Universität Bielefeld
Forschungsschwerpunkt Mathematisierung —
Strukturbildungsprozesse

Materialien/Preprints XCIX

A General Method for Fast Multiple Sequence Alignment

Udo Tönges
Sören W. Perrey
Jens Stoye
Andreas W. M. Dress

Bielefeld 1996

Classification code(s) according to the
1980 Mathematics Subject Classification of
Mathematical Reviews:

92-04
92A10
68K05

Forschungsschwerpunkt Mathematisierung —
Strukturbildungsprozesse
Universität Bielefeld
Postfach 10 01 31
D-33501 Bielefeld

Telefon: (05 21) 1 06-47 64
Fax: (05 21) 1 06-60 07

e-mail:
fsp@mathematik.uni-bielefeld.de

A General Method for Fast Multiple Sequence Alignment

Udo Tönges[*], Sören W. Perrey[†], Jens Stoye[*], Andreas W. M. Dress[*]

Abstract

We have developed a fast heuristic algorithm for multiple sequence alignment which provides near-to-optimal results for sufficiently homologous sequences. The algorithm makes use of the standard dynamic programming procedure by applying it to all pairs of sequences. The resulting score matrices for pairwise alignment give rise to *secondary* matrices containing the additional charges imposed by forcing the alignment path to run through a particular vertex. Such a constraint corresponds to slicing the sequences at the positions defining that vertex, and aligning the remaining pairs of prefix and suffix sequences seperately. From these secondary matrices, one can compute – for any given family of sequences – suitable positions for cutting all of these sequences simultaneously, thus reducing the problem of aligning a family of n sequences of average length l in a *Divide and Conquer* fashion to aligning two families of n sequences of approximately half that length.

In this paper, we explain the method for the case of 3 sequences in detail, and we demonstrate its potential and its limits by discussing its behaviour for several test families. A generalization for aligning more than 3 sequences is lined out, and some actual alignments constructed by our algorithm for various user-defined parameters are presented.

Key words: dynamic programming; secondary matrix; divide and conquer; pairwise sequence alignment; multiple sequence alignment

[*]Research Center for Interdisciplinary Studies on Structure Formation (FSPM), University of Bielefeld, Postfach 10 01 31, D-33501 Bielefeld, Germany.
e-mail: {toenges,stoye,dress}@mathematik.uni-bielefeld.de

[†]Department of Mathematics, Massey University, Palmerston North, New Zealand. e-mail: S.W.Perrey@massey.ac.nz

1 Introduction

The construction of biologically plausible alignments for given families of DNA or protein sequences is one of the major problems in computational molecular biology. Consequently, the design and study of alignment procedures is presently a very active area of research, with an abundance of publications and software contributions (see [48], [34], [9], or [28] for a survey). While some papers (e.g. [27], [26], [8], [20], [1]) propose procedures to speed up the search for the "true" multiple sequence alignment, that is, the optimal alignment relative to some predefined optimality criterion (see below), others ([39], [17], [42], [23], [6]) – like the present one – propose computationally efficient heuristics for constructing good, though not necessarily "optimal" alignments.

An often discussed formalization of the multiple alignment problem is the following one: Given n sequences $s_1, s_2, .., s_n$ of lengths $l_1, l_2, .., l_n$, respectively, whose entries have been taken from a finite alphabet $\mathcal{A}$, and given a *weight function*

$$w \ : \ (\ \mathcal{A} \cup \{-\}\ )^n \ \longrightarrow \ \mathbf{R},$$

where "$-$" denotes the *gap letter* which is different from all letters of $\mathcal{A}$,
find an $n \times N$ matrix $M = (m_{kj})_{1 \le k \le n, 1 \le j \le N}$ for some $N \le \sum_{k=1}^{n} l_k$ with entries $m_{kj} \in \mathcal{A} \cup \{-\}$ that minimizes its *w-score*

$$w(M) := \sum_{j=1}^{N} w(m_{1j}, \ldots, m_{nj})$$

among all such matrices which present an *alignment* of the sequences s_1, s_2,...,s_n, that is, which do not contain any column consisting of gaps only and which, for each row $(m_{k1}, m_{k2}, \ldots, m_{kN})$, reproduce the sequence s_k upon eliminating all of its gap letters $(k = 1, 2, \ldots, n)$.

Weight functions often used in this context are *weighted sum-of-pairs* scores, defined by[1]

$$w(m_{1j}, m_{2j}, .., m_{nj}) \ := \ \sum_{k<k'} \delta(m_{kj}, m_{k'j}) \cdot \alpha_{k,k'},$$

where the parameters $\alpha_{k,k'}$ are sequence-dependent weights, and

$$\delta : (\mathcal{A} \cup \{-\})^2 \to \mathbf{R}$$

is a (real-valued) weight function, defined on all possible pairs of matrix entries (cf. [27], [13], [8]). Obviously, the optimization task is to elongate the given sequences by inserting gaps, bringing all of them up to the same length (denoted above by N) and, simultaneously, minimizing the sum over all "column weights".

