BIOINFORMATICS

Vol. 15 no. 2 1999
Pages 122–130

Combining many multiple alignments in one improved alignment

Klaus Bucka-Lassen[1], Ole Caprani[2] and Jotun Hein[3]

[1]Object Oriented Ltd, 6004 Luzern, Switzerland, [2]Department of Computer Science and [3]Department of Ecology and Genetics, University of Aarhus, 8000 Aarhus C, Denmark

Received on April 1, 1998; revised on October 20, 1998; accepted on November 2, 1998

Abstract

Motivation: *The fact that the multiple sequence alignment problem is of high complexity has led to many different heuristic algorithms attempting to find a solution in what would be considered a reasonable amount of computation time and space. Very few of these heuristics produce results that are guaranteed always to lie within a certain distance of an optimal solution (given a measure of quality, e.g. parsimony). Most practical heuristics cannot guarantee this, but nevertheless perform well for certain cases. An alignment, obtained with one of these heuristics and with a bad overall score, is not unusable though, it might contain important information on how substrings should be aligned. This paper presents a method that extracts qualitatively good sub-alignments from a set of multiple alignments and combines these into a new, often improved alignment. The algorithm is implemented as a variant of the traditional dynamic programming technique.*

Results: *An implementation of ComAlign (the algorithm that combines multiple alignments) has been run on several sets of artificially generated sequences and a set of 5S RNA sequences. To assess the quality of the alignments obtained, the results have been compared with the output of MSA 2.1 (Gupta et al., Proceedings of the Sixth Annual Symposium on Combinatorial Pattern Matching, 1995; Kececioglu et al., http://www.techfak.uni-bielefeld.de/bcd/Lectures/kececioglu.html, 1995). In all cases, ComAlign was able to produce a solution with a score comparable to the solution obtained by MSA. The results also show that ComAlign actually does combine parts from different alignments and not just select the best of them.*

Availability: *The C source code (a Smalltalk version is being worked on) of ComAlign and the other programs that have been implemented in this context are free and available on WWW {http://www.daimi.au.dk/~ocaprani}.*

Contact: *klaus@bucka-lassen.dk; jotun@pop.bio.au.dk; ocaprani@daimi.au.dk*

Introduction

Pairwise alignment

Under the assumption that the two DNA sequences s_0 and s_1 have had a common ancestor, there must be a number of mutations (here only substitutions, insertions and deletions are taken into consideration) that transform s_0 into s_1 and vice versa (Sankoff, 1972). These mutations will here be represented by an alignment, a $2 \times len$ matrix, where *len* determines the length of the alignment and a dash represents an insertion or deletion. For example:

$$A\ C\ C - G\ A\ C\ T\ T$$
$$- C\ T\ T\ G - C\ T -$$

Given a cost for each of the possible mutations (the costs used are given in Table 1), the overall cost of an alignment can be determined by accumulating all column mutation costs. The optimal alignments (there can be more than one) are defined to be those that have the lowest possible cost (parsimony). An optimal alignment can be determined by calculating the cost for all possible alignments and choosing the cheapest. This might, depending on the length of the sequences, take a very long time.

Table 1. The mutation cost used

	A	C	G	T	–
A	0	5	2	5	10
C	5	0	5	2	10
G	2	5	0	5	10
T	5	2	5	0	10
–	10	10	10	10	–

An alignment can also be illustrated with a weighted directed acyclic graph (WDAG), as shown in Figure 1, where each edge corresponds to a mutation (and its weight with the mutations cost) and an alignment to a path from the source to the target.

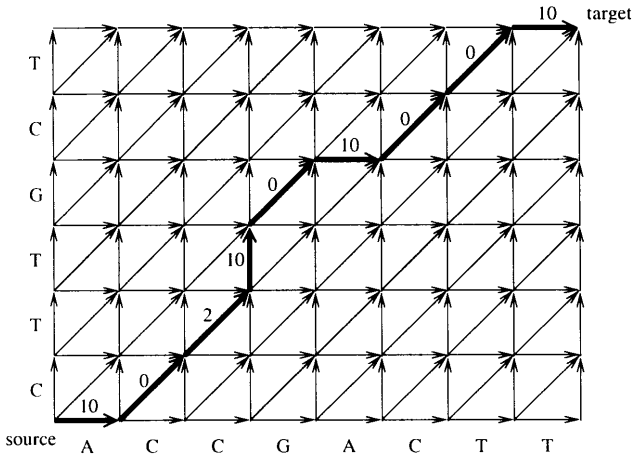

Fig. 1. A WDAG that illustrates a possible alignment of the two DNA sequences ACCGACTT and CTTGCT.

