Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems

Catherine Grasso and Christopher Lee*

Department of Chemistry and Biochemistry, Molecular Biology Institute, Center for Genomics and Proteomics, University of California, Los Angeles, CA 90095-1570, USA

Received on May 25, 2003; revised on January 5, 2004; accepted on January 6, 2004

Advance Access publication February 12, 2004

ABSTRACT
Motivation: Partial order alignment (POA) has been proposed as a new approach to multiple sequence alignment (MSA), which can be combined with existing methods such as progressive alignment. This is important for addressing problems both in the original version of POA (such as order sensitivity) and in standard progressive alignment programs (such as information loss in complex alignments, especially surrounding gap regions).

Results: We have developed a new Partial Order–Partial Order alignment algorithm that optimally aligns a pair of MSAs and which therefore can be applied directly to progressive alignment methods such as CLUSTAL. Using this algorithm, we show the combined Progressive POA alignment method yields results comparable with the best available MSA programs (CLUSTALW, DIALIGN2, T-COFFEE) but is far faster. For example, depending on the level of sequence similarity, aligning 1000 sequences, each 500 amino acids long, took 15 min (at 90% average identity) to 44 min (at 30% identity) on a standard PC. For large alignments, Progressive POA was 10–30 times faster than the fastest of the three previous methods (CLUSTALW). These data suggest that POA-based methods can scale to much larger alignment problems than possible for previous methods.

Availability: The POA source code is available at http://www.bioinformatics.ucla.edu/poa
Contact: leec@mbi.ucla.edu

INTRODUCTION
Partial order alignment (POA) has been proposed as a new approach to the challenges of multiple sequence alignment (MSA) (Lee et al., 2002) and was recently validated by independent testing (Lassmann and Sonnhammer, 2002). The basic idea behind this approach consists of two modifications to existing alignment methods: first, the linear representation of an MSA is replaced by a directed acyclic graph (DAG) representation, called a partial order MSA (PO-MSA); second, standard pairwise dynamic programming alignment is extended to yield a method for aligning a single sequence to a PO-MSA. These modifications enabled the development of a simple iterative MSA procedure, presented as the original version of POA, in which sequences are aligned one-by-one to a growing PO-MSA structure consisting of all the sequences aligned previously (Lee et al., 2002).

The original POA algorithm has some interesting benefits. During alignment of a sequence to a PO-MSA, the dynamic programming algorithm considers all possible alignments of the sequence to not only all the sequences already stored in the PO-MSA but also all possible homologous recombinations of these sequences (effectively, introducing homologous recombination as an evolutionary edit operator). In contrast, standard profile-based methods force the sequence to align to an ‘averaged sequence’ (profile) representation of the MSA. As a consequence, POA is less susceptible to the MSA local minimum problem described previously (Thompson et al., 1994), both because the PO-MSA representation handles gaps more naturally and because it retains (and uses) all the information in the MSA (unlike linear profiles). Additionally, since the original version of POA merges any pair of aligned nodes representing identical residues, when there is a significant redundancy in the input sequences, the PO-MSA representation effectively compresses the MSA data, resulting in impressive alignment speed increases.

However, the original version of POA (which we will refer to throughout this paper as Iterative POA) has significant weaknesses, suggesting that it could be improved considerably. One shortcoming, which we address in this paper, is that Iterative POA provided only a local alignment, not a global alignment algorithm. A more fundamental shortcoming is that Iterative POA is sensitive to the order in which...
Combining partial order alignment and progressive multiple sequence alignment

Fig. 1. Progressive partial order alignment algorithm. The alignment of eight hypothetical sequences is illustrated. The guide tree is shown in black and indicates the order in which the sequences are aligned. The progressive alignment algorithm proceeds from the leaves of the rooted tree to the root. Sequences are shown as colored rectangular segments. Only regions of aligned sequence are stacked on top of each other. To ensure this, thin lines representing directed edges in the PO-MSA are included so that regions that are unaligned are separate in space. This visual representation of a PO-MSA is consistent with the one drawn by the POAVIZ program (Grasso et al., 2003).

sequences are aligned and makes no attempt to optimize the alignment ordering. Iterative POA aligns a set of sequences in the exact order in which they are supplied. In contrast, progressive MSA algorithms (Feng and Doolittle, 1987), such as CLUSTAL (Higgins and Sharp, 1988), have long employed a heuristic for determining a preferred alignment ordering. These algorithms construct a guide tree to determine the alignment ordering, by applying the neighbor-joining method to the matrix of pairwise distances among all the sequences (Saitou and Nei, 1987). Progressive MSA is then performed in the order dictated by the guide tree so that sequences that are more similar are aligned before those that are more distant from each other. This suggests that POA could be improved by extending it to align sequences progressively in the order dictated by a guide tree (we will refer to this throughout the paper as the Progressive POA algorithm).