[1] In the formula below, we charge for gaps at the begining or the end of a sequence the same way we charge for them anywhere inbetween two residues, and we assume additive gap penalties. There are simple remedies available if other charges are applied. They will be discussed in a separate paper.

It is well known that this problem can be solved optimally by *dynamic programming* (cf. [31], [48]). In our algorithm, we use this dynamic programming procedure by applying it to all pairs of sequences. The resulting score matrices for pairwise alignment give rise to *secondary* matrices $C(i_\nu, i_\mu)_{1\le i_\nu \le l_\nu, 1\le i_\mu \le l_\mu}$ containing the additional charges imposed by slicing the sequences s_ν and s_μ at the positions i_ν and i_μ, respectively, and aligning the remaining pairs of prefix and suffix sequences seperately.

Upon fixing one position where we want to cut one of our sequences – in general at or, at least, somewhere close to its midpoint –, we search for those positions for cutting the remaining $n-1$ sequences which correspond to a minimal (appropriately weighted, if neccessary) sum of the corresponding $\binom{n}{2}$ additional secondary charges.

This way, our original problem of aligning a family of n sequences of average length l can be reduced in a *Divide and Conquer* fashion to the problem of aligning two families of n sequences of approximately half that length. Consequently, applying this step iteratively will eventually result in a collection of multiple alignment problems involving rather short sequences only, which can then be handled by standard procedures.

A first outline of this method has recently been presented at the *Third International Conference on Intelligent Systems for Molecular Biology* in Cambridge (cf. [11]). Here, we give a detailed description of our procedure, and we present some test cases which allow to evaluate its potential and its limits. We focus in particular on the *quality* of the results as measured by the alignment score, and on the *time complexity* (measured in CPU time). We also study its dependence on several user-defined parameters, and on the input order. In an appendix, some actual alignments constructed by our algorithm are presented.

2 Method

We introduce the algorithm for the case of three sequences. There are several possibilities to generalize the method to more than three sequences, – a rather natural one will be mentioned later on, a more thorough discussion is in preparation.

In a way, our method is reminiscent of D.S. Hirschberg's proposal for calculating a shortest alignment path of two sequences using memory in the magnitude of only the length of one of the sequences (cf. [19]). Also, our method of calculating the degree of deviation from the path of the optimal alignment is similar to a procedure used in [43].

Let s_1, s_2 be two sequences with entries from a finite alphabet $\mathcal{A}$ and with sequence lengths $l_1 = l(s_1)$ and $l_2 = l(s_2)$, respectively, and let w denote a score function defined on pairs of sequences calculated by applying the standard (dynamic programming) alignment scheme. Then, the *additional charge* $C_{s_1,s_2}[i_1, i_2]$ imposed by slicing the sequences s_1 and s_2 at the positions i_1 and i_2, respectively, is defined

by[2]

$$C_{s_1,s_2}[\,i_1,i_2\,] \;:=\; w\,[\,s_1(\leq i_1), s_2(\leq i_2)\,] + w\,[\,s_1(> i_1\,), s_2(> i_2)\,] - w\,[\,s_1, s_2\,],$$

where $s_k(\leq i_k)$ denotes the (prefix) subsequence of s_k with indices running from 1 to i_k and $s_k(> i_k)$ denotes the (suffix) subsequence of s_k running from $i_k + 1$ to l_k $(k = 1, 2)$.

The matrix $C_{s_1,s_2} = (C_{s_1,s_2}[\,i_1,i_2\,])_{1\leq i_1\leq l_1, 1\leq i_2\leq l_2}$, we'll call the *secondary* matrix of the sequences s_1, s_2.

Note that there exists, for every i_1, at least one position $i_2(i_1)$ with

$$C_{s_1,s_2}[\,i_1, i_2(i_1)\,] = 0 :$$

If an optimal alignment of s_1 and s_2 matches the prefix sequence $s_1(\leq i_1)$ with $s_2(\leq i_2)$, then put $i_2(i_1) := i_2$. Obviously, once we know $i_2(i_1)$, we can find an optimal alignment of s_1 and s_2 by concatenating optimal alignments of $s_1(\leq i_1)$ and $s_2(\leq i_2)$ and of $s_1(> i_1)$ and $s_2(> i_2)$ which can be computed independently of each other.