The cost $D_{|s0|,|s1|}$ of an optimal alignment of the two sequences s_0 and s_1 can be determined with the recursive function:

$$D_{i,j} = min\begin{cases} D_{i,j-1} + indel \\ D_{i-1,j} + indel \\ D_{i-1,j-1} + sub(s_0[i], s_1[j]) \end{cases}$$

$$D_{0,0} = 0$$

where i and j are non-negative integers and *indel* and $sub(n_0,n_1)$ specify the mutation cost of an insertion/deletion and a substitution of the two nucleotides n_0 and n_1, respectively.

By applying dynamic programming (Füllen, 1997; Myers, 1991; Waterman *et al.*, 1991; Chan *et al.*, 1992; Waterman, 1995) to this recursive function, each value of $D_{i,j}$ will only be calculated once. Since there are only $|s_0| \cdot |s_1|$ different values $D_{i,j}$ and the computation of each of them requires a constant amount of time, the time complexity will be:

$$T[\text{DynAlg}(s_0,s_1)] \in \mathcal{O}(|s_0| \cdot |s_1|)$$

The dynamic programming algorithm is equivalent to a single-source shortest path (Dijkstra, 1959) computation on the WDAG. Hereafter, an optimal alignment is obtained by backtracking from the target to the source.

Multiple alignment

Multiple alignment is a generalization of pairwise alignment. For example:

$$A = \begin{bmatrix} A & C & C & - & G & A & C & T & T \\ - & C & T & T & G & - & C & T & - \\ - & C & C & C & T & G & A & C & T \end{bmatrix}$$

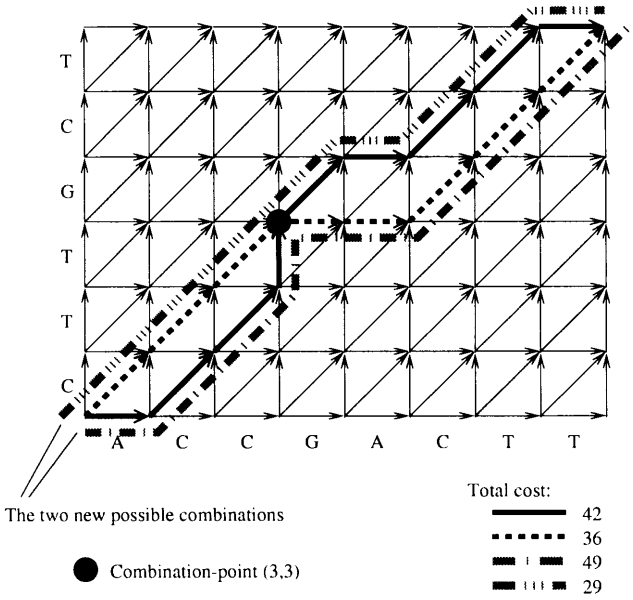

Fig. 2. Two alignments that can be combined into two new and different alignments.

The cost of a multiple alignment is here defined to be the accumulated cost of all projected pairwise alignments ('Sum-of-pairs'; Gusfield, 1997).

The time complexity of the dynamic programming algorithm that computes an optimal multiple alignment is:

$$T[\text{DynAlg}(s_0, s_1, \ldots, s_{m-1})] \in \mathcal{O}((2^m-1) \cdot \prod_{j=o}^{j<m} |s_j|)$$

The space required is $\mathcal{O}(\prod_{j=0}^{j<m} |s_j|)$.

For ease of understanding, mostly throughout this paper, ideas and algorithms will be presented in a two-dimensional case, i.e. for alignments of two sequences.

Combining alignments

Assume that two different alignments A_0 and A_1 of the two DNA sequences s_0 and s_1 have a common point (i_0,i_1) (combination point), such that both align the subsequences $s_0[0..i_0]$ with $s_1[0..i_1]$ and $s_0[i_0..|s_0|]$ with $s_1[i_1..|s_1|]$, where $s[i..j]$ determines the substring of s that starts at position i and ends with position $j-1$. One alignment might align the first part of the sequences qualitatively well, while the other alignment performs better on the remainder of the sequences. A combination of these two superior sub-alignments will then score at least as good as, but probably even better than, the original alignments did. Figure 2 illustrates how two such alignments can be combined into two new alignments (one of them being better than either of the original alignments).

This idea is easy generalized to handle n multiple alignments ($A_0, A_1, \ldots, A_{n-1}$—called base alignments or set) of m sequences ($s_0, s_1, \ldots, s_{m-1}$).

The available algorithms for aligning two sequences to optimality perform reasonably well in both time and space, so the combination method does not add much value in determining pairwise alignments. For multiple alignments of a certain size, however, there are no practical algorithms for determining an optimal alignment (indicated by the time and space complexity of the dynamic programming algorithm); various heuristics that produce close-to-optimal solutions are the best there is. Combining good sub-alignments can never hurt (apart from the extra computation time), since an optimal combination will always score at least as well as the best base alignment.