POA suggests that we can reformulate the basic operation of progressive alignment. In principle, each stage of progressive alignment involves aligning two alignments (each a cluster of sequences that have already been aligned). However, standard dynamic programming (Needleman and Wunsch, 1970; Smith and Waterman, 1981) can align a pair of sequences, not a pair of alignments. In practice, progressive MSA algorithms have simply reduced both alignments to linear profile sequences, which can then be aligned using standard dynamic programming (Higgins and Sharp, 1988; Thompson et al., 1994). However, reducing an alignment to a linear profile causes significant information loss and potential for alignment artifacts (Lee et al., 2002). Because a PO-MSA can fully represent an alignment without information loss, and because dynamic programming can be applied directly to PO-MSAs, it is possible to extend POA to compute directly an optimal alignment of a pair of alignments. In this paper, we generalize our earlier dynamic programming algorithm to perform alignment of two PO-MSAs. In addition, we have implemented a combination of this algorithm with progressive MSA methods. Finally, we provide extensive test results comparing the Progressive POA method with the original Iterative POA method and to other MSA methods, such as CLUSTALW, DIALIGN2, T-COFFEE and MAFFT.

ALGORITHM
Input for Progressive POA

The Progressive POA algorithm takes as input both a set of sequences in FASTA format, and the set of pairwise similarity scores between all of these sequences. Progressive POA makes no assumptions about the source of these pairwise similarity scores. In this paper, we used BLAST bit scores since they can be calculated rapidly; however, any rapid pairwise sequence homology method that returns similarity scores could be used.

Overview of the Progressive POA algorithm

The Progressive POA algorithm has three main components: guide tree construction; progressive alignment following the guide tree; and the dynamic programming algorithm to align two PO-MSAs (we will refer to this as the PO–PO alignment
algorithm). An application of the Progressive POA algorithm to a set of eight sequences is shown in Figure 1 in order to illustrate these components graphically. Overall, the construction of the MSA follows the structure of the guide tree, which resembles a phylogenetic tree in that the sequences that are most similar to each other, and which will therefore be aligned earlier, are placed on adjacent branches of the tree. The Progressive POA alignment method differs from existing methods only in that the MSAs stored at the branch points are not reduced to linear profile sequences, as would be required for standard dynamic programming. Instead, the MSAs at the internal branch points (nodes 9–15 in Fig. 1) are stored as PO-MSAs, which are aligned to each other using the PO–PO alignment algorithm.

Briefly, the guide tree is constructed via the application of agglomerative nearest-neighbor clustering [Kruskal’s Minimum Spanning Tree algorithm; Cormen et al. (1997)] to the matrix of pairwise similarity scores between sequences. Here, we treat sequences with high similarity scores as less distant and low similarity scores as more distant.

The PO-MSA data structure

We previously described the PO-MSA representation of a MSA as a DAG (Lee et al., 2002). We can represent a MSA (Fig. 2A) as a partially ordered graph (Fig. 2B) in which individual sequence letters are represented by nodes, directed edges are drawn between consecutive letters in each sequence, aligned residues are recorded as being aligned to each other, identical aligned residues are fused into a single node, and redundant edges are pruned. When letters are fused as one node, the resulting node stores information about all the individual sequence letters from which it was derived, specifically the ID of the original sequence(s) and the index of the letter’s position within that sequence.

