{A}$ and $\mathcal{B}$ are preserved within $\mathcal{C}$.

Computing an optimal $\mathcal{C}$ given $\mathcal{A}$ and $\mathcal{B}$ using sum-of-pairs scoring has been called the problem of *aligning alignments* (Kececioglu and Zhang, 1998), and was first considered by Gotoh (1993). Scores of the alignments are usually evaluated using *affine gap penalties*, where a gap of length ℓ in a pairwise alignment has cost $\gamma + \lambda\ell$, for constants γ and λ. (A gap of length ℓ is a run of either ℓ insertions or deletions.) The constants γ and λ are called the gap open and extension penalties, and may have different values for terminal or internal gaps. When computing an optimal merge $\mathcal{C}$ by dynamic programming, the essential difficulty is in correctly counting the total number of gaps incurred in the induced pairwise alignments of $\mathcal{C}$.

We study two basic ways of computing the merge $\mathcal{C}$ by aligning alignments: using *exact* gap counts, which yields a merge that has optimal sum-of-pairs score; and using *pessimistic* gap counts, which is a fast heuristic that may yield a suboptimal merge. There are also a variety of ways of merging by aligning profiles (Gotoh, 1994; Kececioglu and Zhang, 1998; Edgar, 2004).

3.1 Exact counts

Surprisingly, computing an optimal merge $\mathcal{C}$ of two alignments $\mathcal{A}$ and $\mathcal{B}$ with affine gap penalties is NP-complete (Ma *et al.*, 2003). While this shows there is likely no algorithm that computes an optimal merge and is fast in the *worst case*, Kececioglu and Starrett (2004) developed an exact algorithm that computes an optimal merge and is remarkably fast in *practice*. To optimally align two alignments, each having k sequences and n columns, their algorithm takes worst-case time $O(5^k n^2)$. Extensive experiments, however, show empirically that it runs in $O(k^2 n^2)$ time on biological data (Kececioglu and Starrett, 2004).

In this study, we compute an optimal merge $\mathcal{C}$ with exact gap counts using the tool `AlignAlign` (Starrett *et al.*, 2005), which implements the algorithm of Kececioglu and Starrett (2004).

3.2 Pessimistic counts

Another approach to computing the merge $\mathcal{C}$ is to avoid the difficulty of determining exact gap counts by instead using an approximation introduced by Altschul (1989a) called pessimistic gap counts (Kececioglu and Zhang, 1998). This approximation overestimates the true number of gaps by assuming, in cases where the number of gaps started by a multiple alignment column is not determined by the preceding column, that the number of gaps started attains its largest possible value. The benefit of pessimistic gap counts is that the merge of two alignments that is best under this estimate can always be found efficiently. Computing the best pessimistic merge of two alignments over a constant-size alphabet, where each alignment has k sequences and n columns, takes worst-case time $O(kn + n^2)$. `AlignAlign` also implements pessimistic counts.

Most multiple alignment tools use some version of profile alignment to merge alignments, which in general counts gaps less accurately than the pessimistic approach. It is entirely possible that the pessimistic heuristic is outperforming such profile methods, though we do not study that here.

3.3 Comparison

Below, we compare the accuracy of exact and pessimistic gap counts when merging alignments. We use an `MST` merge tree with normalized costs, no weighting, no polishing, and default parameters.

Merge method	`BAliBASE`	`SABmark`	`PALI`	Average
Exact	**82.4**	**48.4**	**84.0**	**71.6**
Pessimistic	81.6	48.2	83.0	70.9

Exact gap counts are consistently superior to pessimistic gap counts, though only slightly, and this persists even after polishing (results omitted due to page limits). We use exact counts in the rest of the paper.

4 WEIGHTING SEQUENCE PAIRS

Sum-of-pairs scoring of multiple alignments is a potentially biased scoring measure. If the input sequences are not independent but instead over-sample some groups compared to others, the higher number of pairwise alignments to an over-sampled group can dominate the alignment score. This greater contribution of an over-sampled group to the score will tend to drive the multiple alignment toward improving the pairwise alignments to such groups at the price of worsening the pairwise alignments to under-sampled groups, thus degrading the overall quality of the alignment.

Several schemes have been proposed to correct for this bias by non-uniformly weighting the pairwise alignment scores in the sum-of-pairs measure. We study three such schemes. The first is new, and the other two are the schemes often used by current alignment software.