Algorithms

Determining an optimal combination

The number of different new alignments that can be built by combining sub-alignments from a base of two alignments grows exponentially with the number of combination points. At the same time, one would expect the number of combination points to grow with the number of different alignments in the base (the more alignments, the higher the probability that some of them intersect). So the number of combined alignments one has to examine to find the optimal combination can be expected to grow faster than the number of alignments in the base. Once again, dynamic programming offers a simple solution to this problem:

Algorithm 1: OptiCom
Input: *The base $A = \{A_0, A_1, \ldots, A_{n-1}\}$, a set of alignments of $s_0, s_1 \ldots, s_{m-1}$.*
Output: *An optimal combination of the base alignments.*
Method: *Mark all edges that are part of a base alignment in the WDAG and run the dynamic algorithm only on these edges (see recursive definition below).*

For this dynamic algorithm, the time complexity is as follows ($|A|$ is the number of edges in the alignment A):

$$T[\text{OptiCom}(A_o, A_1, \ldots, A_{n-1})] \in \mathcal{O}(\sum_{j=0}^{n-1} |A_j|)$$

$$\subseteq \mathcal{O}(n \cdot \max_{\leq j<n} \{|A_j|\})$$

$$\subseteq \mathcal{O}(n \cdot (\sum_{j=0}^{j<m} |s_j|))$$

which is a lot better than $\mathcal{O}((2^m - 1) \cdot \prod_{j=0}^{j<m} |s_j|)$, the time complexity of the dynamic algorithm that determines an optimal multiple alignment.

The actual implementation

The algorithm OptiCom is actually implemented a bit differently. Instead of marking edges, all nodes that are part of a base alignment are marked. The best combination is now determined as:

$$D_{i,j} = \min \begin{cases} D_{i,j-1} + \mathit{indel}, & \text{if } \mathit{node}(i, j-1) \text{ is marked} \\ D_{i-1,j} + \mathit{indel}, & \text{if } \mathit{node}(i-1, j) \text{ is marked} \\ D_{i-1,j-1} + \mathit{sub}(s_0[i], s_1[j]), & \text{if } \mathit{node}(i-1, j-1) \text{ is marked} \end{cases}$$

Hence, a combined path can also consist of edges that are not marked, which means that the algorithm does not necessarily need a combination point to combine two different alignments; it can also 'jump' from one alignment to another if the distance is not too big (here one unmarked edge). The time complexity will be affected by this, but is still much better than the standard dynamic algorithm:

$$T[\text{OptiCom}(A_0, A_1, \ldots, A_{n-1})] \in \mathcal{O}(n \cdot (2^m - 1) \cdot \sum_{j=o}^{j<m} |s_j|))$$

The reasons for this choice of implementation are that it gives more combined alignments and is easier to implement.

Iterative combination of alignments

In general, it will not be the case that one is given a set of alignments. More often, the input will be a set of sequences that should be multiple aligned 'as good as possible'. This is achieved with the following extension to OptiCom:

Algorithm 2: ComAlign
Input: *$s_0, s_1, \ldots, s_{m-1}$, a set of sequences. n, the number of iterations.*
Output: *X, a multiple alignment of $s_0, s_1, \ldots, s_{m-1}$*
Method:
1. *Generate an alignment A of $s_0, s_1, \ldots, s_{m-1}$.*
2. *Mark the nodes, given by A, in the WDAG.*
3. *Repeat n times step 1 and 2.*
4. *Use OptiCom to determine an optimal combination X of all generated alignments.*

The time it takes to run ComAlign can be split into two different components: the time it takes to generate all the base alignments and the time it takes OptiCom to find an optimal combination.

For a given amount of time, this means that one can either spend the majority of computational time on finding good, but presumably not very many, alignments (under the assumption that there is some correlation between the quality of an alignment and the time it takes to determine this alignment), or one could generate alignments of a possibly poorer quality, but more of them. Heuristics that produce close to optimal multiple alignments are rare, mostly very time consuming and hard to implement (because they tend to be rather

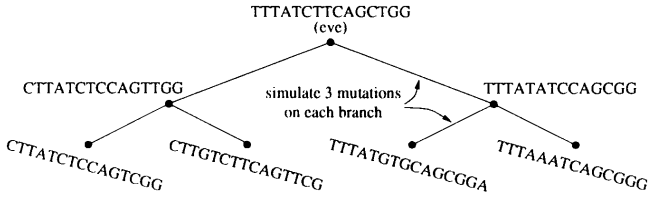

Fig. 3. Example of how DNAgen generates four sequences.

complex). Progressive alignment (or 'Alignment along a Tree') (Feng and Doolittle, 1987; Füllen, 1997), on the other hand, is a heuristic that is easy to implement and produces results rather quickly, but it does in general not score very high on the quality scale. Another advantage of using progressive alignment in this context is that it is quite easy to produce many different alignments from the same sequences by choosing random phylogenies; reasons enough to choose this heuristic to generate the base alignments for ComAlign.