The PO–PO alignment algorithm

We have presented an algorithm for dynamic programming alignment of a sequence to a PO-MSA graph (Lee et al., 2002). Here, we will describe the changes required to extend this algorithm to an algorithm for aligning a PO-MSA graph $G$ to a PO-MSA graph $G'$. This algorithm, which finds the optimal alignment over the set of all alignments of any path $P$ in $G$ to any path $P'$ in $G'$, can be accomplished by an extension of the standard dynamic programming alignment algorithm. This involves calculating the optimal score, $S(n, o)$, and alignment move for every possible pairing of a node $n$ in $G$ and a node $o$ in $G'$. The possible moves include aligning the two nodes [with substitution score $s(n, o)$] and deletion/insertion [omitting node $n$ from the alignment, with score $\Delta(n)$; or omitting node $o$ with score $\Delta(o)$]. If $G$ consists of a single sequence, every node $n$ would have at most one predecessor node $p$. But in general, since both $G$ and $G'$ are PO-MSAs, there can be multiple predecessors for both $n$ in $G$ and $o$ in $G'$, and we must consider all possible combinations of predecessor nodes to find the move with the highest score,

$$S(n, o) = \max_{p \rightarrow n, q \rightarrow o} \begin{cases} S(p, q) + s(n, o) \\ S(p, o) + \Delta(n) \\ S(n, q) + \Delta(o) \end{cases}$$

over all predecessor nodes $p$ in $G$ that have a directed edge from $p \rightarrow n$, and all predecessor nodes $q$ in $G'$ that have a directed edge from $q \rightarrow o$.

Other aspects of the algorithm, such as the details of implementing Smith–Waterman dynamic programming and trace-back of the best alignment, are identical with our previous description (Lee et al., 2002).
Global PO–PO alignment scoring

We also implemented global alignment scoring as described previously (Needleman and Wunsch, 1970), in contrast to the local alignment scoring system (Smith and Waterman, 1981) used in the original version of POA (Lee et al., 2002). Since there can be more than one sequence in a given PO-MSA, our algorithm considers correctly all combinations of the possible sequence starts in \( G \) versus all possible sequence starts in \( G' \), as well as all possible sequence ends in \( G \) versus all possible sequence ends in \( G' \), to find the optimal global alignment path.

Truncated gap penalties

In order to favor alignments with a few large gaps rather than many small gaps throughout the alignment, it is standard practice to vary the gap penalty, \( \Delta \), so that it is higher for inserting the first gap than for extending a gap. In order to allow for large terminal extensions and internal insertions in the optimal global alignment of a path \( P \) in PO-MSA \( G \) and path \( P' \) in PO-MSA \( G' \), we have implemented a truncated gap penalty, \( \Delta_t \), in addition to the standard gap opening penalty, \( \Delta_o \), and gap extension penalty, \( \Delta_e \). This requires three new parameters: a gap truncation penalty, \( \Delta_t \), a gap truncation length and a gap decay length. After a number of gaps equal to the gap truncation length have been inserted into the alignment, the gap penalty decays linearly, over the gap decay length, from \( \Delta_e \) to \( \Delta_t \). These parameters, which can be tuned by the user, have been included to allow for the alignment of multi-domain proteins that do not align over their complete length. A rigorous implementation of non-affine gap penalties requires storing and considering an optimal score, \( S \), for every possible value of the gap penalty, \( \Delta \), in every cell of the dynamic programming matrix, leading to a large increase in the computational space and time complexity of the algorithm. The current implementation of Progressive POA compromises optimality for speed, varying \( \Delta \) according to a non-affine function but storing and considering only one optimal score, \( S \), in each cell of the matrix.

IMPLEMENTATION

This algorithm has been implemented as a C program, utilizing the POA library described previously (Lee et al., 2002). It is included in the existing POA program as a command-line option. The source code compiles and runs on a wide variety of UNIX and Windows platforms and can be downloaded from http://www.bioinformatics.ucla.edu/poa

SYSTEM AND METHODS

The results for the BAliBASE testing, as well as the results for the ROSE alignments of increasing length and the ROSE alignments with increasing numbers of sequences, were generated on a 1.4 GHz AMD Athlon running Linux.

Alignment programs and parameter settings

CLUSTALW (Thompson et al., 1994), DIALIGN2 (Morgenstern, 1999), T-COFFEE (Notredame et al., 2000) and MAFFT (Katoh et al., 2002) were run using the default parameters. MAFFT was run using the progressive method, FFT-NS-2. Iterative POA Local (Lee et al., 2002), the original version of POA, was run using the following default parameters: BLOSUM80 scoring matrix, gap opening penalty of 12, gap extension penalty of 6 and gap truncation penalty of 6 (i.e. the gap penalty is not truncated). Progressive POA Global (Progressive POA run with the ‘-global’ command line argument) was run with the following parameters: BLOSUM80 scoring matrix, gap opening penalty of 12, gap extension penalty of 6, truncated gap penalty of 0, gap truncation length of 15 and gap decay length of 5. Iterative POA Global was also run with these parameters.