All these schemes assign weights to pairs of input sequences on the basis of a tree T whose leaves correspond to the sequences, and that has edge lengths ℓ_e for all edges $e \in T$. Each scheme assigns a weight w_{ij} to a pair i, j of leaves in T. Suppose that in a multiple alignment $\mathcal{A}$ of the sequences, the score of the induced pairwise alignment of the sequences associated with leaves i and j is s_{ij}. The *weighted sum-of-pairs score* of alignment $\mathcal{A}$ using these weights is

$$\sum_{i,j} w_{ij}\, s_{ij}.$$

4.1 Influence weights

One way to assign a weight w_{ij} to a pair i, j of leaves is to quantify the *influence* of one leaf on another on the basis of the shape of T and its edge lengths. Suppose we have a measure $\omega(i,j)$ of the influence of leaf j on leaf i in T where function ω is not necessarily symmetric. Then we can define a symmetric weight on a pair of leaves by the geometric mean of their influences:

$$w_{ij} \;:=\; \sqrt{\omega(i,j)\;\omega(j,i)}.$$

We call the w_{ij} obtained in this way, *influence weights*.

Our influence function ω is nonnegative, and for every leaf i it satisfies

$$\sum_{j\,:\,j\neq i} \omega(i,j) \;=\; 1.$$

As a consequence, the resulting weights satisfy $0 \le w_{ij} \le 1$.

To determine the influence $\omega(i,j)$ of leaf j on leaf i, we carry out the following recursive process. Imagine taking leaf i, making i the new root of T and letting T hang from root i. We denote this *rerooted tree* with root i by T_i. Starting from root i, we process T_i top-down, splitting the total mass of weight 1 in the above equation among the descendants of i. The new root i has exactly one child (which was originally the parent of i in T), and this child receives the entire mass of weight 1 passed down from its parent i. In general, if a descendant x receives mass w_x from its parent, this mass is split among its two children y and z (possibly unequally) so that

$$w_x \;=\; w_y + w_z.$$

After completing this top-down splitting process, we take as the influence of leaf j on root i, the amount of the original mass 1 at the root that ends up at leaf j:

$$\omega(i,j) \;:=\; w_j.$$

The key is determining how to split the mass at a parent among its two children on the basis of the edge lengths of T. We do such a split as follows.

For nodes v and w of T, denote the path in T between v and w by P_{vw}. For a node v, let $T_u(v)$ be the subtree of T_u rooted at node v together with the path P_{uv}. The total *size* of $T_u(v)$ is

$$S_u(v) \;:=\; \sum_{e\,\in\, T_u(v)} \ell_e.$$

Denote the length of path P_{xy} by ℓ_{xy}, and the *set of leaves* in $T_u(v)$ by $L_u(v)$. The average *height* of $T_u(v)$ is

$$H_u(v) \;:=\; \frac{1}{|L_u(v)|} \sum_{x\,\in\, L_u(v)} \ell_{ux}.$$

We call the effective *number of leaves* in $T_u(v)$,

$$N_u(v) \;:=\; \begin{cases} \dfrac{S_u(v)}{H_u(v)}, & H_u(v) \neq 0; \\[2ex] 1, & \text{otherwise.} \end{cases}$$

The effective number of leaves satisfies $1 \le N_u(v) \le |L_u(v)|$.

In tree T_i, we split weight $w_x = w_y + w_z$ between the two children y and z of x according to the ratio,

$$\frac{w_y}{w_z} \;=\; \frac{N_x(y)}{N_x(z)}\,\frac{H_i(z)}{H_i(y)}$$

(where when $H_i(y) = 0$ and $H_i(z) \neq 0$, we assign node y all the weight). For example, with children that have identical average heights and effective numbers of leaves $N_x(y) = 1$ and $N_x(z) = 3$, child y gets 1/4 of w_x and child z gets 3/4.

Splitting the weight $w_i = 1$ top-down over T_i according to these ratios fully specifies the leaf weights w_j, and hence the influence function $\omega(i,j)$ and the weights w_{ij}.

Influence weights have several nice properties, such as being independent of where the tree is rooted. For k sequences, the weights w_{ij} can be computed in time $O(k^2)$, which is optimal. We omit the details, due to page limits.

4.2 Covariance weights

Perhaps the best-known weights for sum-of-pairs multiple alignment are those of Altschul *et al.* (1989). Of the two weighting schemes they suggest, their second scheme, which they call rationale-2 weights, has the more rigorous basis and is the one that has been more widely adopted. We call their second scheme, *covariance weights*.

