BIOINFORMATICS ORIGINAL PAPER Vol. 21 no. 8 2005, pages 1408–1414
doi:10.1093/bioinformatics/bti159

Sequence analysis

Evaluation of iterative alignment algorithms for multiple alignment

Iain M. Wallace*, Orla O'Sullivan and Desmond G. Higgins

The Conway Institute of Biomolecular and Biomedical Research, University College Dublin, Ireland

Received on October 29, 2004; revised and accepted on November 15, 2004
Advance Access publication November 25, 2004

ABSTRACT

Motivation: Iteration has been used a number of times as an optimization method to produce multiple alignments, either alone or in combination with other methods. Iteration has a great advantage in that it is often very simple both in terms of coding the algorithms and the complexity of the time and memory requirements. In this paper, we systematically test several different iteration strategies by comparing the results on sets of alignment test cases.

Results: We tested three schemes where iteration is used to improve an existing alignment. This was found to be remarkably effective and could induce a significant improvement in the accuracy of alignments from most packages. For example the average accuracy of ClustalW was improved by over 6% on the hardest test cases. Iteration was found to be even more powerful when it was directly incorporated into a progressive alignment scheme. Here, iteration was used to improve subalignments at each step of progressive alignment. The beneficial effects of iteration come, in part, from the ability to get round the usual local minimum problem with progressive alignment. This ability can also be used to help reduce the complexity of T-Coffee, without losing accuracy. Alignments can be generated, using T-Coffee, to align subgroups of sequences, which can then be iteratively improved and merged.

Availability: All of the scripts are freely available on the web at http://www.bioinf.ucd.ie/people/iain/iteration.html

Contact: iain.wallace@ucd.ie

INTRODUCTION

Multiple sequence alignment is an important first step in many bioinformatic applications such as structure prediction, phylogenetic analysis and detection of key functional residues. The accuracy of these methods relies heavily on the quality of the underlying alignment.

Unfortunately the traditional multiple sequence alignment problem is NP-hard, which means that it is impossible to solve for more than a few sequences. In order to align a large number of sequences, many different approaches have been developed. For example, SAGA (Notredame and Higgins, 1996) uses a genetic algorithm to try and optimize a multiple sequence alignment given an objective function. POA (Lee, 2003) builds a multiple alignment using partial order graphs. MSA (Gupta *et al*., 1995) attempts to generate an optimal multiple sequence alignment using a branch and bound technique.

Progressive alignment (Taylor, 1987) is the most widely used heuristic to align a large number of sequences. The multiple alignment is built up progressively by aligning pairs of sequences followed by pairs of alignments/profiles. A guide tree determines the order in which the sequences/alignments are combined with the most closely related being aligned first. This technique is used in many different multiple alignment packages such as MULTALIGN (Barton and Sternberg, 1987), ClustalW (Thompson *et al*., 1994) and T-Coffee (Notredame *et al*., 2000).

One of the problems with progressive alignment is that there is no mechanism to correct mistakes introduced early in the alignment process leading. Iteration was used to get round this problem by Barton and Sternberg (1987) in their MULTALIGN program. Later, PRRP (Gotoh, 1996) used a more highly developed iteration strategy which allowed very accurate alignments. Here, a double iteration loop was used to make the alignment, guide tree and sequence weights mutually consistent. Two of the most accurate multiple alignment packages at the moment, ProbCons (Do *et al*., 2004) and Muscle (Edgar, 2004), refine the final multiple alignment using iteration. ProbCons implements a random partitioning algorithm, while Muscle implements a tree-based partitioning algorithm for the iterations.

Hirosawa *et al*. (1995) investigated the performance of different iteration strategies. They used a group of 30 protein kinase sequences as the basis for their evaluations. They tested how effective each algorithm was at improving the overall Sum of Pairs score for each alignment. We wanted to revisit this important work, as iteration strategies are an effective way of improving the performance of progressive alignment programs. We investigated the most computationally simple iterative algorithms, and benchmarked their performance against the HOMSTRAD database of structure-based alignments (Mizuguchi *et al*., 1998).

SYSTEM AND METHODS

Benchmarking

HOMSTRAD is a collection of high quality structure-based sequence alignments, which can be used as benchmark test cases. The October 2003 release of the HOMSTRAD database was used, containing 1031 alignments of 2–41 sequences each.

Three subsets of the database were created and used as test sets:

HOM184: All alignments that contain four or more sequences. The set contains 184 alignments.