Reference alignments

To test MSA quality, we used the BAliBASE 2.0 benchmark test set (Thompson et al., 1999a; Bahri et al., 2001). BAliBASE 2.0 is divided into eight reference sets, of which we used only the first five: reference 1 contains alignments of a small number of equidistant sequences; reference 2 contains alignments of a family (sequences with >25% ID) along with up to three ‘orphan’ sequences (sequences with <20% ID to the family); reference 3 contains alignments of up to four families such that there is <25% ID between sequences in different families; reference 4 contains alignments with sequences with large N/C-terminal extensions; reference 5 contains alignments with large internal insertions. Reference 1 is subdivided further to reflect the degree of homology between the sequences in the alignments: category v1 contains alignments with <25% identity between the sequences; category v2 contains alignments with 20–40% identity between the sequences; category v3 contains alignments with >35% identity between the sequences.

In order to test the speed of Progressive POA Global relative to Iterative POA Local, Iterative POA Global, CLUSTALW, DIALIGN2, T-COFFEE and MAFFT, we ran all seven programs on sets of sequences generated using ROSE (Stoye et al., 1998). ROSE randomly generates a phylogenetic tree according to the user’s specifications and then simulates sequence evolution according to the tree to generate a family of related sequences along with their ‘true’ alignment. ROSE was run with default values for all parameters except for sequence length, sequence number and sequence relatedness. To generate sets of sequences of increasing length along with their ‘true’ alignments, ROSE was run with the sequence number set to 50, the sequence relatedness set to 150 (i.e. there is an average of 30% identity between the sequences) and the sequence length varying from 100 to 2000 in steps of 100. To generate sets of increasing numbers of sequences and varying levels of sequence identity along with their alignments, ROSE was run with the sequence length set to 500, the sequence

1549
relatedness set to 150 (i.e. there is an average of 30% identity between the sequences), 56 (i.e. there is an average of 60% identity between the sequences) or 11 (i.e. there is an average of 90% identity between the sequences) and the sequence number varying from 100 to 1000 in steps of 100. These data were generated independently five times for each of the three sequence relatedness settings. All the results given below on the ROSE test set were averaged over these five independent trials.

Alignment evaluation

The BAliSCORE program was run to calculate the sum-of-pairs score (SPS), which is the percentage of residue pairs aligned correctly, and the column score (CS), which is the percentage of columns aligned correctly, in the new alignment relative to the 'true' reference alignment (Thompson et al., 1999b). For the BAliBASE alignments, the SPS and the CS scores are calculated only using residues in the annotated core blocks. Consistent with the analysis in Thompson et al. (1999b), the average SPSs are provided for references 1 and 2, while the average CSs are provided for references 3, 4 and 5. For the ROSE alignments, the SPS and CS are calculated using all residues.

RESULTS

BAliBASE testing of alignment accuracy

To assess the quality of the alignments generated by Progressive POA Global relative to those generated by Iterative POA Local, Iterative POA Global, CLUSTALW, DIALIGN2 and T-COFFEE, we ran all six programs on reference sets 1–5 of the BAliBASE 2.0 benchmark test set (Thompson et al., 1999a; Bahr et al., 2001). The alignments in BAliBASE 2.0 are alignments of real protein sequences that have been manually refined and verified by superposition of all known three-dimensional protein structures and that have had their core blocks annotated using protein structure data. The results of running Progressive POA Global, Iterative POA Global, Iterative POA Local, CLUSTALW, DIALIGN2 and T-COFFEE on the BAliBASE alignment test set are shown in Figure 3.

Progressive POA provided a major improvement in alignment quality over the original Iterative POA. The improvements were greatest in the most difficult alignment categories: below 25% identity (Ref1.1); multifamily (Ref3); long terminal extensions (Ref4); and long internal insertions (Ref5). These improvements ranged from a 73% increase in alignment score for Ref3, 70% increase for Ref5, 60% for Ref1.1, and 29% for Ref4. The use of global alignment scoring and a truncated gap penalty both contributed to these large increases. For example, global alignment scoring improved partial order alignment quality for reference sets 1–3, in which the sequences have been trimmed to core blocks that align over their complete length (Karplus and Hu, 2001), as demonstrated by the fact that Iterative POA Global outperformed Iterative POA Local on these sets. By the same token, the global alignment programs CLUSTALW and T-COFFEE outperformed the local alignment program DIALIGN2 on these sets as well. The truncated gap penalty, when used with global...
alignment scoring, was responsible for the improvement in partial order alignment quality for reference sets 4 and 5, in which the sequences should not align over their complete length, as demonstrated by the fact that we ran Iterative POA Global without the truncated gap penalty it did not exhibit an improvement relative to Iterative POA Local on these sets (data not shown). Still, the use of global alignment scoring and a truncated gap penalty does not account for all the improvement of Progressive POA (Global) over the original Iterative POA. Progressive POA (Global) outperformed Iterative POA Global, which includes the truncated gap penalty, on reference sets 3 (multifamily alignment) and 4 (long terminal extensions).