We must refer the reader to the original paper (Altschul *et al.*, 1989) for the details of this scheme. For k sequences, the formula for weights w_{ij} inverts a matrix of $O(k^4)$ covariances. Altschul (1989b) was able to show, by an involved algorithm that avoids matrix inversion, that these weights can be computed in $O(k^2)$ time. This scheme is not directly used in common software; instead Gotoh's 3-way method (Gotoh, 1995) of approximating covariance weights is normally used.

Covariance weights can exhibit counterintuitive behavior. For example, consider a tree with three leaves x, y, z, where x and y are children of the same parent v. Suppose edge lengths ℓ_{vx} and ℓ_{vy} are very small compared to path length ℓ_{vz}. If $\ell_{vx} = c\,\ell_{vy}$, then $w_{yz} \approx c\, w_{xz}$ even for arbitrarily long ℓ_{vz}. Influence weights do not exhibit this anomaly, but converge to $w_{yz} \approx w_{xz}$ as ℓ_{vz} grows.

4.3 Division weights

The program `ClustalW` introduced a weighting scheme that has been incorporated into other alignment software as well, such as `MAFFT` and `Muscle`. We call `ClustalW`'s scheme, *division weights*.

In this scheme, each leaf i of T is first assigned a weight w_i as follows. The length ℓ_e of an edge from parent x to child y is equally divided among the k leaves in the subtree rooted at y, and the portion ℓ_e/k that leaf i gets is accumulated in w_i. Weight w_i is the total of these portions from all edges on the path from leaf i to the root of T. The weight for pair i, j is defined to be $w_{ij} := w_i w_j$. The w_{ij} for k sequences can be computed in time $O(k^2)$.

Division weights can also exhibit counterintuitive behavior. Consider again a tree with three leaves x, y, z where x and y are children of v and the root is u. Suppose lengths ℓ_{vx} and ℓ_{vy} are very small compared to lengths $\ell_{uv} = \ell_{uz}$. Then $w_{xz} = w_{yz} \approx 2w_{xy}$. Influence weights avoid this anomaly, and converge to $w_{xz} = w_{yz} \approx w_{xy}/(2\,\ell_{vz})$ as $\ell_{uv} = \ell_{uz}$ grows.

4.4 Comparison

Below, we compare the accuracy of each weighting method. We use an MST merge tree with normalized costs, exact gap counts, no polishing, and default parameters. (Polishing yields the same ranking of methods.) Covariance weights are computed exactly using matrix inversion. The edge lengths ℓ_e for the weighting methods are computed from normalized costs d_{ij} by fitting edge lengths to T so as to minimize the L_1 norm between path lengths ℓ_{ij} and the distances d_{ij}. These optimally-fitted edge lengths are efficiently computed by solving a linear program. (We omit the details of the linear programming formulation due to page limits.) On BAliBASE and PALI, we also report the TC score as the second measure of quality, since it is less distorted by overrepresentation of groups than the SPS score.

Weighting method	BAliBASE	SABmark	PALI	Average
Uniform	**82.4** / **53.6**	**48.4**	84.0 / 57.3	**71.6** / **55.5**
Influence	82.2 / 53.3	**48.4**	84.1 / **57.7**	**71.6** / **55.5**
Division	82.2 / 53.4	48.2	**84.3** / 57.3	**71.6** / 55.4
Covariance	82.1 / 53.2	**48.4**	83.5 / 57.5	71.3 / 55.4

Surprisingly, and in contrast to what has generally been suggested in the literature (Gotoh, 1995; Edgar, 2004) *unweighted* sum-of-pairs (the uniform row of the table) performs as well as all three weighting schemes. Even for inputs with the largest numbers of sequences, where over-representation is more likely, weighting continues to give no benefit under both measures of quality.

While this behavior might be due to most benchmark alignments not containing strongly overrepresented groups, BAliBASE references 2 and 3 do provide a set of inputs that are likely to suffer from overrepresentation. On the 28 inputs from these two references, the ranking of schemes is influence, uniform, division, covariance, with corresponding SPS scores of 83.3, 82.8, 82.5, 81.5. Though these results are encouraging, the sample size is small. Accordingly, we use uniform weighting in the rest of the paper.

5 POLISHING THE ALIGNMENT

When forming an alignment with a merge tree, errors in early merges can accumulate in later merges and degrade the resulting alignment. The two basic approaches to correcting such errors are to try to (1) avoid them in the first place using consistency (Notredame *et al.*, 2000; Do *et al.*, 2005) and (2) fix them afterwards by polishing (Berger and Munson, 1991; Hirosawa *et al.*, 1995).