ProgAlign, our implementation of Progressive Alignment, is implemented straightforwardly as described in Füllen (1997) with one little tune-up: we have added a profile representation (Chan *et al*., 1992, p. 574) to each mutliple alignment and speeded up the process of aligning multiple alignments that way.

Results for simulated data

A variant of ComAlign, where step 4 is executed after each step 2 so that we can observe how the score of an optimal alignment is evolving and how each individual base alignment is influencing this, has been implemented and run on various inputs (sets of sequences).

To assess the quality of the multiple alignments that are produced by ComAlign, these are compared with the output from the MSA 2.1 (Gupta e*t al*., 1995) program. MSA is run with the options:

–c dna = mutation costs defined in file 'dna'.
–g = all gaps cost the same.
–b = use sum-of-pairs score function.

The 'dna' file was defined in accordance with Table 1. ComAlign and MSA have been run on a Silicon Graphics SGI Indigo2, 200 MHz IP22 processor with 128 Mb of memory. The sequences that have been used for comparison were generated artificially with DNAgen.

DNAgen—a DNA simple sequence generator

DNAgen is a program that generates a random eve-sequence and simulates evolution based on various parameters. The possible parameters for DNAgen are:

s# = random seed.
n# = number of sequences to be generated.
1# = approximate length of the sequences.
g# = number of mutations per generation.

To generate four related sequences, each being ~15 nucleotides long, use the following set of parameters for DNAgen:

DNAgen –s0 –n4 –l15 –g3

The result is four simulated sequences written in a format that can be read by both MSA and ComAlign:

```
>Artificial generated sequence 1
CTTATCTCCAGTCGG
*
*
>Artificial generated sequence 2
CTTGTCTTCAGTTCG
*
*
>Artificial generated sequence 3
TTTATGTGCAGCGGA
*
*
>Artificial generated sequence 4
TTTAAATCAGCGGG
*
*
```

DNAgen starts by generating a random eve-sequence with exactly 15 uniformly distributed nucleotides. Then two new sequences are obtained by simulating three mutations on the eve-sequence. Each of these two new sequences is then used as a root, just as the eve-sequence, to generate four sequences, etc. (see Figure 3). The simulated mutations incorporate transitions, transversions and indels with given frequencies (transitions 0.6, transversions 0.3 and indels 0.1).

Example 1 Make a multiple alignment with MSA and ComAlign and compare the results.

- Use DNAgen, generate eight simulated sequences with ~50 nucleotides each. Simulate 20 mutations on each generation:

```
CCAAGTGCTAGATCGTTTCTTCCTCTACTAAATATGCGACGCCAG
CTGCGGATACATAACTCGTGGCCGTGCCGAAAATCC M ATGAG
CATGAGATTGACTATTACCCTTAGAGCTAGGACAATGAAAACATGTTAA
CACTCTAGTCCACAAACGACCTATGGAAACTATGATTAGCAGATGTGAGGCTGA
TTAAACCAGATACGAAATGCACCCGAGCGTGTCTCCTGGCACA
TTAGCGCTCTGCAAAGTGATCCTGGACCGCTTCCTGAGAGG
TCTATCCGCAAGATATTACTCCACAAGACTGCCTATCCATAGT
TCTACGTAAGTTACTCGCAAGGGTTGTATCTAATGCTTTTGAGACA
```

- Align these sequences with MSA. The output is as follows (irrelevant information, in this context, has been removed):

```
C--C-AAGTGCTAGA-TCG-TT-TCTT--CCTCTACTA-AATATGCGACGCCAG
C--T-GCGGATAC-A-TAA-CT-CGTGG-CCGTGCCGA-AAATCCT-TTATGAG
CA-TGAGATTGACTATTACCCT-TAGAG-CTAGGACAA-TGAAAAC-ATGTTAA
CACTCTAGTCCACAAACGACCTATGGAAACTATGATTAGCAGATGTGAGGCTGA
T--T-AAACCAGATA-CGA-AA-TGCA--CC-CGAGCG-TGTCTCC-TGGCACA
T--T-A-GCGCTCTG-CAA-AG-TGAT--CC-TGGAC--CGCTTCC-TGAGAGG
T--C-T-ATCCGCAA-GAT-AT-TACT--CCACAAGAC-TGCCTAT-CCATAGT
TC-T-ACGTAAGTTA-CTC-GC-AAGGG-TTGTATCTA-ATGCTTT-TGAGACA


Alignment cost: 4307
Elapsed time = 24.054 [s]
```