Similarly, if an optimal multiple alignment of s_1, s_2, and s_3 matches the prefix sequence $s_1(\leq i_1)$ with $s_2(\leq i_2')$ and $s_3(\leq i_3')$, we can find an optimal multiple alignment of s_1, s_2, and s_3 by concatenating optimal alignments of $s_1(\leq i_1)$, $s_2(\leq i_2)$, and $s_3(\leq i_3)$ and of $s_1(> i_1)$, $s_2(> i_2)$, and $s_3(> i_3)$. Consequently, to construct an (almost) optimal multiple alignment of s_1, s_2, and s_3 by concatenation of (almost) optimal alignments of appropriately chosen prefix and suffix sequences, we try to estimate, for a given position i_1 of s_1 and any two positions i_2 and i_3 of s_2 and s_3, respectively, the additional charges imposed by concatenating an alignment of $s_1(\leq i_1)$, $s_2(\leq i_2)$, and $s_3(\leq i_3)$ as well as of $s_1(> i_1)$, $s_2(> i_2)$, and $s_3(> i_3)$.

To this end, we use the (weighted) sum of secondary charges over all projections $(i_1, i_2), (i_1, i_3)$, and (i_2, i_3) as such an estimate, that is, for any three sequences s_1, s_2, s_3 and any three integers i_1, i_2, i_3 with $0 \leq i_k \leq l(s_k)$ $(k = 1, 2, 3)$, we put

$$C(i_1, i_2, i_3) \;:=\; C_{s_1,s_2}[i_1, i_2] \cdot \alpha_{1,2} + C_{s_1,s_3}[i_1, i_3] \cdot \alpha_{1,3} + C_{s_2,s_3}[i_2, i_3] \cdot \alpha_{2,3},$$

where the coefficients $\alpha_{1,2}, \alpha_{1,3}$, and $\alpha_{2,3}$ are appropriately chosen weight factors reflecting e.g. phylogenetic relationship[3].

Next, we put $i_1 := \lceil l_1/2 \rceil$ (or close to this value), and use the secondary matrix $C_{s_2,s_3} = (C_{s_2,s_3}[x, y])_{1\leq x\leq l_2, 1\leq y\leq l_3}$ as well as the rows $(C_{s_1,s_2}[i_1, x])_{1\leq x\leq l_2}$ and $(C_{s_1,s_3}[i_1, y])_{1\leq y\leq l_3}$ of the secondary matrices C_{s_1,s_2} and C_{s_1,s_3}, respectively, to find

[2]Here, as above, we charge for gaps at the begining or the end of a sequence the same way we charge for them anywhere between two residues, and we assume additive gap penalties.

[3] How to choose weight factors appropriately for a given task is itself an important and much studied scientific topic (cf. [3], [35], [45]). In our context, it seems biologically plausible to give higher weights to the more similar pairs, as having them aligned optimally should be more important than aligning two fairly unrelated sequences optimally on the expense of worthening a good alignment of closely related sequences.

those positions i_2 and i_3 which minimize $C(i_1, i_2, i_3)$. These positions can, in principle, be found by exhaustive search over all i_2 $(0 \leq i_2 \leq l_2)$ and i_3 $(0 \leq i_3 \leq l_3)$. Of course, shortcuts can be used, the most obvious one being not to consider any position i_3 for those positions i_2, for which $C_{s_1,s_2}[i_1, i_2]$ is at least as large as the optimum found so far. The speed-up resulting from this simple device alone guarantees already that the overall time needed for finding the optimal cut positions i_2 and i_3 is a comparatively small fraction, only, of the – also quite small – total time needed by our algorithm, – so it is definitely not the bottleneck.

We expect these positions to either coincide with $i'_2(i_1)$ and $i'_3(i_1)$ or, at least, to define slicing positions such that the score of the resulting alignment is not much above the score of an optimal alignment of s_1, s_2, and s_3. The reason for this expectation is that aligning three sequences optimally, a compromise has to be found between the three, in general not compatible optimal pairwise alignments and that the value of $C(i_1, i_2, i_3)$ is a good measure for the total "expenses" of the compromise associated with aligning the sequences by concatenating optimal alignments of the three prefix and the three suffix sequences resulting from the associated cuts.

Based on this expectation, we iterate our procedure, that is, we replace the original multiple alignment problem by the two alignment problems posed by the three prefix sequences $s_1(\leq i_1), s_2(\leq i_2), s_3(\leq i_3)$ and by the three suffix sequences $s_1(> i_1), s_2(> i_2), s_3(> i_3)$, and we go on searching for suitable vertices in their alignment diagrams to cut them down further into still smaller sequence triples.

Note that, at this stage, the sequences under consideration have only been divided into several triples of subsequences but have not been aligned so far. This is simply done as follows: At some iteration step, we stop the dividing process and start using some score-optimal alignment procedure (e.g. standard dynamic programming, or faster versions of it, like `MSA`, cf. [25], [15]) to align the remaining subsequences.

At present, we have investigated combinations of the following two alternatives as a stopping criterion[4]:

- a threshold L for the shortest length of the three (sub)sequences under consideration, and
- a threshold D for the number of iteration steps of the dividing process (recursion depth).