With *polishing*, the sequences are partitioned into two groups and the subalignments $\mathcal{A}$ and $\mathcal{B}$ induced on these groups are realigned, usually without altering $\mathcal{A}$ or $\mathcal{B}$. Realignment of $\mathcal{A}$ and $\mathcal{B}$ is done as when merging alignments in Section 3 by aligning profiles or aligning alignments. The resulting alignment is retained if its score improves. This process is a form of local search, and is repeated for a fixed number of iterations or until there is no improvement.

Tools in wide use that employ polishing partition the sequences into two groups either randomly (Do *et al.*, 2005) or based on the merge tree. Tree-based approaches partition the leaves by repeatedly cutting edges of the merge tree, either cycling over all edges (Edgar, 2004) or randomly choosing edges (Katoh *et al.*, 2005). Tools using random choices tend to do many iterations: ProbCons by default does 100 iterations, and MAFFT does 1000.

We considered many new polishing methods. Below, we report results for the best three and their combinations.

5.1 Exhaustive 2-cut

We implemented a tree-based method we call exhaustive 2-cut that cuts tree edges and realigns until there is no improvement. Since a tree with k leaves has $O(k)$ edges, if there is an edge whose cut gives improvement this finds it within a linear number of realignments.

Rather than scanning tree edges in a fixed order (Edgar, 2004) we dynamically order the edges e by a measure $\Phi(e)$ of their potential for improvement. For edge e, let $P(e)$ be the set of pairs of sequences that are on opposite sides of the partition given by cutting e, let c_{ij} be the cost of the pairwise alignment induced on sequences i and j in the current multiple alignment, and let d_{ij} be the cost of their optimal pairwise alignment. We use the potential

$$\Phi(e) \;:=\; \frac{1}{|P(e)|} \sum_{(i,j)\,\in\, P(e)} \big(c_{ij} \;-\; d_{ij}\big).$$

These potentials are updated after several cuts alter the alignment (typically five). This approach yields a slight speedup in convergence over the method used by Edgar (2004) while attaining the same accuracy.

5.2 Random 3-cut

Consider the situation where two sequences in a large alignment are misaligned. The 2-cut method cannot separate these two sequences from the rest of the input and realign them without interference from all other sequences. This can be easily accomplished, however, by instead partitioning into *three* groups. We examined a variety of methods for three-way partitioning, both tree-based and not. We report the best one here, which we call *random 3-cut*.

This method partitions the sequences by cutting *two* tree edges selected at random that are not on the same path from the root. (A tree with k leaves has $O(k^2)$ such cuts, so an exhaustive approach would be slow.) The resulting groups a, b, and c are merged in two steps by realigning a and b to form group ab (or alternately keeping the alignment of a and b), and then realigning c with ab. We consider the three merge orders $ab|c$, $ac|b$, and $bc|a$, and retain the best of the three if it gives improvement in score. This process is repeated until a time or iteration limit is reached. Most edges are near the leaves, so this method tends to split off two relatively small groups of sequences, enabling repair of errors between small groups followed by integration into the rest of the alignment.

In contrast to 2-cut, this method in essence alters the merge tree. We also considered more complicated 3-cut variants that reattach tree edges to reflect the merges of the three groups, or that rebuild the tree on the affected paths up to the root, but surprisingly none gave better quality than the above method that does not change the tree.

5.3 On-the-fly

An attractive idea is to polish subalignments as they are formed (Subbiah and Harrison, 1989), rather than waiting for a complete alignment. This allows errors to be fixed before causing further misalignment. We implemented a version we call *on-the-fly* polishing: when an internal node v is created during merge tree construction, cut one edge to a grandchild or child of v, realign, and repeat until no improvement. This is similar to exhaustive 2-cut, but scans a limited set of edges and operates while forming the initial alignment.

5.4 Combined

We also considered combining these methods by following on-the-fly polishing with either exhaustive 2-cut or random 3-cut polishing, which in general we call *k-cut + on-the-fly*.

5.5 Comparison

Below, we compare the accuracy of polishing methods using `MST` trees with normalized costs, exact gap counts, no weighting, and default parameters. For 3-cut we did 60 iterations; 2-cut and on-the-fly iterated until no improvement.