HOM37: All alignments that contain four or more sequences and have less than 25% average identity. This set contains the most demanding cases in HOMSTRAD and contains 37 alignments.

*To whom correspondence should be addressed.

HOM_large: All alignments that contain eight or more sequences. There are 33 alignments in this set.

The quality of an alignment was assessed by comparing it with the reference alignment using the column score (CS) (Karplus and Hu, 2001) function of the aln_compare programme (Notredame *et al*., 2000). The CS calculates the number of identical columns in the reference alignment and the alignment to be tested as a percentage of the number of columns in the reference.

Alignment programs

Five multiple alignment programs were used to generate alignments as inputs for the alignment improvement algorithms. The programs used were T-Coffee (Notredame *et al*., 2000), FFTNSI from the Mafft package (Katoh *et al*., 2002), ClustalW (Thompson *et al*., 1994), Muscle (Edgar, 2004) and ProbCons (Do *et al*., 2004).

ClustalW is the most widely used progressive alignment program. Muscle also uses progressive alignment but with a novel function to align two profiles, called the Log Expectation (LE) scoring function:

$$LE^{xy} = \left(1 - f_G^x\right)\left(1 - f_G^y\right) \log \sum_i \sum_j \frac{f_i^x f_j^y p_{ij}}{p_i p_j}.$$

Here i and j are different amino acid types, p_i is the background probability of i, p_{ij} the joint probability of i and j being aligned. f_i^x is the observed frequency of amino acid type i in column x of the first profile, and f_G^x is the observed frequency of gaps at that column (similarly for column y in profile 2). The factor $\left(1-f_G^x\right)$ encourages more highly occupied columns to align to each other. This LE score is based on the log-average function (van Ohsen and Zimmer, 2001), which has been shown to find more distantly related homologues than the conventional profile alignment function used by ClustalW when used in profile searches.

The FFT-NSI script from the Mafft package was used as it was reported to give the best results of all scripts in the Mafft package on the BAliBASE database (Katoh *et al*., 2002). This script calculates the tree for progressive alignment using a fast Fourier transform (FFT) algorithm. It implements a tree-dependent restricted partitioning alignment improvement algorithm, as well as a normalized similarity matrix.

T-Coffee uses a consistency-based objective function to align sequences along a guide tree. It tries to maximize the score between the final multiple alignment and a library of pairwise local and global alignments. ProbCons also uses a consistency-based function, but it uses a library of pairwise hidden Markov models (HMM) instead. ProbCons is currently the most accurate method as benchmarked on the BAliBASE database (Do *et al*., 2004).

ALGORITHMS

The performance of three iterative alignment improvement algorithms, described by Hirosawa *et al*., were investigated.

Remove first (RF). In each iteration step a sequence is removed from the alignment and realigned to the remaining alignment. If the resulting alignment is better, it is kept and used as input for the next iteration. The iteration cycle is terminated if the alignment score converges, or if a limit of $2N^2$ iterations, where N is the number of sequences, is reached.

Best first (BF). This algorithm tries to reduce the greedy nature of the RF algorithm. In each iteration cycle every sequence is removed and realigned to the rest. The alignment with the best score is kept as input for the next iteration. Again, the iteration cycle is terminated if the alignment score converges or if the limit of $2N^2$ profile–profile alignments is reached. This algorithm is more time-consuming, in general, than the RF algorithm as each iteration contains N profile–profile alignments.

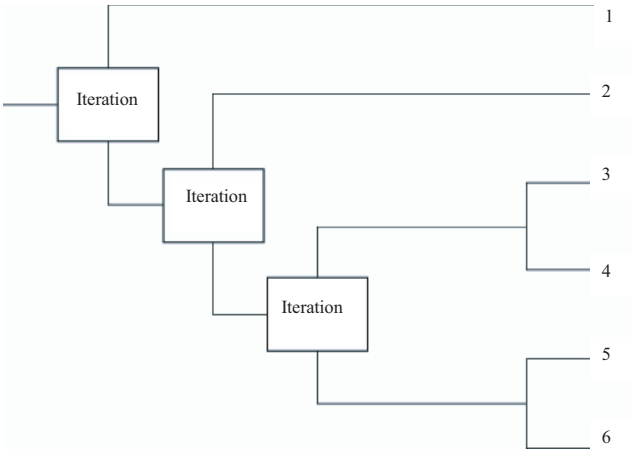

Fig. 1. Schematic of the tree-based iterative algorithm. This is a progressive alignment but incorporating iteration whenever alignments containing more than two sequences are combined. Sequences 3 and 4 are aligned, as are sequences 5 and 6. These two profiles are then aligned. This alignment is then improved using an iterative algorithm. Sequence 2 is then aligned, again followed by an iteration step. Sequence 1 is finally aligned followed by an iteration step.