Altogether, this leads to the following general procedure:

Algorithm *d&c-align* (s_1, s_2, s_3, L, D)

`If` $\min_{k\in\{0,1,2\}}\{l_k\} \leq L$ or $D = 0$
`then` return the optimal alignment of s_1, s_2, s_3;

[4] Other or additional criteria are also conceivable, e.g. one could start the score-optimal procedure once all three (sub)sequences under consideration are 'sufficiently homologous'. This in turn could be defined by some function depending on the length and the pairwise alignment scores of the three (sub)sequences.

`else` put $i_1 := \lceil l_0/2 \rceil$, and search for indices $i_2 \in \{0,..,l_1\}$ and $i_3 \in \{0,..,l_2\}$ which minimize $C(i_1,i_2,i_3)$;
return the Concatenation of $d\&c\text{-}align\,(\,s_1(\leq i_1), s_2(\leq i_2), s_3(\leq i_3), L, D-1\,)$
and $d\&c\text{-}align\,(\,s_1(> i_1), s_2(> i_2), s_3(> i_3), L, D-1\,)$;

3 Results

Our alignment algorithm has been applied to several sets of random and of biologically defined sequences. For defining the weight factors $\alpha_{i,j}$ $(i,j \in \{1,2,3\},\ i \neq j)$, we have simply used the optimal (distance) score of the pairwise alignment to define

$$\alpha_{i,j} := \frac{\min\{w(s_1,s_2), w(s_1,s_3), w(s_2,s_3)\}}{w(s_i,s_j)}\ ,$$

so that $\alpha_{i,j} \leq 1$ for all i,j and $\alpha_{i,j} = 1$ if and only if the pair of sequences (s_i,s_j) has the highest similarity (lowest alignment score) of all three possible pairs of sequences under consideration[3].

We have evaluated the quality of the resulting alignments by comparing the scores to the (formally) optimal scores (obtained by our procedure for $L \geq \min\{l_1,l_2,l_3\}$ or $D = 0$).

Figures 1 and 2 show the running time (CPU) in seconds versus $D = 0,1,\ldots,10$ and $L = 5,10,20,\ldots,100$, respectively. For variable D (Fig.1), we fixed $L := 3$ in order not to force the algorithm trying to cut our sequences more often than possible. For variable L (Fig.2), we fixed $D := 10$ in order not to stop the algorithm before the threshold L is reached.

The sequence triples considered are protein sequences (average length $l \approx 300$: tyrosine kinase, some actual alignments are given in the appendix; $l \approx 900$: glr1-human, glr2-human, glr3-human; $l \approx 1000$: P115-Bovin, P115-Rat, P115-Mychr) and DNA-sequences ($l \approx 1300$: human immuno deficiency virus: hiv1d044, hiv1d744, hiv1d868).

As expected, the CPU time needed by our procedure is much smaller than that needed by standard dynamic programming (that is, the value obtained for $L \geq \min(l_1,l_2,l_3)$ or $D := 0$). For increasing D with $D \leq 4$, the running time decreases rapidly and becomes almost constant, – in the magnitude of seconds, – for $D \geq 5$. Similarly, for increasing L with $L \leq 30$, the running time is almost constant while for $L \geq 30$ it also increases considerably.

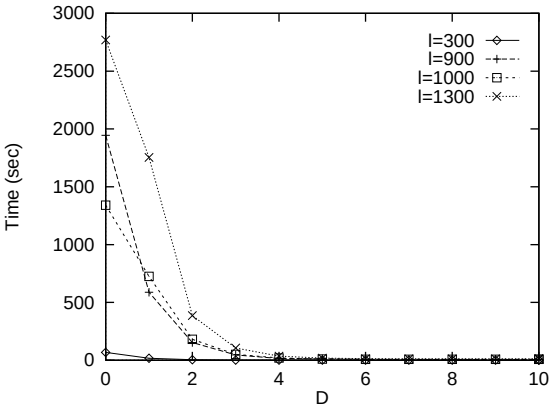

Figure 1

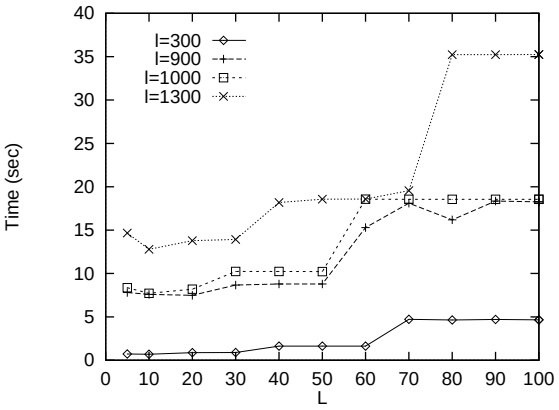

Figure 2

In Figures 3 and 4, the *relative scores*, obtained for various values of D (with fixed $L := 3$) and L (with fixed $D := 10$) are shown, that is, the percentage of the optimal score by which the score actually obtained by our algorithm exceeds the optimal one. The relative scores increase very slowly for increasing D with $D \leq 5$ and for decreasing L with $L \geq 30$.