Table 2. Output of ComAlign after 100 iterations and 44.27 s (an optimal combination was found after 87 iterations and 37.00 s)

iteration	current score	min. score	max. score	ComAlign score	time $\frac{\text{sec.}}{100}$
0	4683	4683	4683	4683	34
1	4560	4560	4683	4560	70
2	5108	4560	5108	4560	103
3	4800	4560	5108	4560	136
4	4490	4490	5108	4485	170
...					
27	5052	4447	5375	4447	1033
28	4813	4447	5375	4430	1070
...					
31	4730	4447	5375	4430	1192
32	4794	4447	5375	4326	1231
...					
45	4653	4447	5375	4326	1780
46	4725	4447	5375	4315	1822
...					
50	4767	4447	5375	4315	1998
51	4538	4447	5375	4285	2042
...					
62	4463	4445	5375	4285	2533
63	4972	4445	5375	4273	2578
...					
86	4996	4445	5452	4273	3653
87	4547	4445	5452	4267	3700
...					
98	4982	4445	5463	4267	4307
99	4742	4445	5463	4267	4369
100	4810	4445	5463	4267	4427

```
C--C-AA-GTGCTAGATCGTTTCTTC---CTCTACTAAATA-TGCG-ACGCCAG
C--T--G-CGGATACATAACTCGTGG---CCGTGCCGAAAA-TCCT-TTATGAG
CA-T-GAGATTGACTATTACCCTTAGAG-CTAGGACAATGA-AAAC-ATGTTAA
CACTCTAGTCCACAAACGACCTATGGAAACTATGATTAGCAGATGTGAGGCTGA
T--T-AA-ACCAGATACGAAATGCAC---CCGAGCGT-GT--CTCC-TGGCACA
T--T--A-GCGCTCTGCAAAGTGATC---CTG-GACC-GC--TTCC-TGAGAGG
T--C-TA-TCCGCAAGATATTACTCC---ACAAGACT-GC--CTAT-CCATAGT
T--C-TACGTAAGTTACTCGCAAGGG---TTGTATCTAATG-CTTT-TGAGACA

Statistics: Best score:            4267
              found after:           87  iterations
            Total time elapsed:    4427  1/100 sec.
              + overhead time:        0  1/100 sec.
```

- Run ComAlign with $n = 100$ iterations. The result is shown in Table 2 and Figure 4.

Observations:

- The score of an optimal combined alignment is not just as good as the score of the best base alignment, it is better. This means that there is actually made a combination of at least two different base alignments. This combination scores 4% better than the best base alignment.
- Even though the individual base alignments' overall score is comparably bad, sub-alignments of it might be of such a high quality that these are used in the combined alignment. This can, for example, be seen in iterations 4, 28, 32, 46, 51, 63 and 87 (see Table 2 and Figure 4).
- The multiple alignment that ComAlign produces scores a little better than MSA's solution.

A comparison of ComAlign and MSA has also been performed with different numbers of sequences and lengths of these sequences. By default, DNAgen sets the number of mutations to be 10% of the length of the eve-sequence (e.g. if the eve-sequence is 400 nucleotides long, 40 mutations are simulated between each two generations). All other parameters were as in example 1. The results are shown in Table 3.

It seems as if the longer and more sequences there are, the better ComAlign performs compared to MSA. For several complex cases, MSA was not able to produce an output because it ran out of memory.

The space and time consumption by ComAlign has not been examined systematically. Manual monitoring of the resources has shown that ComAlign often uses less space than MSA. For instance, MSA was not able to align seven sequences with an approximate length of 400 nucleotides because it ran out of space after allocating 180 Mb (including swap-space). After 100 iterations, ComAlign had only used 15 Mb on the same sequences. On a few short sequences, however, the actual implementation of ComAlign spends more time finding a solution as good as MSA's alignment. In one particular case, ComAlign spent 30 times as much time as MSA did.

Results for biological data

As many as 22 5S RNA sequences from GenBank (GenBank, 1998) have been chosen to demonstrate the usage of ComAlign on real biological data. These relatively short sequences serve a structural role in the ribosome and are thus not coding for proteins, but will have structural constraints on their sequences, especially related to their secondary structure and interaction with proteins of the ribosome. They are ubiquitous in cellular organisms and thus their most recent common ancestor dates at least 2–3 billion years back. For details on the sequences, see the web page mentioned in the abstract.

The alignment of all 22 sequences is shown in Figure 5. Concentrating on the alignment positions 75–90, it seems as if sequences 4–8 and 18–22 group together and look a bit different than the remaining 12 sequences. We have aligned these 10 sequences with ComAlign and compared the output with a multiple alignment from MSA. The results are shown in Figures 6 and Figures 7 and Table 4.