To summarize the total results, we tabulated the relative performance of Progressive POA versus other methods, treating differences in score of less than 0.05 as a ‘tie’. This threshold is sensitive to the differences in performance among existing methods but not excessively. For example, T-COFFEE generally had higher scores than CLUSTALW and DIALIGN2. By the above criterion, T-COFFEE’s alignment accuracy was better than CLUSTALW in three sets and tied in two sets (which we will tabulate as 3–2–0). Its performance versus DIALIGN2 was identical. By the same criterion, Progressive POA scored better than Iterative POA in four out of five sets and was tied in the remaining set (4–1–0).

Progressive POA appeared to be intermediate in overall alignment quality between CLUSTALW, DIALIGN2 and T-COFFEE. It scored better than CLUSTALW in two sets and tied in the remaining three (2–3–0). It tied with DIALIGN2 in four out five sets (1–4–0). In contrast, it scored worse than T-COFFEE in two sets, and tied in the remaining three (0–3–2).

**Speed and scalability of Progressive POA compared with other MSA methods**

We have compared the speed and accuracy of Progressive POA, including the time to run BLAST to generate the matrix of pairwise similarity scores, versus Iterative POA, CLUSTALW, DIALIGN2 and T-COFFEE, as a function of increasing sequence length (Fig. 4). A test set of 50 sequences of varying lengths was generated by ROSE (see System and methods section). Consistent with the results of Lassmann and Sonnhammer (2002), Iterative POA was dramatically faster than CLUSTALW, DIALIGN2 and T-COFFEE. Progressive POA was slower than Iterative POA but still much faster than other methods. Progressive POA was 6–7 times faster than T-COFFEE in four out five sets (1–4–0). In contrast, it scored worse than T-COFFEE in two sets, and tied in the remaining three (0–3–2).

We have compared the speed and accuracy of Progressive POA, including the time to run BLAST to generate the matrix of pairwise similarity scores, versus Iterative POA, CLUSTALW, DIALIGN2 and T-COFFEE, as a function of increasing sequence length (Fig. 4). A test set of 50 sequences of varying lengths was generated by ROSE (see System and methods section). Consistent with the results of Lassmann and Sonnhammer (2002), Iterative POA was dramatically faster than CLUSTALW, DIALIGN2 and T-COFFEE. Progressive POA was slower than Iterative POA but still much faster than other methods. Progressive POA was 6–7 times faster than T-COFFEE in four out five sets (1–4–0). In contrast, it scored worse than T-COFFEE in two sets, and tied in the remaining three (0–3–2).

We have compared the speed and accuracy of Progressive POA, including the time to run BLAST to generate the matrix of pairwise similarity scores, versus Iterative POA, CLUSTALW, DIALIGN2 and T-COFFEE, as a function of increasing sequence length (Fig. 4). A test set of 50 sequences of varying lengths was generated by ROSE (see System and methods section). Consistent with the results of Lassmann and Sonnhammer (2002), Iterative POA was dramatically faster than CLUSTALW, DIALIGN2 and T-COFFEE. Progressive POA was slower than Iterative POA but still much faster than other methods. Progressive POA was 6–7 times faster than T-COFFEE in four out five sets (1–4–0). In contrast, it scored worse than T-COFFEE in two sets, and tied in the remaining three (0–3–2).

To summarize the total results, we tabulated the relative performance of Progressive POA versus other methods, treating differences in score of less than 0.05 as a ‘tie’. This threshold is sensitive to the differences in performance among existing methods but not excessively. For example, T-COFFEE generally had higher scores than CLUSTALW and DIALIGN2. By the above criterion, T-COFFEE’s alignment accuracy was better than CLUSTALW in three sets and tied in two sets (which we will tabulate as 3–2–0). Its performance versus DIALIGN2 was identical. By the same criterion, Progressive POA scored better than Iterative POA in four out of five sets and was tied in the remaining set (4–1–0).