Random. The alignment is split randomly into two sets of sequences, which are realigned. If the score improves, the resulting alignment is kept. $2N^2$ splits are carried out. This is the most time-consuming algorithm implemented.

These algorithms have time requirements that are at worst $O(N^3)$ as they involve at most $2N^2$ profile alignments each of which is $O(N)$. However in practice the complexity is often much lower.

Tree-based iterative algorithm. The alignment improvement algorithms were also incorporated into a progressive alignment strategy as shown in Figure 1. Every time two profiles are combined, the resulting alignment is refined using one of the iterative algorithms described previously.

Tree-based splitting algorithm. An extension of the tree-based iterative algorithm, called tree-based splitting, was also implemented. The sequences are split into smaller subsets of a predefined maximum size using a tree as in Figure 2. These subsets are aligned using T-Coffee. The resulting alignments are combined using the tree, with an iterative alignment refinement at each step. This algorithm was designed as an attempt at using iteration to reduce the running time of T-Coffee.

A fundamental step in each iteration step is the alignment of two profiles. Two different programs were used to align profiles in this experiment; ClustalW, which maximizes the Average Score, and Muscle, which maximizes the LE score. The performance of each of the algorithms using both the Average Score and the LE score was benchmarked against the HOM184 and HOM37 datasets.

Another alignment improver program, RASCAL (Thompson *et al*., 2003), was also used to improve the default alignments. RASCAL implements a knowledge-based strategy to improve alignments. The alignment is decomposed into reliable and unreliable regions. Only the unreliable regions are modified, trying to maximize the NorMD objective function (Thompson *et al*., 2001), a new objective scoring function for multiple alignment.

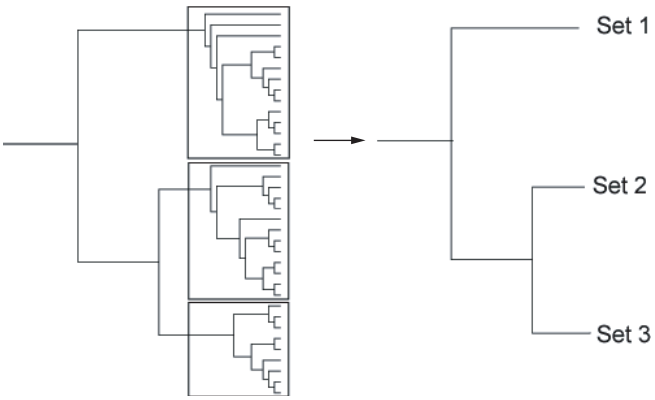

Fig. 2. Tree-based splitting schematic. The full tree on the left is used to create three sets of sequences, which contain no more than half the total number of sequences. These sets are then aligned using T-Coffee, and the resulting alignments are combined using the tree on the right, with an iterative refinement step at each node.

IMPLEMENTATION

Perl was used to implement all of the algorithms. Extensive use of the BioPerl (http://www.bioperl.org) modules was made to produce a series of three scripts.

A generic alignment improver (Iteration.pl). The steps involved in this are:

(1) The alignment is scored using the Sum of Pairs (SP) score by calling SAGA from the command line. SAGA (Notredame and Higgins, 1996) is a multiple alignment package, which uses a genetic algorithm to optimize an objective function. It is possible to make SAGA score an alignment with a command line argument. SAGA can be downloaded from http://igs-server.cnrs-mrs.fr/~cnotred/Projects_home_page/saga_home_page.html

(2) The alignment is then split into two profiles. The split depends on which algorithm is implemented.

(3) When a subset of sequences is removed from an alignment, columns, which only contain gaps, can be formed in either alignment. These are removed from each profile.

(4) An Average score or an LE score is used to align the two profiles. The Average Score algorithm was implemented by calling ClustalW v1.83, while the LE score was calculated by calling Muscle v3.3. Any program that can align two profiles from the command line can be used here.

(5) The new alignment is scored with SAGA, and if it is an improvement on the initial alignment, the new alignment is kept and the process (steps 1–5) continues.

(6) If there is no further improvement in the score, or if the limit of $2N^2$ iterations is reached, the algorithm is terminated.