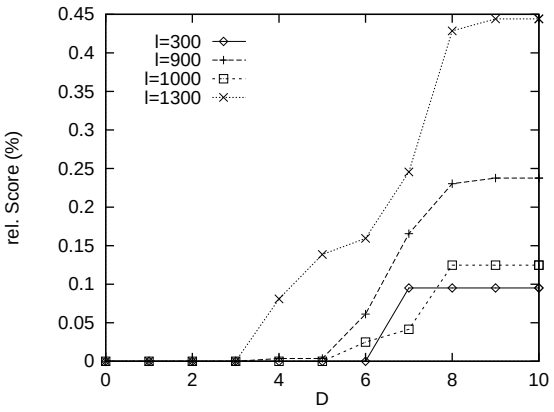

Figure 3

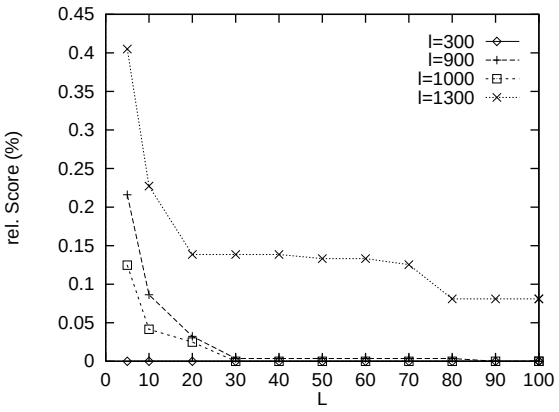

Figure 4

Together, these observations suggest that putting $L \approx 40$ and $D \gg 0$ will lead to a very fast alignment procedure with close to score-optimal results.

We also have investigated the sensitivity of our approach to the *input order* of the sequences. The running time and the scores seem to be basically independent of this order when the sequences under consideration have essentially the same degree of pairwise homology. If these degrees differ significantly, the input order has some more influence on the performance and quality of our procedure: in this case, it seems best to choose for s_1 the sequence exhibiting least homology relative to the other two sequences.

Finally, we have investigated whether the alignment scores improve upon relaxing the restriction imposed by fixing the first cut point i_1 on exactly $\lceil l_1/2 \rceil$. For some

small $\delta \in \mathbf{N}$ (e.g. $\delta = 5, \ldots, 25$), we simply search in each division step for that i_1 in the intervall

$$\lceil l_1/2 \rceil - \delta \leq i_1 \leq \lceil l_1/2 \rceil + \delta,$$

which leads to the smallest additional charge $C(i_1, i_2, i_3)$. As is shown in Figures 5 and 6, there is no substantial influence on running time and obtained relative score for those sequence triples we have investigated so far (here, we put $L := 40$ and $D := 10$, as suggested above).

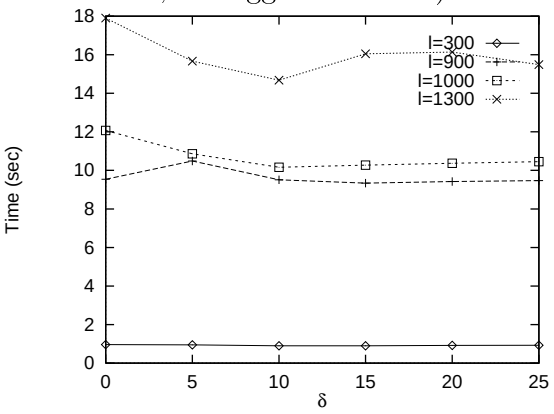

Figure 5

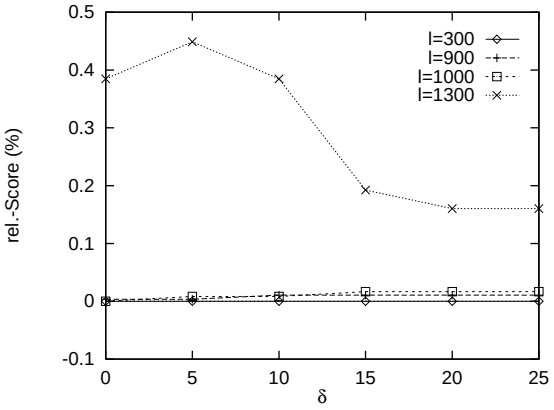

Figure 6

In the appendix, some actual alignments constructed by our algorithm are presented.