Progressive POA appeared to be intermediate in overall alignment quality between CLUSTALW, DIALIGN2 and T-COFFEE. It scored better than CLUSTALW in two sets and tied in the remaining three (2–3–0). It tied with DIALIGN2 in four out five sets (1–4–0). In contrast, it scored worse than T-COFFEE in two sets, and tied in the remaining three (0–3–2).

**Speed and scalability of Progressive POA compared with other MSA methods**

We have compared the speed and accuracy of Progressive POA, including the time to run BLAST to generate the matrix of pairwise similarity scores, versus Iterative POA, CLUSTALW, DIALIGN2 and T-COFFEE, as a function of increasing sequence length (Fig. 4). A test set of 50 sequences of varying lengths was generated by ROSE (see System and methods section). Consistent with the results of Lassmann and Sonnhammer (2002), Iterative POA was dramatically faster than CLUSTALW, DIALIGN2 and T-COFFEE. Progressive POA was slower than Iterative POA but still much faster than other methods. Progressive POA was 6–7 times faster than T-COFFEE in four out five sets (1–4–0). In contrast, it scored worse than T-COFFEE in two sets, and tied in the remaining three (0–3–2).
Fig. 4. Comparison of the performance of Progressive POA and other programs, as a function of increasing sequence length. (A) Log–log plot of the average CPU time consumed by each program. The CPU time for Progressive POA Global includes the time to run BLAST to get the pairwise similarity scores between the sequences. (B) Average SPS for each program plotted versus the log of the sequence length. The data for average CPU times greater than 2000 s were calculated but are not shown. The average CPU time for running CLUSTALW, DIALIGN2 and T-COFFEE on a set of sequences of length 2000 was 832.58, 2248.40 and 8346.20 s, respectively.

exponent for the Progressive POA step alone was only 1.25 for the sets with 90% sequence identity. In fact, the Progressive POA step is so fast that as much as 85% of the total alignment time is actually the pre-POA pairwise similarity score matrix construction step (using BLAST). For example, for a set of 1000 sequences of length 500 with 90% sequence identity, the Progressive POA step takes only 2 min 14.7 s, while the pre-POA BLAST step takes roughly 13 min, resulting in a total alignment time of 15 min. Indeed, the Progressive POA algorithm is frequently faster than the original Iterative POA algorithm, contrary to the appearance in Figures 4A and 5A, where the overall CPU time (including the BLAST pairwise similarity score matrix construction step) is shown.

DISCUSSION

Progressive POA offers some significant improvements. First, it improves alignment quality over Iterative POA, without a serious loss of speed. In our tests its BALiBASE alignment quality scores were 30–70% higher than Iterative POA. More fundamentally, it addresses a basic problem of Iterative POA, namely its sensitivity to the order in which sequences are supplied for alignment. While our Progressive POA tests took about twice as long as Iterative POA, most (and in some cases all) of the increase was due to the simplistic pairwise similarity score matrix construction method we used (All versus All BLAST). The Progressive POA algorithm itself is
Combining partial order alignment and progressive multiple sequence alignment

Fig. 5. Comparison of the performance of Progressive POA and other programs, as a function of increasing number of sequences. (A) Log–log plot of the average CPU time consumed by each program. The CPU time for Progressive POA Global includes the time to run BLAST to get the pairwise similarity scores between the sequences. (B) Average SPS for each program plotted versus the log of the number of sequences. (C) Log–log plots of the CPU time for each program, to show the effect of the level of sequence identity (30%, 60%, 90%) on Iterative POA (IP30, IP60, IP90), MAFFT (FFT30, FFT60, FFT90) and Progressive POA (PP30, PP60, PP90). N.B. The CPU time show in (C) only includes the Progressive POA algorithm itself, not the time for running BLAST to get the matrix of pairwise similarity scores, which is included in (A) and Figure 4A (see text).
approximately equal in speed to Iterative POA. It is likely that a faster pairwise similarity score matrix construction method and/or a faster guide tree construction method, such as the one used in MAFFT (see below), which does not rely on BLAST, can be found. Second, Progressive POA produces alignments comparable in quality with CLUSTALW, DIALIGN2 and T-COFFEE, but is ten to hundreds of times faster for large alignments. It also appears to be more efficient in memory usage.