Splitting sequences based on a tree (TreebasedSplitting.pl). This program takes in a rooted tree, a set of sequences and maximum number of sequences per set. It then splits the sequences into groups no bigger than the maximum number based on the input tree. It outputs a new tree and a new set of sequence files.

Progressive alignment (UseGuideTree.pl). This program takes in the tree output above and aligns the sequences as described by the tree. If there are more than one sequences in a file, they are aligned using T-Coffee. If the maximum number per group is set to 1, this reduces to normal progressive alignment.

RESULTS

Alignment improver

Alignments generated by ProbCons (v1.06), T-Coffee (v1.83) and Mafft (v3.88), using the default settings for each program, were optimized by the three different iteration algorithms. The quicktree option was used with ClustalW, as well as the default setting. The results for this are labelled 'ClustalW–Quicktree'. When ClustalW is run using the default parameters, the guide tree that is used for the progressive alignment is generated using dynamic programming algorithm for pairwise alignment followed by the Neighbour Joining method of Saitou and Nei (1987). When the quicktree option is used, the guide tree is built using a much faster algorithm based on ktuples (Wilbur and Lipman, 1983).

Two versions of the Muscle program were also used to generate input alignments, v3.2 and v3.3. The difference between the two versions is that the latter uses a tree-based partitioning algorithm to improve the final alignment. This iterative refinement step scales as $O(N^3)$ (Edgar, 2004).

The results for HOM184 are shown in Table 1. The numbers are the average CS for the entire dataset. The best result for each method is highlighted in bold, and the best percentage improvement in the CS is shown in the right-hand column. The final row shows the average CS for each alignment strategy over all methods. The significance of the results was determined using the Wilcoxon Signed Rank test as implemented in SPSS version 12.

Both the RF and BF algorithms using the LE scoring method give the best overall performance. There is very little difference between them for any method. They significantly improve the results for Muscle v3.3 (63.8 versus 63.1%), Muscle v3.2 (63.6 versus 61.8%), ClustalW (61.5 versus 59.9%), ClustalW-Quicktree (60.9 versus 59.3%) and FFTNSI (62.1 versus 59.7%). It is interesting to note that when either the BF or RF algorithms refine the Muscle v3.2 alignment, a slightly higher score is achieved than by using Muscle v3.3, which includes a tree-based iteration strategy (63.6 versus 63.1%).

The LE scoring system nearly always outperforms the Average Scoring system, irrespective of the refinement algorithm. However, the Average Scoring system is still able to improve some methods, such as ClustalW (61.5 versus 59.9%), Muscle v3.2 (62.6 versus 61.8%) and FFTNSI (61.4 versus 59.7%). The same overall performance (the average of all the methods) as the knowledge-based method RASCAL is achieved.

However, neither scoring system method is able to improve either T-Coffee or ProbCons significantly. This is presumably because the SP is not a perfect biological objective function, and improving the score can actually make the alignment worse. The consistency-based objective functions used by both T-Coffee and ProbCons tend to provide better results than the SP objective function. Surprisingly when the iteration step was removed from ProbCons, there was no degradation in performance implying that the consistency-based objective function is indeed the important part in the program.

Table 1. Results from HOM184 for each of the iterative algorithms using both the Average Score, and LE to carry out the profile alignment and the RASCAL alignment improver

Method	Default	RASCAL	Average score			Log expectation			Summary	
			RF	BF	Random	RF	BF	Random	Best score (%)	Improvement (%)
ProbCons	64.88	64.99	63.64*	63.46**	62.73*	64.69	**65.00**	64.27*	**65.00**	0.18
Muscle v3.3	63.12	62.96	62.84	62.72	62.56	**63.77*****	63.70**	63.39	**63.77*****	1.03
T-Coffee	62.87	62.86	62.40	62.11	62.83	63.24	**63.38**	62.70	**63.38**	0.81
Muscle v3.2	61.76	62.31	62.62*	62.57*	61.52	**63.58*****	63.57***	62.75**	**63.58*****	2.94
ClustalW	59.87	60.47	60.90***	61.08***	61.46***	**61.54*****	61.44***	60.99**	**61.54*****	2.80
FFT-NSI (Mafft)	59.65	60.92	61.23*	61.27*	61.42**	**62.10*****	62.05***	61.55***	**62.10*****	4.10
ClustalW–Quicktree	59.32	59.36	59.93*	60.16**	60.20**	60.70***	**60.88*****	60.57***	**60.88*****	2.63
Average	61.64	61.98	61.94	61.91	61.82	62.80	**62.86**	62.32		

The values are all percentage columns correct (CS). Significant differences are marked: $^{*}p \leq 0.05$; $^{**}p \leq 0.01$; $^{***}p \leq 0.001$. The highest value for each row is highlighted in bold.