4 Discussion

It is standard practice to use dynamic programming for calculating a global (score-)optimal alignment of two sequences. The natural extension of this algorithm to multiple alignment is limited to small numbers of relatively short sequences because the search space increases exponentially with the number of sequences under consideration (cf. [25]). Clearly, pairwise alignments (so-called 'projections') can be used for cutting down the computational complexity of the (standard) dynamic programming procedure (cf. [8], [25]). Similarly, secondary matrices resulting from imposing specific vertices to be traversed by the path through the dynamic programming graph have also been considered before to reduce the amount of memory required (cf. [19], [29], [20]).

Yet, the use of secondary matrices of the projections of the multiple alignment problem in order to presort simultaneously every sequence under consideration into several subsequences in a "Divide and Conquer" fashion appears to be a novel approach.

Recently, it was shown that – not unexpectedly – multiple sequence alignment with the *sum-of-pairs* score is NP-complete (cf. [47]). Therefore, to align large-size sets of sequences in reasonable time, one needs fast heuristic algorithms. Unfortunately, most of the more reliable heuristic approaches suffer from high computational

costs for large-size problems, while fast heuristics often do not yield plausible results. So, there is some pressure for developing fast and, simultaneously, sufficiently reliable heuristics.

In this context, our algorithm offers the following advantages:

- It is **very fast**. Even when the input sequences are rather long, the user can specify the thresholds L and D appropriately in order to cut down the running time by orders of magnitudes, that is, down to just seconds.
- It results in **near-to-optimal alignments** as measured by the alignment score.
- It uses **memory** of about the magnitude needed for pairwise alignment even if the multiple alignment itself (and not only its score) is calculated.

In addition, there exist several ways to *generalize* our algorithm to a multiple alignment procedure for more than 3 sequences. One natural and simple extension of the method outlined above is to compute all pairwise secondary matrices and the additional charge function

$$C(i_1, \ldots, i_n) \;:=\; \sum_{1 \le k < k' \le n} C_{s_k, s'_k}[i_k, i_{k'}] \cdot \alpha_{k,k'}.$$

Using a small (sub)sequence length threshold L or a high recursion depth threshold D, the size of the hypercubes to be solved by dynamic programming (or `MSA`) can always be reduced to manageable dimensions. In addition, if a large number of sequences is to be considered, the design of clever branch and bound methods for speeding up the search for appropriate slicing points will become more and more important.

Most multiple alignment procedures align the sequences in a hierarchical order by first pre-clustering the sequences and then aligning them – or profiles derived from them – recursively (cf. [42]). To obtain a biologically reasonable alignment, these clusters should actually reflect phylogenetic relationships between the sequences in question. Yet, most algorithms for reconstructing phylogenies need aligned sequences (or distances between the sequences, derived from alignments) as their input.

In other words, (multiple) sequence alignment and the reconstruction of phylogenetic relationships require each other. Consequently, it has been argued that both should be done in synchrony (cf. [16]). As branching points in trees always need exactly three leaves (or other branching points constructed already) to be specified as the *median* (cf. [7]) of those tree vertices, our method – even in its present state of infancy – might be rather useful in this context (cf. [33], [46]).

Acknowledgements

The authors wish to thank the referees for helpful constructive comments on an earlier version of this paper. This research is supported by the German Ministry for Research and Technology (BMBF) under grant number 01 IB 301 B4.

Appendix

Finally, we present some actual alignments constructed by our algorithm.

The first example shows alignments of three tyrosine kinase sequences (of lengths $l_1 = 273, l_2 = 280, l_3 = 280$), obtained with thresholds $L = 20, 10, 7, 5$ ($D = 10$ fixed), whereby – as we know from running our algorithm with $L \geq 273$ (or $D = 0$) – the (score-)optimal alignment is obtained for $L = 20$. In order to compare them easily, we present the alignments blockwise. Above the alignments, the dividing step numbers are printed, and the positions different from the optimal alignment are marked by "$*$" below the alignment. Note that there are only three regions (of length 4 or 7) where the fastest suboptimal version ($L = 5$) differs from the optimal one. In particular, all regions of a suboptimal alignment obtained for some threshold L have been aligned in exactly the same way by all versions L' with $L' < L$. This can be used to select the (most probably) correctly aligned regions from a set of suboptimal alignments constructed by fast versions (L very small) of our procedure.

We have observed that the alignments obtained for small L differ from the optimal alignment mainly in rather short regions around gaps. This suggest a windowing approach where, for a sufficiently small window around gaps, an optimal alignment procedure is applied. We will discuss this idea in a later paper.

As score matrix, we have used PAM250 (with integer entries between 0 and 25), homogeneous gap penalties of value -8, and gap-to-gap penalties of value 0. In the following table, the CPU time, the relative score, and the number of alignment positions different from the optimal alignment are given. The CPU time is measured in seconds on a Silicon Graphics computer (CPU: MIPS R4000, 100 MHZ, 64 MB memory, 1 MB cache), and the memory needed is about 250 KB. Note that standard dynamic programming, obtained by our method for $L \geq 273$ (or $D := 0$), needs 93.41 seconds CPU time on this computer. The commonly used so-called `MSA` algorithm needs 7.56 seconds (cf. [8]).