We wish to emphasize that Progressive POA is an extremely simple algorithm. It is nothing more than the original dynamic programming alignment algorithm (Needleman and Wunsch, 1970; Smith and Waterman, 1981) extended to work with partial order graphs (Lee et al., 2002) and performed in the order determined by running agglomerative nearest-neighbor clustering on the matrix of pairwise sequence similarity scores (i.e. BLAST bit scores). None of the clever and powerful improvements to MSA from the last 15 years has yet been applied. It is pleasing that such a simple approach can produce results comparable with the best existing methods. It also suggests that major improvements to the algorithm and its results are still possible.

Indeed, our approach has many shortcomings and deficits that require further work. For example, POA uses fixed pairwise residue scoring (e.g. PAM, BLOSUM matrices), rather than more sophisticated profile or position-specific scoring systems. Even worse, the standard scoring matrices (designed for pairwise linear sequence versus sequence scoring) are not completely appropriate for PO-MSA versus PO-MSA alignment. As the number of sequences (and their level of sequence dissimilarity) in a PO-MSA increases, the average score for aligning a random sequence to this PO-MSA using these scoring matrices will increase, and eventually become positive, potentially yielding spurious alignments of regions of sequence that actually have no statistically significant homology (Lee et al., 2002). This known problem can be corrected by applying scoring statistics that handle partial order branching properly. Application of a profile scoring system like CLUSTALW (Thompson et al., 1994) or an information–theoretic scoring function (Yona and Levitt, 2002) could help correct this.

More sophisticated scoring functions will require more computing time. However, there may even be ways to use them to accelerate the overall alignment process. One of the reasons why the power law $\beta$ for Progressive POA (excluding BLAST) approaches 2 ($\beta$ is 1.63 for 30% I.D.) as the sequence identity decreases is that low levels of node fusion throughout the progressive alignment process result in intermediate PO-MSAs containing roughly one path for each sequence they contain. Since the PO–PO alignment algorithm finds the best alignment of any path in one intermediate PO-MSA to any path in another intermediate PO-MSA, as the number of paths in each of the intermediate PO-MSAs approaches the total number of sequences they each contain, the full Progressive POA algorithm essentially performs $0.5N(N - 1)$ pairwise sequence alignments. On the other hand, the power law $\beta$ for Progressive POA (excluding BLAST) approaches 1 ($\beta$ is 1.25 for 90% I.D.) as the sequence identity increases because high levels of node fusion throughout the progressive alignment process result in intermediate PO-MSAs containing only a single path. In this case, the PO–PO alignment algorithm essentially performs one pairwise sequence alignment each time it is called, so that the full Progressive POA algorithm performs $N - 1$ pairwise sequence alignments. These data show that increasing the amount of node fusion that takes place during the buildup of a PO-MSA increases Progressive POA’s overall speed. This suggests that a profile-based scoring system, in which all aligned amino acids, regardless of identity, are fused into single ‘profile’ nodes, each representing a mix of several different amino acids (for several example approaches, see Altschul et al., 1997; Eddy, 1996; Gribskov et al., 1987; Thompson et al., 1994; Yona and Levitt, 2002), could result in increased alignment speed, despite the constant increase in the amount of time required to compare two profiles relative to comparing two residues. Similarly, using ‘profile’ nodes could increase the speed of Iterative POA.

Recently, the program MAFFT has been reported as a very high speed and accurate MSA algorithm (Katoh et al., 2002) and in our tests is somewhat faster than POA (Fig. 5C). However, it is surprising that POA is able to approach the speed of MAFFT since unlike CLUSTALW, T-COFFEE and POA, MAFFT bypasses full dynamic programming sequence alignment in favor of a fast diagonal-finding method. In this respect MAFFT is like BLAST and FASTA, although it uses a different segment-match algorithm based on the fast Fourier transform. MAFFT uses these segment matches to accelerate sequence alignment dramatically, by excluding all regions of the full dynamic programming matrix outside of these segment matches. Although this heuristic approach does not guarantee that the resulting alignment will be optimal (unlike full dynamic programming, there may exist other alignments with a higher score), it tremendously accelerates the alignment process. In contrast, POA performs full dynamic programming, computing the entire matrix for all sequences, and so it is surprising that it is able to achieve a speed similar to that of MAFFT [unlike other full dynamic programming algorithms such as CLUSTALW and T-COFFEE, which have been reported to be at least 100-fold slower than MAFFT (Katoh et al., 2002)]. Since MAFFT and POA represent completely orthogonal modifications to MSA, aspects of each of the algorithms could be combined. For example, the time for MAFFT, which includes the time for building the guide tree, is less than that of Progressive POA, which excludes the time for constructing the pairwise similarity scores necessary for guide tree construction (Fig. 5C). This suggests that Progressive POA could be greatly accelerated simply by employing the guide tree construction method used in MAFFT, rather than using BLAST and agglomerative nearest-neighbor clustering. Moreover, the
impressive quality of MAFFT’s results on BAliBASE (Katoh et al., 2002) suggests that such a hybrid method could produce good alignments.