Table 2. Results from HOM37 for each of the alignment improver algorithms

Method	Default	RASCAL	Average score			Log expectation			Summary	
			RF	BF	Random	RF	BF	Random	Best score (%)	Improvement (%)
ProbCons	**41.84**	41.65	38.01**	37.31**	37.73**	39.60*	41.33	39.86**	**41.84**	0
Muscle v3.3	38.46	37.46	37.96	38.21	37.21	**39.42**	39.08	38.26	**39.42**	2.49
T-Coffee	37.20	37.08	35.61**	34.15***	36.05	37.69	**38.64**	37.73	**38.64**	3.87
Muscle v3.2	35.88	35.58	37.35	36.94	35.20	**38.61****	38.58*	38.14	**38.61****	7.61
ClustalW	32.52	33.19	32.90	33.60	33.72	**34.62*****	34.55**	34.22*	**34.62*****	6.47
FFT-NSI (Mafft)	32.43	34.25	34.09	34.19	34.15	**36.83***	36.41*	36.45	**36.83***	13.56
ClustalW–Quicktree	29.88	31.13	31.00	31.88	32.64	32.47*	**32.71***	32.20	**32.71***	9.47
Average	35.46	35.76	35.50	35.18	35.24	37.04	**37.33**	36.69		

Details are as in Table 1.

Table 2 shows the results on the HOM37 test set. The difference between the alignment programs and alignment improvement strategies is more pronounced on this dataset. The order of the performance of the programs is the same as in Table 1. Also, RASCAL performs better on average than all of the algorithms that use the Average Score profile method.

The relative order of algorithmic performance is BF $\cong$ RF $>$ Random Partitions. Interestingly the Random Partitions algorithm performs worst even though it is the most computationally intensive method. While the alignment improver algorithms do improve the quality of most of the alignments, they do not cause a big change in the ranking of the methods. Only the rank of FFT-NSI is improved. ProbCons is still the best method overall even after improvement of other methods.

FFTNSI is the method that is improved the most in both sets of results (4.1% in Table 1 and 13.56% in Table 2). The default rank for FFTNSI was surprisingly low (it ranked below ClustalW) as FFTNSI was previously reported to perform as well as T-Coffee on the BAliBASE benchmark (Katoh *et al*., 2002). If FFTNSI was tuned to the BAliBASE database, this would explain the low rank on this dataset, and also the large relative improvement in the alignments due to the iteration. We would like to point out that a new version of Mafft (version 4.0) has been released and initial testing shows it has much improved accuracy than the version evaluated in this study.

An important consideration in evaluating the usefulness of iteration algorithms is the number of iterations required for convergence. The average number of iterations for each test set was plotted against the number of sequences in the alignment. The RF algorithm always required fewer iterations than the BF even though a similar level of accuracy was obtained. The data points were fitted using the TableCurve package, and the trend lines are shown in Figure 3. The random partitioning algorithm always requires $2N^2$ iterations and as such was best fitted by line with an $O(N^2)$ equation. The BF algorithm was fitted with a line that was $O(N^{1.5})$, and the RF with a line that was $O(N)$. These graphs show the number of iterations. As each iteration consists of a profile alignment which is $O(N)$, the random partitioning is therefore always $O(N^3)$ overall. The remove first algorithm is approximately $O(N^2)$ overall, and the best first algorithm is in between.

Tree-based iteration

An alternative method of using iteration for multiple sequence alignment is to incorporate it into a progressive alignment strategy. This is shown in Figure 1. At each node in the guide tree the alignment is improved using an iterative algorithm.