We also have compared our alignments with those constructed by Clustal W and by Maximum Weight Trace (cf. [42], [23]). Most positions of these alignments are identical to the score-optimal and almost all of those are contained in long stretches without gaps. Consequently, once again, most differences occur around gaps.

L	5	7	10	20	60	300
CPU time (in seconds)	0.32	0.34	0.39	0.59	1.40	93.41
absolute score	7008	7011	7015	7016	7016	7016
number of non-optimal positions	17	12	5	0	0	0

```
                          3                2                  3
     ---GLAKDA--WEIPRESLRLEAKLGQGCFGEVWMGTWND-TTRVAI-KTLKPGTMSPEA
L=20 TIYGVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGVWKKYSLTVAV-KTLKEDTMEVEE
     ---S-S-YY--WKMEASEVMLSTRIGSGSFGTVYKGKWHG-DVAVKILKVVDPTPEQLQA

                4        3       4        2         4        3
     ---GLAKDA--WEIPRESLRLEAKLGQGCFGEVWMGTWND-TTRVAI-KTLKPGTMSPEA
L=10 TIYGVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGVWKKYSLTVAV-KTLKEDTMEVEE
     ---S-S-YY--WKMEASEVMLSTRIGSGSFGTVYKGKWHG-DVAVKILKVVDPTPEQLQA

                4   5    3   5   4   5    2   5     4   5    3
     ---GLAKDA--WEIPRESLRLEAKLGQGCFGEVWMGTWNDTTRV-AI KTLKPGTMSPEA
L=7  TIYGVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGVWKKYSLTVAV KTLKEDTMEVEE
     ---S-S-YY--WKMEASEVMLSTRIGSGSFGTVYKGKWHGDVAVKIL KVVDPTPEQLQA
                                          *******
              5    4   5    3   5   4   5    2   5     4   5    3
     ---GLA-KDA-WEIPRESLRLEAKLGQGCFGEVWMGTWNDTTRV-AI KTLKPGTMSPEA
L=5  TIYGVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGVWKKYSLTVAV KTLKEDTMEVEE
     ---S-S-YY--WKMEASEVMLSTRIGSGSFGTVYKGKWHGDVAVKIL KVVDPTPEQLQA
           ****                           *******


                    1                  3                2
     FLQEAQVMKKLRHEKLVQLYAVVSEEP-IYIVIEYMSKGSLLDFLKGEMGKYLRLPQLVD
L=20 FLKEAAVMKEIKHPNLVQLLGVCTREPPFYIITEFMTYGNLLDYLRECNRQEVSAVVLLY
     FRNEVAVLRKTRHVNILLFMGYMTKDN-LAIVTQWCEGSSLYKHLHVQETKF-QMFQLID

           4        1       4         3       4        2       4
     FLQEAQVMKKLRHEKLVQLYAVVSEEP-IYIVIEYMSKGSLLDFLKGEMGKYLRLPQLVD
L=10 FLKEAAVMKEIKHPNLVQLLGVCTREPPFYIITEFMTYGNLLDYLRECNRQEVSAVVLLY
     FRNEVAVLRKTRHVNILLFMGYMTKDN-LAIVTQWCEGSSLYKHLHVQETKF-QMFQLID

       5   4   5    1   5   4    5    3   5   4   5    2       4
     FLQEAQVMKKLRHEKLVQLYAVVSEEP-IYIVIEYMSKGSLLDFLKGEMGKYLRLPQLVD
L=7  FLKEAAVMKEIKHPNLVQLLGVCTREPPFYIITEFMTYGNLLDYLRECNRQEVSAVVLLY
     FRNEVAVLRKTRHVNILLFMGYMTKDN-LAIVTQWCEGSSLYKHLHVQETKF-QMFQLID

       5   4   5    1   5   4    5    3   5   4   5    2   5   4
     FLQEAQVMKKLRHEKLVQLYAVVSEEP-IYIVIEYMSKGSLLDFLKGEMGKYLRLPQLVD
L=5  FLKEAAVMKEIKHPNLVQLLGVCTREPPFYIITEFMTYGNLLDYLRECNRQEVSAVVLLY
     FRNEVAVLRKTRHVNILLFMGYMTKDN-LAIVTQWCEGSSLYKHLHVQETKF-QMFQLID
```

```
           3                0                3                2
     MAAQIASGMAYVERMNYVHRDLRAANILVGENLVCKVADFGLARLIED-NEYTARQGAKF
L=20 MATQISSAMEYLEKKNFIHRDLAARNCLVGENHLVKVADFGLSRLMTG-DTYTAHAGAKF
     IARQTAQGMDYLHAKNIIHRDMKSNNIFLHEGLTVKIGDFGLATVKSRWSGSQQVEQPTG

           3       4        0       4        3        4        2
     MAAQIASGMAYVERMNYVHRDLRAANILVGENLVCKVADFGLARLIED-NEYTARQGAKF
L=10 MATQISSAMEYLEKKNFIHRDLAARNCLVGENHLVKVADFGLSRLMTG-DTYTAHAGAKF
     IARQTAQGMDYLHAKNIIHRDMKSNNIFLHEGLTVKIGDFGLATVKSRWSGSQQVEQPTG

       5    3   5   4   5    0   5   4   5    3   5    4   5    2
     MAAQIASGMAYVERMNYVHRDLRAANILVGENLVCKVADFGLARLIED-NEYTARQGAKF