At an early stage (iteration 40), ComAlign has obtained an alignment that is slightly better than MSA's solution, but it has also spent more time than MSA has on these short sequences. This alignment is clearly a combination of sub-alignments since the best progressive alignment achieved at that point scores worse. At iteration 4597, ProgAlign finds an alignment that scores as good as ComAlign's solution from iteration 40; no better combination is found hereafter.

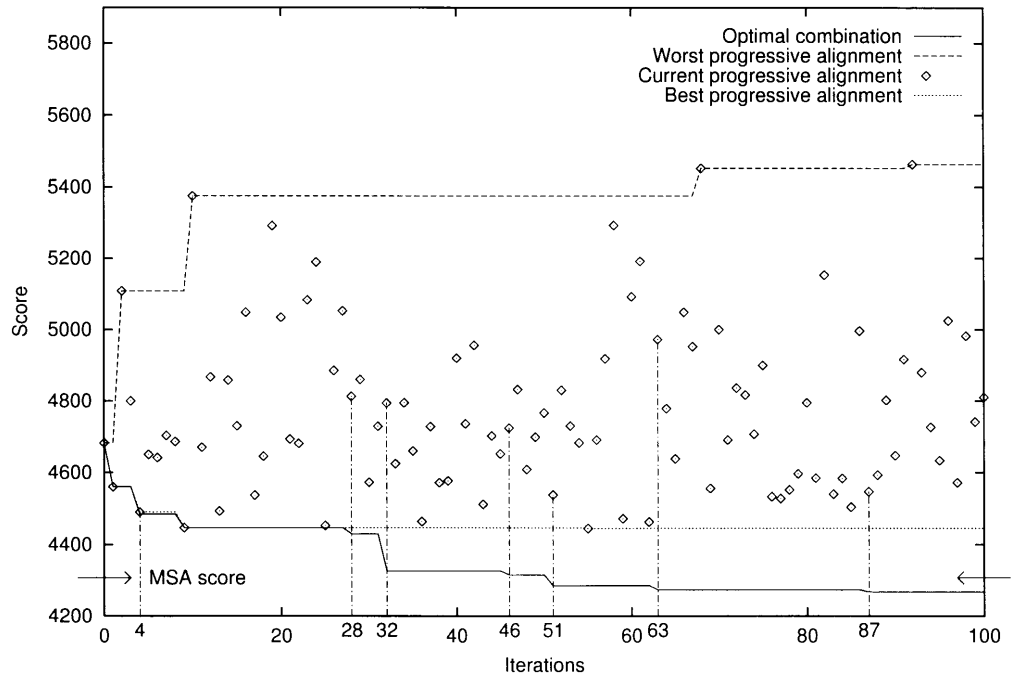

Fig. 4. Comparison of ComAlign and MSA. The points show the score of the current alignment for each iteration. The top and bottom dotted lines show the score of the respectively best and worst base alignment so far. The bottom full line shows the score of an optimal combination (generated by ComAlign).

To prevent ComAlign from running for a very long time without any progress, an optional stopping rule (command line parameter *c*) has been introduced.

This stopping rule makes ComAlign stop as soon as no improvement in the score of the optimal combination has been achieved for a given number of iterations.

Discussion

ComAlign produces the alignments iteratively. This makes it possible to interact with the program. The user could somehow 'guide' ComAlign while it is running, such as telling it to concentrate on a certain area of the sequences or even stop the execution as soon as a solution is found that by any objective or subjective means is 'good enough'.

Furthermore, the quality level can be set to fit the time and space available by:

- Regulating the quality of the base alignments

  The assumption is that ComAlign scores better as the quality of the base alignments rises. To increase the quality of the base alignments, one could, instead of using ProgAlign on random trees, run ProgAlign on a set of trees that describe a realistic phylogeny of the sequences.

- Regulating the coverage of the base alignments

  By coverage, we mean a measure of how much of the dynamic programming lattice is covered. The assumption is that ComAlign in general will score better as the coverage grows. A higher level of coverage can be achieved by extending the base set with new alignments that cover edges that no other base alignment has covered before. In the examples above, a relatively high level of coverage has been achieved by assuming a random phylogenetic tree for each run of ProgAlign. The number of iterations in ComAlign is, therefore, a tool to adjust the coverage. Another possibility would be to change the mutation costs a little after each iteration. Using different heuristics would be another interesting approach. Good local alignments, which often are easier to determine than similar good global alignments, could also serve as a source for ComAlign (the base alignments do not need to be global).

So an increase in the quality or the coverage of the base is assumed to make ComAlign produce better results, but at the same time, this will probably mean an increase in time and space used to determine the base.