Each of the algorithms were incorporated into a progressive alignment and benchmarked on the HOM184 and HOM37 datasets. The guide tree output by ClustalW was used as the input tree for the UseGuideTree script. Two different guide trees were generated for

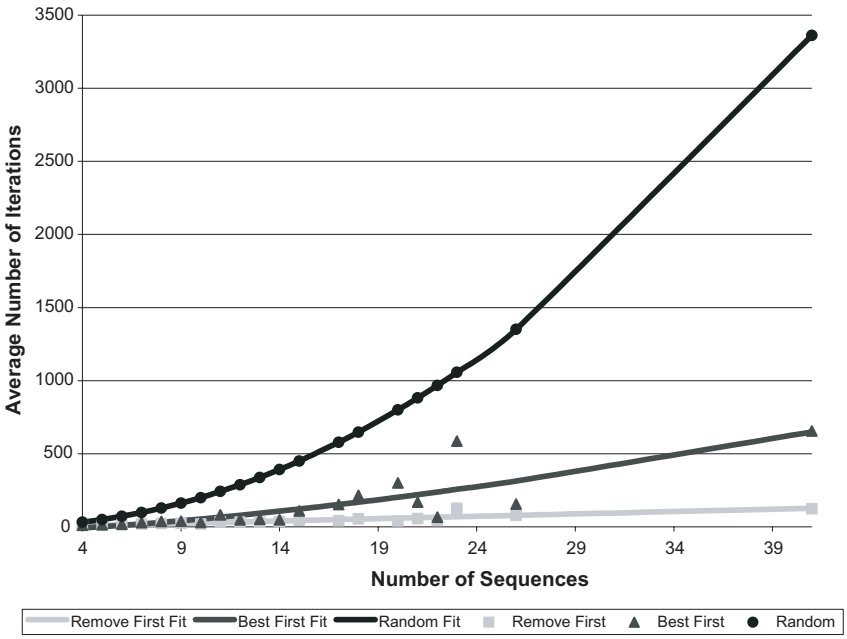

Fig. 3. A graph of the average number of iterations each algorithm requires before it converges, plotted against the number of sequences in each alignment in HOM184.

Table 3. Results for the tree-based iterative algorithm for the HOM37 and HOM184 datasets

	Average score			Log expectation			
	RF	BF	Random	RF	BF	Random	Best score
HOM37							
Quick tree	34.61*	36.35***	34.65*	39.40*	**39.89***	37.48	**39.89**
Slow tree	34.03	34.62	32.99	38.62	**39.05**	35.66	**39.05**
HOM184							
Quick tree	61.58**	61.93***	61.42**	63.45**	**63.69*****	62.47**	**63.69**
Slow tree	61.53**	61.70**	61.06	63.10*	**63.27****	61.74	**63.27**

Two trees were generated by ClustalW, the default tree 'Slow Tree', and the quick tree. The significance has been calculated for the Average score quick tree versus ClustalW–quicktree; Average Score slow tree versus ClustalW–slowtree; LE versus Muscle v3.2. The highest value for each row is highlighted in bold.

each alignment, the default ClustalW tree that is created using full dynamic programming, and the ClustalW '–quicktree' option. The results are shown in Table 3. The significance of the results was determined by using the Wilcoxon rank-sign test.

The BF algorithm performs best for both test sets on both trees followed by the RF algorithm which performs slightly worse. Surprisingly, the quick tree provides the best results, as it is computationally much simpler to calculate. It should be noted, however, that the default settings of ClustalW perform better than the ClustalW quicktree option, when run without iterative improvement.

When the BF algorithm with the Average Score is incorporated into a progressive alignment, it outperforms the BF algorithm used as an alignment improver, on the hardest dataset. ClustalW is improved from 32.5 to 33.6% when BF is used as an alignment improver, and is further improved to 34.6% when it is incorporated into a tree. A similar effect is also observed for ClustalW '–quicktree' (29.9 to 31.9 to 36.4%) and Muscle v3.2 with the LE function (35.9 to 36.9 and finally up to 39.9%). The BF algorithm and the LE function achieve a better score than Muscle v.3.3 (39.9 versus 38.5%) on the HOM37 dataset.

Tree-based splitting

T-Coffee cannot easily handle much more than 100 sequences of average length. In an effort to reduce the running time and computational expense of T-Coffee, the tree-based splitting algorithm was developed. A set of sequences to be aligned is split into smaller sets, which are aligned by T-Coffee. The resulting alignments are combined using progressive alignment, as described in the previous section.

The performance of this algorithm was tested using the HOM_large dataset. The sequences were split using a tree generated by ClustalW using the '-quicktree' option. This tree was chosen as it is very quick to compute, and gave good performance in the previous section.