Polishing method	`BAliBASE`	`SABmark`	`PALI`	Average
3-cut + on-the-fly	84.3	**50.2**	84.6	**73.1**
3-cut	84.2	49.7	**84.8**	72.9
2-cut	**84.4**	49.8	84.7	72.9
2-cut + on-the-fly	83.6	50.0	84.5	72.7
On-the-fly	83.3	49.6	84.4	72.4
None	82.4	48.4	84.0	71.6

As can be seen, polishing helps. The k-cut methods converge to roughly the same quality, but their *rate* of convergence differs markedly: 3-cut achieves the same quality as 2-cut but is an order of magnitude faster on larger inputs. Adding on-the-fly to 2-cut hastens convergence (yet is still slower than 3-cut) and slightly boosts 3-cut. In the rest of the paper, we use 3-cut + on-the-fly polishing.

6 CHOOSING ALIGNMENT PARAMETERS

Selecting gap penalties is fraught with difficulty (Vingron and Waterman, 1994), and most practitioners simply use the default values provided by modern software. Fortunately there are now well-designed suites of alignment benchmarks, and the sequences in these benchmarks presumably represent the kinds of inputs a tool will see in the wild, which suggests parameters optimized on those benchmarks should be reasonably good choices.

We look at the effect of parameter choice on accuracy from three perspectives: finding the best *default* values, determining how well a perfect input-dependent choice given by an *oracle* can perform, and designing an *advisor* that makes a good choice.

6.1 Default

To select default parameters for aligning proteins, we trained initially on `BAliBASE`. Based on results from doing inverse parametric sequence alignment on `BAliBASE` using the tool `InverseAlign` (Kececioglu and Kim, 2006; Kim and Kececioglu, 2007), we fixed the substitution matrix at `BLOSUM62` (Henikoff and Henikoff, 1992), and identified a reasonable seed value for the gap open and extension penalties. We then enumerated a range of penalties around this seed, including variants with reduced terminal gap costs. (In total, we examined around 800 parameter choices.) From this set of parameters we selected as our default the choice that had the highest recovery over all three suites of benchmarks. With `BLOSUM62` transformed to a cost matrix in the range [0,88], our *default* parameter choice has internal gap open and extension penalties γ_I and λ_I, and terminal gap open and extension penalties γ_T and λ_T, of

$$(\gamma_I, \lambda_I, \gamma_T, \lambda_T) = (60, 38, 15, 36).$$

A reduced terminal gap open penalty is common, typically half the internal gap open penalty (Edgar, 2004), though a reduced terminal gap extension penalty is not. This parameter choice was used in all results described in prior sections.

6.2 Oracle

While the default choice performs well overall, it substantially underperforms other choices on many benchmark alignments. (In accuracy, the default is more than 10% worse than the optimal choice on about 15% of the inputs.) This leads us to ask what accuracy could be achieved if we had an *oracle* that could identify the best parameter choice for each input. We consider results with an oracle for purposes of comparison.

6.3 Advisor

Though a true oracle is unattainable, it is possible to design an *advisor* that can choose an input-dependent parameter value that improves alignment quality. We are aware of only one other attempt at implementing a method to automatically choose parameters, called `MULTICLUSTAL` (Yuan *et al.*, 1999).

We considered a variety of advisor methods, and describe two of them here. One approach is to select the parameter choice p from among a small set $\mathcal{P}$ of choices such that the alignment of the input using p has features most like those seen in good alignments under p. (Here a good alignment under p is one that was computed using p on a benchmark input $\mathcal{I}$ and that has high recovery compared to the best parameter choice for $\mathcal{I}$.) We considered features such as percent identity and gap densities, and used a selection rule similar to a Bayes classifier, but omit the details due to page limits.

Another approach works as follows. Define a *core column* to be a column in a multiple alignment where at least a fraction α of its rows have letters from the same character class in the compressed alphabet. (In the experiments, we use $\alpha = 0.9$.) Given a set $\mathcal{I}$ of input sequences to align and a candidate parameter choice x, let $\mathcal{A}_x$ be the alignment of $\mathcal{I}$ that results from using parameter choice x, and let $f(\mathcal{A}_x)$ be the number of core columns in $\mathcal{A}_x$.

The parameter choice that this advisor selects based on core columns is

$$p \ := \ \operatorname*{argmax}_{x \in \mathcal{P}} f(\mathcal{A}_x),$$

where ties are broken in favor of shorter alignments. In other words, it selects the parameter choice that yields an alignment with the most core columns. The output alignment using this advisor is $\mathcal{A}_p$.