Since the probability of alignments crossing each other's paths presumably decreases radically for each extra sequence that is added to the alignment problem, we expect ComAlign to be best suited for a medium number of sequences.

Table 3. ComAlign versus MSA

Number of sequences	Sequence-length	MSA's score msa	ComAlign's score com	Deviation msa - com	Deviation in $\frac{1}{100}\%$ $10000 \cdot (\frac{msa}{com} - 1)$
6	50	961	961	0	0
6	100	2039	2039	0	0
6	150	2977	2977	0	0
6	200	4184	4177	7	17
6	250	5182	5182	0	0
6	300	6098	6098	0	0
6	350	7524	7524	0	0
6	400	8507	8510	-3	-4
6	450	9248	9252	-4	-4
6	500	10166	10162	4	4
7	50	1479	1479	0	0
7	100	3020	3020	0	0
7	150	4581	4581	0	0
7	200	6263	6248	15	24
7	250	7889	7889	0	0
7	300	9377	9379	-2	-2
7	350	11217	11218	-1	-1
7	400	to large a dataset for MSA			
7	450				
7	500				
8	50	2089	2089	0	0
8	100	4276	4270	6	14
8	150	6373	6394	-21	-33
8	200	8657	8604	53	62
8	250	11372	11369	3	3
8	300	13298	13298	0	0
8	350	15474	15482	-8	-5
8	400	17647	17662	-15	-8
8	450	to large a dataset for MSA			
8	500				
9	50	2964	2964	0	0
9	100	5783	5773	10	17
9	150	8821	8528	293	344
9	200	11737	11667	70	60
9	250	15020	14998	22	15
9	300	17418	17441	-23	-13
9	350	to large a dataset for MSA			
9	400				
9	450				
9	500				
10	50	3845	3845	0	0
10	100	7706	7694	12	16
10	150	10983	10903	80	73
10	200	15622	15588	34	22
10	250	19123	19049	74	39
10	300	22079	22079	0	0
10	350	to large a dataset for MSA			
10	400				
10	450				
10	500				

ComAlign has many interesting prospects, one of them being the possibility for the user to interact with the program at run time. Another strength of ComAlign is the possibility of setting the correlated factors time, space and quality according to needs and availability of resources. If the user is interested in a 'quick and dirty' solution, he can choose a base with a low coverage and quality. If a better alignment is needed, the user can increase the coverage and quality of the base accordingly.

Conclusion

We have introduced an algorithm to combine sub-alignments in one new multiple alignment. The goal of this study was to see whether this approach has potential or not. The first results have been very encouraging.

The detailed analysis of one particular randomly chosen case showed that ComAlign actually combined different alignments from the base set. Especially interesting was the combination of sub-alignments from base alignments that had a comparably bad global score.

The general heuristic outlined here could be used in other situations where different solutions are obtained by non-optimal methods, these solutions can be split up in non-correlated sub-solutions, and it is easy to score an actual solution exactly. Combined RNA folding and alignment or multiple