Table 4. Results for the tree-based splitting algorithm

Number of sequences	Average score		Log expectation score	
	Best-first	No iteration	Best-first	No iteration
1	49.48	45.74	50.55	46.53
1/8 *N*	49.52	45.90	50.60	46.90
1/4 *N*	49.34	47.03	50.89	47.42
1/2 *N*	48.67	47.72	50.27	46.62
Default T-Coffee	48.91	48.91	48.91	48.91

A set of sequences is split into sub-sets. These sub-sets are aligned using T-Coffee, and then combined. *N* is the total number of sequences in the original set, and the maximum number of sequences in each sub-set is shown on the left.

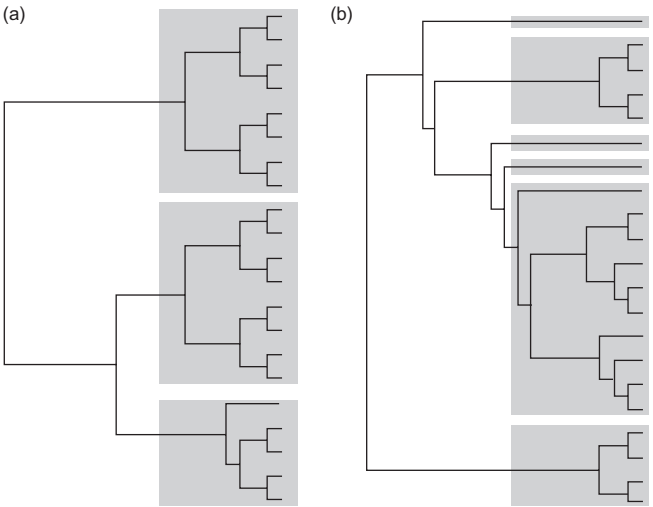

Fig. 4. Tree (**a**) is the balanced tree and (**b**) is a tree created by the UPGMA algorithm. The maximum number of sequences in each set is $N/2$ or 10. The balanced tree has three similar sized groups, which will be aligned by T-Coffee, compared with the UPGMA tree, which has one very large group and many much smaller groups. The balance tree maximizes the amount of information used by T-Coffee to generate better alignments.

T-Coffee scores 48.9% columns correct on this HOM_large dataset. The results for the tree-based splitting algorithm are shown in Table 4. The importance of the iteration step is clear; without the iteration step all of the splits perform worse than the default T-Coffee, as would be expected. However, when the iteration step is used the splitting algorithm improves on the default T-Coffee result. There is an average difference of 3% between the results with iteration versus the results without.

It was observed with many datasets that when the tree-based splitting algorithm was used, one relatively large set of sequences, which contained the maximum number of sequences, and many small sets of sequences were created. This is because the trees created by ClustalW try to recreate the phylogenetic relationships between the sequences, and are not always balanced (Fig. 4b). In order to create a balanced tree, a modified version of the UPGMA algorithm was used. The Protdist programme in the Phylip package was used to create a distance matrix from a ClustalW–quicktree alignment. The two sequences with the smallest distances are grouped together.

Table 5. Results for the tree-based splitting algorithm using the balanced tree

Number of sequences	Average score		Log expectation score	
	BF	No iteration	BF	No iteration
1	50.39	43.07	49.65	42.60
1/8 *N*	**50.65**	42.77	50.42	43.29
1/4 *N*	50.00	43.25	**51.43**	43.70
1/2 *N*	50.62	45.44	49.47	46.20
Default T-Coffee	48.91	48.91	48.91	48.91

A set of sequences is split into sub-sets. These sub-sets are aligned using T-Coffee, and then combined. *N* is the total number of sequences in the original set, and the maximum number of sequences in each sub-set is shown on the left. The highest value for each scoring function is highlighted in bold.

Then the next two most related sequences are grouped together, and so on until all of the sequences have been paired off. This process is then repeated except the closest groups, instead of closest sequences, are paired off with each other, until there is only one group left. An example of the balanced tree is shown in Figure 4a.

Balanced trees as described above were used as input for the tree-based splitting algorithm and the results are shown in Table 5. Without the iteration step, the balanced tree performs worse than the trees generated by ClustalW (Table 4) for all of the different split sizes. The average disimprovement is 2.9%. However when the iteration step is included, the results improve noticeably. The average improvement caused by the iteration is 6.5% versus no iteration for the balanced trees.

Using 1/4 *N* split and the LE score (51.4%), the tree-based splitting algorithm outperforms both the default T-Coffee (48.9%) and the tree-based algorithm (50.6%). This split would reduce the complexity of T-Coffee by a factor of 16.