Both of the above advisors perform similarly. We present results for the core column approach.

6.4 Comparison

Below, we compare the hypothetical accuracy of the oracle to that achieved using default parameters and the advisor. We picked a set $\mathcal{U}$ of twelve parameter choices that performed well on average and that cover the domain of reasonable gap open and extension values, and applied the oracle to set $\mathcal{U}$. A smaller set $\mathcal{P} \subseteq \mathcal{U}$ of four parameter choices was identified by selecting the subset of size four that gave the best recovery under the advisor. (An alignment $\mathcal{A}_x$ must be computed by the advisor for each $x \in \mathcal{P}$, so a small set $\mathcal{P}$ is preferable.). The advisor chose $p = (\gamma_I, \lambda_I, \gamma_T, \lambda_T)$ from the set

$$\mathcal{P} \ = \ \left\{ \begin{array}{l} (56, 38, 7, 36), \\ (58, 37, 7, 35), \\ (64, 37, 8, 37), \\ (64, 38, 32, 36) \end{array} \right\}.$$

We also considered running the oracle on $\mathcal{P}$.

Given a parameter choice, alignments were computed using the best methods from prior stages: `MST` trees with normalized costs, exact gap counts, no weighting, and on-the-fly + 3-cut polishing.

Parameter method	`BAliBASE`	`SABmark`	`PALI`	Average
Oracle on $\mathcal{U}$	**87.0**	**54.4**	**87.1**	**76.1**
Oracle on $\mathcal{P}$	86.2	52.9	86.2	75.1
Advisor on $\mathcal{P}$	84.7	50.5	84.9	73.4
Default	84.3	50.2	84.6	73.1

The oracle clearly provides a large boost in recovery, and offers an intriguing target for further research. The improvement of the advisor over the default is modest, and might conceivably be the result of fortuitous choices that exploit the variation in accuracy within set $\mathcal{P}$. A closer look, however, reveals that the advisor's performance is far better than random. For a given subset $\mathcal{S} \subseteq \mathcal{U}$, we can compare the accuracy (averaged over all benchmarks) of the advisor on $\mathcal{S}$ to that of the single best choice from $\mathcal{S}$. Our advisor outperforms the best choice for 60% of all subsets $\mathcal{S} \subseteq \mathcal{U}$ where $|\mathcal{S}| = |\mathcal{P}|$. It also outperforms the mean accuracy of the $x \in \mathcal{S}$ for 94% of all subsets.

In contrast, the method of Yuan *et al.* (1999) outperforms the best choice from $\mathcal{S}$ for only 1% of the subsets, and outperforms the mean of $\mathcal{S}$ for less than 3% of the subsets.

When comparing against other tools in the next section, we consider performance both with and without an advisor.

7 COMPARING TO OTHER TOOLS

The prior sections have examined six stages in the form-and-polish strategy, and identified the best method for each stage. Below, we summarize for all stages the net improvement in alignment quality gained by the best method over a standard method.

The baseline methods in the table are a `UPGMA` merge tree, percent identity for distances, pessimistic gap counts to merge subalignments, unweighted sum-of-pairs, no polishing, and default parameters. We omit the stage of choosing weights, which gave no improvement.

Stage	`BAliBASE`	`SABmark`	`PALI`	Average
(Baseline)	78.0	42.7	80.5	67.1
Tree	+1.4	+1.4	−0.7	+0.7
Distance	+2.2	+4.1	+3.2	+3.1
Merge	+0.8	+0.2	+1.0	+0.7
Polish	+1.9	+1.8	+0.6	+1.5
Parameters	+0.4	+0.3	+0.3	+0.3
(Combined)	**84.7**	**50.5**	**84.9**	**73.4**

This best combination of methods yields a new tool we call `Opal`. In brief, `Opal` uses an `MST` merge tree, normalized costs for distances, exact gap counts to merge subalignments, unweighted sum-of-pairs, and both on-the-fly and 3-cut polishing. A slight boost in accuracy can be gained using the parameter advisor.

In Table 1, we compare the accuracy of `Opal` to other commonly-used tools: `ProbCons` (Do *et al.*, 2005), `MAFFT` (Katoh *et al.*, 2005), `Muscle` (Edgar, 2004), `T-Coffee` (Notredame *et al.*, 2000) and `ClustalW` (Thompson *et al.*, 1994). All tools were run at their highest accuracy. (For `MAFFT` this was their `L-INS-i` variant.) On `BAliBASE` and `PALI`, the second quality measure we report is the percentage of reference columns completely recovered, called the TC score (Bahr *et al.*, 2001).