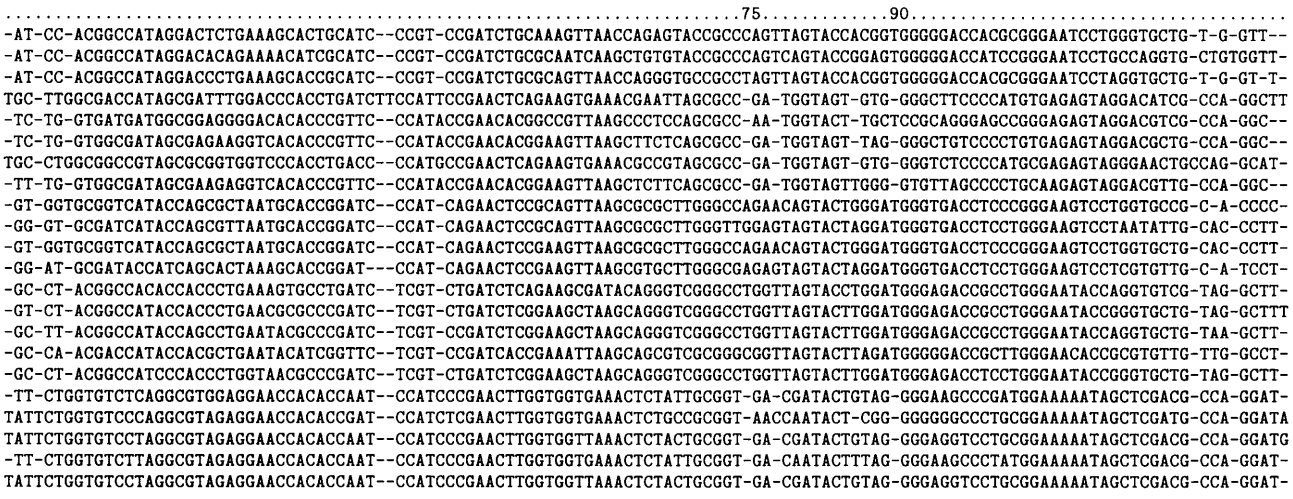

Fig. 5. ComAlign's multiple alignment of 22 5S RNA sequences. Score = 52 421 (found after 113 min at iteration 974); best progressive alignment = 55 137 (found at iteration 874); worst progressive alignment = 83 592 (found at iteration 204); total time spent on 1000 iterations: ~2 h.

Fig. 6. MSA's multiple alignment of the 10 selected 5S RNA sequences. Score = 7640. Total time elapsed: 1.7 s.

Fig. 7. ComAlign's multiple alignment of the selected 10 5S RNA sequences. Score = 7620 (found after 32 s at iteration 40); best progressive alignment = 7620 (found at iteration 4597); worst progressive alignment = 9818 (found at iteration 3123); total time spent on 10 000 iterations: ~3 h.

alignment of protein sequences are problem areas that immediately come to mind.

ComAlign might be able to save researchers time as it in many ways mimics the way that people actually combine different proposed solutions into one which is superior.

References

Chan,S.C., Wong,A.K.C. and Chiu,D.K.Y. (1992) A survey of multiple sequence comparison methods. *Bull. Math. Biol.*, **54**, 563–598.

Dijkstra,E.W. (1959) A note on two problems in connexion with graphs. *Nummer. Math.*, **1**, 269–271.

Table 4. Result of running ComAlign for 10 000 iterations on the set of 10 5S RNA sequences. The best combination was found after 40 iterations or 32 s after the program was started. Later, at iteration 4597 (~1.5 h), a profile alignment is generated that scores equally well. The actual alignment is shown in Figure 7

iteration	current score	min. score	max. score	ComAlign score	time $\frac{\text{sec.}}{100}$
0	8316	8316	8316	8316	78
1	7764	7764	8316	7764	157
2	8254	7764	8316	7700	237
3	7830	7764	8316	7700	316
4	8092	7764	8316	7688	394
5	7837	7764	8316	7688	473
6	7890	7764	8316	7688	552
7	8493	7764	8493	7688	632
8	7686	7686	8493	7632	712
			...		
39	7867	7686	9180	7632	3205
40	7892	7686	9180	7620	3286
41	8131	7686	9180	7620	3368
			...		
3122	8088	7628	9624	7620	296714
3123	9818	7628	9818	7620	296819
3124	7712	7628	9818	7620	296922
			...		
4596	8185	7628	9818	7620	456096
4597	7620	7620	9818	7620	456207
4598	8046	7620	9818	7620	456319
			...		
9999	8390	7620	9818	7620	1110543
10000	8267	7620	9818	7620	1110670

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

Füllen,G. (1997) A gentle guide to multiple alignment 2.03. http://www.techfak.uni-bielefeld.de/bcd/Curric/MulAli/mula-li.html.

GenBank (1998) Genbank. http://www.ncbi.nlm. nih.gov.

Gupta,S.K., Kececioglu,J.D. and Schäffer,A.A. (1995) Making the shortest-paths approach to sum-of-pairs multiple sequence alignment more space efficient in practice (extended abstract). In *Proceedings of the Sixth Annual Symposium on Combinatorial Pattern Matching*. Springer, Berlin.

Gusfield,D. (1997) Algorithms on strings: a dual view from computer science and computational molecular biology. In *Multiple String Comparison—The Holy Grail*. Cambridge University Press, Cambridge, Chapter 13.

Kececioglu,J.D. *et al*. (1995) Discussion theme: The MSA algorithm. http://www.techfak.uni-bielefeld.de/bcd/Lectures/kececioglu.html.

Myers,E.W. (1991) *An Overview of Sequence Comparison Algorithms in Molecular Biology.* Department of Computer Science, The University of Arizona, Tucson, TR 91-29.

Sankoff,D. (1972) Matching sequences under deletion-insertion constraints. *Proc. Natl Acad. Sci. USA*, **68**, 4–6.

Waterman,M.S. (1995) *Introduction to Computational Biology, Maps, Sequences and Genomes Interdisciplinary Statistics.* Chapman and Hall, London, Chapters 1 and 8–10.

Waterman,M.S., Joyce,J. and Eggert,M. (1991) Computer alignment of sequences. In Miyamoto,M.M. and Cracraft,J. (eds), *Phylogenetic Analysis of DNA Sequences.* Oxford University Press, Oxford, Chapter 4.