DISCUSSION

Iteration is a very simple, fast and effective way to improve multiple alignment methods. Iteration can be used to improve existing software with any objective function. It can also be incorporated into a progressive alignment strategy to build alignments from scratch to produce even better results.

We found that the LE function resulted in better alignments than the more conventional Average function. Using the LE function in conjunction with either the BF or RF algorithms, alignments from a variety of different multiple alignment packages were improved. Specifically, when the Muscle v3.2 alignment was refined with the RF algorithm, similar results were obtained for Muscle v.3.3, which has an $O(N^3)$ iteration step. We found the RF algorithm, on average, had a complexity of $O(N^2)$. We also attempted to use mixtures and combinations of the iteration strategies but found no significant improvement.

The performance of progressive alignment algorithms can be improved by including iteration steps during the progressive alignment algorithm. We found the performance of ClustalW with the quicktree option improved from 29.9 to 36.4% on the hardest test cases. Similarly the performance of Muscle v3.2 could be improved from 35.9 to 39.9%. This is also better than Muscle v3.3, which includes an iteration step.

An iteration algorithm also makes it possible to align a larger number of sequences with T-Coffee. The large set of sequences is split into a few smaller sets of sequences, which are aligned by T-Coffee. Using a progressive alignment algorithm the small alignments are combined into the final alignment. With the iteration step the alignment quality is comparable to that of T-Coffee default. If a large set of sequences is split into four equal sized smaller sets, the time spent in T-Coffee is reduced by a factor of 16.

In a recent paper (Ebedes and Datta, 2004), the progressive alignment step was found to be a limiting step in parallelizing ClustalW. This was mainly due to unbalanced guide trees. The balanced tree described above could be used to overcome this problem if an iteration step is included.

ACKNOWLEDGEMENTS

We are especially grateful to Manolo Gouy for a C program that roots an unrooted tree. This work was funded by the Science Foundation Ireland. The authors would like to thank the referees for their helpful comments.

REFERENCES

Barton,G.J. and Sternberg,M.J. (1987) A strategy for the rapid multiple alignment of protein sequences. Confidence levels from tertiary structure comparisons. *J. Mol. Biol.*, **198**, 327–337.

Do,C.B., Brudno,M. and Batzoglou,S. (2004) PROBCONS: probabilistic consistency-based multiple alignment of amino acid sequences. Abstract in the Nineteenth National Conference on Artificial Intelligence AAAI, p. 703.

Ebedes,J. and Datta,A. (2004) Multiple sequence alignment in parallel on a workstation cluster. *Bioinformatics*, **20**, 1193–1195.

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

Gotoh,O. (1996) Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J. Mol. Biol.*, **264**, 823–838.

Gupta,S.K., Kececioglu,J.D. and Schaffer,A.A. (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comput. Biol.*, **2**, 459–472.

Hirosawa,M., Totoki,Y., Hoshida,M. and Ishikawa,M. (1995) Comprehensive study on iterative algorithms of multiple sequence alignment. *CABIOS*, **11**, 13–18.

Karplus,K. and Hu,B. (2001) Evaluation of protein multiple alignments by SAM-T99 using the BAliBASE multiple alignment test set. *Bioinformatics*, **17**, 713–720.

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

Lee,C. (2003) Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics*, **19**, 999–1008.

Mizuguchi,K., Deane,C.M. Blundell,T.L. and Overington,J.P. (1998) HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Sci.*, **7**, 2469–2471.

Notredame,C. and Higgins,D.G. (1996) SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Res.*, **24**, 1515–1524.

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

Taylor,W.R. (1987) Multiple sequence alignment by a pairwise algorithm. *Comput. Appl. Biosci.*, **3**, 81–87.

Thompson,J.D., Higgins,D.G. and Gibson,T.J. (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., Plewniak,F., Ripp,R., Thierry,J.C. and Poch,O. (2001) Towards a reliable objective function for multiple sequence alignments. *J. Mol. Biol.*, **314**, 937–951.

Thompson,J.D., Thierry,J.C. and Poch,O. (2003) RASCAL: rapid scanning and correction of multiple sequence alignments. *Bioinformatics*, **19**, 1155–1161.

van Ohsen,N. and Zimmer,R. (2001) Improving profile-profile alignments via log average scoring. *Proc. WABI*, **2149**, 11–26.

Wilbur,W.J. and Lipman,D.J. (1983) Rapid similarity searches of nucleic acid and protein data banks. *Proc. Natl Acad. Sci. USA*, **80**, 726–730.