As these results show, `Opal` is in the class of state-of-the-art tools. Note that `Opal` achieves this without using alignment consistency (Notredame *et al.*, 2000; Do *et al.*, 2005; Katoh *et al.*, 2005), which appears to be the main source of accuracy in `MAFFT` and `ProbCons`.

`Opal` has not yet been optimized for speed. Its running time is about two orders of magnitude slower than `ClustalW` and `Muscle`, and about the same order of magnitude as the slowest

Table 1. Comparison of the accuracy of Opal to commonly-used tools. For each tool and for each suite of benchmarks, the table reports the average accuracy of the tool across all benchmarks in the suite. The last columns report the average accuracy of a tool across all suites. Accuracy is measured by the SPS score, and where applicable, the TC score. The SPS score is the percentage of pairs of aligned positions from the reference alignment that were correctly recovered; the TC score is the percentage of columns from the reference alignment that were completely recovered. The performance of Opal is shown both using an *advisor* method for choosing input-dependent values for gap penalties for scoring alignments, and using input-independent *default* values. The highest accuracies for a suite are in bold

	BAliBASE		SABmark	PALI		Average	
Tool	SPS	TC	SPS	SPS	TC	SPS	TC
MAFFT	**85.1**	**60.4**	49.2	84.3	**60.3**	72.9	**60.4**
ProbCons	84.5	57.9	50.1	84.8	60.0	73.1	59.0
Opal with advisor	84.7	57.9	**50.5**	**84.9**	59.6	**73.4**	58.7
Opal with default parameters	84.3	58.2	50.2	84.6	58.5	73.1	58.4
T-Coffee	80.1	54.3	46.7	81.4	55.0	69.4	54.7
Muscle	80.2	52.1	45.6	81.2	55.5	69.0	53.8
ClustalW	73.2	41.6	44.0	74.5	44.3	63.9	43.0

tool, T-Coffee. Over all benchmarks, the median run time for Opal was less than 10 seconds, which was on an input of 20 sequences of length about 250.

8 CONCLUSION

We have presented a careful study of methods for each stage of the form-and-polish strategy for multiple alignment. This includes new methods for estimating distances, merging alignments, weighting pairs of sequences, polishing the alignment, and choosing parameters. Our new weighting method is easy to implement, fast to evaluate, and avoids the anomalies of current approaches, but under standard measures of benchmark recovery it has little effect on accuracy. A new merging method that optimally aligns alignments yields only a small improvement over an approximate merging heuristic. The largest gains in quality come from new methods for estimating distances by normalized alignment costs, and polishing by 3-cuts on the merge tree; together these two boost recovery by more than 4%. The best method for a stage is generally the same across all suites of benchmarks, suggesting that what we have identified as best has not been overfitted to the data.

The outcome of this study is a new alignment tool, Opal. By combining the best methods, Opal attains accuracy on par with the state-of-the-art (namely ProbCons and MAFFT) without altering the alignment scoring function by increasing gap penalties near hydrophobic regions, or through position-specific substitution scores based on alignment consistency. Adding hydrophobicity and consistency should give even greater accuracy.

8.1 Further research

Incorporating *hydrophobic gap penalties,* and *alignment consistency* into the exact algorithm for aligning alignments might boost the recovery substantially. For example, without hydrophobic penalties, the recovery of Muscle on BAliBASE drops 4%. Similarly, the addition of consistency to MAFFT results in an increase in recovery on both BAliBASE and SABmark of more than 4%.

Properly assessing how weighted sum-of-pairs scoring affects alignment quality by devising an *unbiased recovery measure* that takes overrepresentation into account appears challenging.

Finally, our experiments with an oracle for *choosing parameter values* suggest large gains in recovery may be possible by an input-dependent choice of parameters.

ACKNOWLEDGEMENTS

We thank Jeff Good, David Maddison, Marcie McClure, and Dean Starrett for helpful discussions. We also thank Eagu Kim for assistance in finding initial parameter values by inverse alignment.

T.W. is supported by a PhD Fellowship from the University of Arizona National Science Foundation IGERT Genomics Initiative Grant DGE-0114420. J.K. is supported by the US National Science Foundation through Grant DBI-0317498.

Conflict of Interest: none declared.

REFERENCES

Altschul,S. (1989a) Gap costs for multiple sequence alignment. *J. Theor. Biol.*, **138**, 297–309.