One major direction for improvement is simply to integrate POA with the most important advances in MSA methods that have proven valuable in common usage. Progressive POA is essentially Higgins’ 1988 CLUSTAL algorithm (Higgins and Sharp, 1988) with PO-MSAs replacing linear profiles, and PO–PO alignment replacing standard pairwise alignment. Since both CLUSTALW and T-COFFEE are orthogonal extensions of the CLUSTAL algorithm, their innovations could be combined with Progressive POA as well. For example, T-COFFEE preprocesses the set of pairwise alignments constructed to build the initial guide tree and then uses this information to build intermediate alignments that are consistent with how all the sequences should ultimately align to each other. The performance of Progressive POA, which uses only the scores from the initial pairwise alignments, would be greatly improved by a similar approach.

On the other hand, iterative POA could also be advanced in ways that do not involve progressive alignment. Partial ordering has previously been proposed as a mathematical condition for the consistency of a proposed MSA (Morgenstern et al., 1996). This principle has been used by the program DIALIGN to prune the set of possible combinations of local (segment) alignments while building up an MSA with a greedy algorithm and has been applied successfully to both DNA and protein sequences (Morgenstern et al., 1996). Since DIALIGN relies on the assumption that an alignment is made up of regions that should and should not align and since one of the major values of the PO-MSA representation is its ability to accurately represent unaligned as well as aligned regions, it could be used in an iterative version of the DIALIGN algorithm as well.

Finally, all the MSAs test sets used in this study provide alignments in row–column MSA (RC–MSA) format. In this format, a single residue substitution is represented as two different letters in the same column, while a single-residue insertion/deletion is represented as a single gap symbol. The fundamental assumption underlying this format is that the set of aligned sequences was generated from a single consensus sequence through a process of single-residue substitutions, insertions and deletions and should therefore be aligned over their complete length. However, there are also five possible large-scale edits observed between biological sequences: large insertions/deletions, terminal extensions, inversions, transpositions and duplications. In addition, it is possible for regions of sequence to be completely unaligned. While a large insertion/deletion, terminal extension or region of unaligned sequence can be represented in RC-MSA format as a long series of gap symbols, inversions, transpositions and duplications cannot be represented in this format at all. In contrast, it is possible to represent all five large-scale edits, in addition to the three single-residue edits, in the PO-MSA format. This suggests that a MSA test set based on the PO-MSA format instead of the RC-MSA format could include complex MSAs of complete sequences. In particular, alignments containing various aligned and unaligned regions could be included. While Progressive POA performed well on standard RC-MSA test sets relative to other standard MSA algorithms, we intend to develop a PO-MSA based test set on which to assess its ability to handle combinations of aligned and unaligned regions. Such a test set could also be used as a benchmark in the development of new methods for aligning sequences that are related by all five types of large-scale edits, including inversions, transpositions and duplications.

The results presented in this paper provide evidence for the utility of the Progressive POA idea and, more importantly, demonstrate the speed and scalability of Progressive POA relative to standard MSA programs, CLUSTALW, DIALIGN2 and T-COFFEE. There is a need for MSA methods that can scale to larger and increasingly challenging alignment problems. At this very early stage in the development of the POA idea, there appear to be many further opportunities for improving performance.

ACKNOWLEDGEMENTS

We wish to thank Dr D.S. Parker, A. Resch, M. Quist and Y. Xing for their helpful discussions and comments on this work. C. G. is supported by a DOE Computational Science Graduate Fellowship. This work was supported by the National Institute of Mental Health and the National Institute of Neurological Disorders and Stroke (Grant no. MH65166), National Science Foundation grant IIS-0082964 and Department of Energy grant DEFG0387ER60615.

REFERENCES