Altschul,S. (1989b) Leaf pairs and tree dissections. *SIAM J. Discrete Math.*, **2**, 293–299.

Altschul,S. *et al.* (1989) Weights for data related by a tree. *J. Mol. Biol.*, **207**, 647–653.

Bahr,A. *et al.* (2001) BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Res.*, **29**, 323–326.

Balaji,S. *et al.* (2001) PALI: a database of alignments and phylogeny of homologous protein structures. *Nucleic Acids Res.*, **29**, 61–65.

Berger,M.P. and Munson,P.J. (1991) A novel randomized iterative strategy for aligning multiple protein sequences. *CABIOS*, **7**, 479–484.

Carrillo,H. and Lipman,D. (1988) The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, **48**, 1073–1082.

Dayhoff,M.O. *et al.* (1978) A model of evolutionary change in proteins. In Dayhoff,M.O. (ed.), *Atlas of Protein Sequence and Structure*, **5**, National Biomedical Research Foundation, Washington DC, pp. 345–352.

Do,C.B. *et al.* (2005) PROBCONS: probabilistic consistency based multiple sequence alignment. *Genome Res.*, **15**, 330–340.

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

Feng,D.F. and Doolittle,R.F. (1987) Progressive alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, **25**, 351–360.

Gotoh,O. (1993) Optimal alignment between groups of sequences and its application to multiple sequence alignment. *CABIOS*, **9**, 361–370.

Gotoh,O. (1994) Further improvement in methods of group-to-group sequence alignment with generalized profile operations. *CABIOS*, **10**, 379–387.

Gotoh,O. (1995) A weighting system and algorithm for aligning many phylogenetically related sequences. *CABIOS*, **11**, 543–551.

Henikoff,S. and Henikoff,J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, **89**, 10915–10919.

Hirosawa,M. *et al.* (1995) Comprehensive study on iterative algorithms of multiple sequence alignment. *CABIOS*, **11**, 13–18.

Katoh,K. *et al.* (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.*, **30**, 3059–3066.

Katoh,K. *et al.* (2005) MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res.*, **33**, 511–518.

Kececioglu,J. and Kim,E. (2006) Simple and fast inverse alignment. In Proceedings of the 10th ACM *Conference on Research in Computational Molecular Biology* (RECOMB), pp. 441–455.

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

Kececioglu,J. and Zhang,W. (1998) Aligning alignments. In Proceedings of the 9th *Symposium on Combinatorial Pattern Matching* (CPM), pp. 189–208.

Kim,E. and Kececioglu,J. (2007) InverseAlign: software for inverse parametric sequence alignment. Version 0.2. http://inversealign.cs.arizona.edu

Kimura,M. (1983). *The Neutral Theory of Molecular Evolution*. Cambridge University Press, Cambridge.

Ma,B. *et al.* (2003) Alignment between two multiple alignments. In Proceedings of the 14th *Symposium on Combinatorial Pattern Matching* (CPM), pp. 254–265.

Murzin,A.G. *et al.* (1995) SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, **247**, 536–540.

Notredame, C. *et al.* (1998) COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, **14**, 407–422.

Notredame,C. *et al.* (2000) T-Coffee: a novel method for multiple sequence alignments. *J. Mol. Biol.*, **302**, 205–217.

Saitou,N. and Nei,M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, **4**, 406–425.

Sneath,P.H.A. and Sokal,R.R. (1973) *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. W.H. Freeman, San Francisco.

Starrett,D. *et al.* (2005). AlignAlign: software for optimally aligning alignments. Version 0.9.7. http://alignalign.cs.arizona.edu

Subbiah,S. and Harrison,S.C. (1989) A method for multiple sequence alignment with gaps. *J. Mol. Biol.*, **209**, 539–548.

Thompson,J.D. *et al.* (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Thompson,J.D. *et al.* (1999) BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, **15**, 87–88.

Wang,L. and Jiang,T. (1994) On the complexity of multiple sequence alignment. *J. Comput. Biol.*, **1**, 337–348.

Van Walle,I. *et al.* (2004) Align-m: a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics*, **20**, 1428–1435.

Vingron,M. and Waterman,M. Sequence alignment and penalty choice: review of concepts, case studies, and implications. *J. Mol. Biol.*, 235, 1–12, 1994.

Yuan,J. *et al.* (1999) MULTICLUSTAL: a systematic method for surveying Clustal W alignment parameters. *Bioinformatics*, **15**, 862–863.