L=7  MATQISSAMEYLEKKNFIHRDLAARNCLVGENHLVKVADFGLSRLMTG-DTYTAHAGAKF
     IARQTAQGMDYLHAKNIIHRDMKSNNIFLHEGLTVKIGDFGLATVKSRWSGSQQVEQPTG

       5    3   5   4   5    0   5   4   5    3   5    4   5    2
     MAAQIASGMAYVERMNYVHRDLRAANILVGENLVCKVADFGLARLIED-NEYTARQGAKF
L=5  MATQISSAMEYLEKKNFIHRDLAARNCLVGENHLVKVADFGLSRLMTG-DTYTAHAGAKF
     IARQTAQGMDYLHAKNIIHRDMKSNNIFLHEGLTVKIGDFGLATVKSRWSGSQQVEQPTG



                      3                1                3
     PIKWTAPEAA-LYG-R-FTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERGY-
L=20 PIKWTAPESL-AYN-K-FSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKDY-
     SVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRGYA

           4           3       4        1       4         3
     PIKWTAPEAA-LYG-R-FTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERGY-
L=10 PIKWTAPESL-AYN-K-FSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKDY-
     SVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRGYA

        5   4     5     3   5   4   5    1       4   5     3   5
     PIKWTAPEAA-L-YGR-FTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERGY-
L=7  PIKWTAPESL-A-YNK-FSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKDY-
     SVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRGYA

        5   4     5     3   5   4   5    1   5   4   5     3   5
     PIKWTAPEAA-L-YGR-FTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERGY-
L=5  PIKWTAPESL-A-YNK-FSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKDY-
     SVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRGYA
```

```
                  2                 3
     R--MPCP-PECPESLHDLMCQCWRKDPEERPTFKYLQAQLLPA-CVLEVAE
L=20 R--MERP-EGCPEKVYELMRACWQWNPSDRPSFAEIHQAFETM-FQESSIS
     SPDLSRLYKNCPKAIKRLVADCVKKVKEERPLFPQILSSIELLQHSLPKIN

          4         2       4         3         4
     R--MPCP-PECPESLHDLMCQCWRKDPEERPTFKYLQAQL-LPACVLEVAE
L=10 R--MERP-EGCPEKVYELMRACWQWNPSDRPSFAEIHQAF-ETMFQESSIS
     SPDLSRLYKNCPKAIKRLVADCVKKVKEERPLFPQILSSIELLQHSLPKIN
                                            ****
          4    5    2   5   4   5    3   5     4   5
     R--MPCP-PECPESLHDLMCQCWRKDPEERPTFKYLQAQL-LPACVLEVAE
L=7  R--MERP-EGCPEKVYELMRACWQWNPSDRPSFAEIHQAF-ETMFQESSIS
     SPDLSRLYKNCPKAIKRLVADCVKKVKEERPLFPQILSSIELLQHSLPKIN
                                            ****
          4    5    2   5   4   5    3   5     4   5
     R--MPCP-PECPESLHDLMCQCWRKDPEERPTFKYLQAQL-LPACVLEVAE
L=5  R--MERP-EGCPEKVYELMRACWQWNPSDRPSFAEIHQAF-ETMFQESSIS
     SPDLSRLYKNCPKAIKRLVADCVKKVKEERPLFPQILSSIELLQHSLPKIN
                                            ****
```

Our second example shows results obtained by applying our procedure with $L = 5$, 10, 20, 40, 60, and 120 to three ATPases of lengths 460, 480, and 509, respectively, also aligned in [26] (ATPase β chain - Escherichia coli, ATPase β chain - bovine mitochondria, ATPase α chain - bovine mitochondria). All other parameters are set to the same values as in the previous example. The following table shows the running times and score values of the resulting alignments. Note that the last column shows results of the "classical" dynamic programming procedure for three sequences, as it is performed by our procedure with $L = 600$. In contrast, `MSA` only needs 12.95 seconds, yielding the same alignment. So, it seems desirable to combine our algorithm with the ideas and procedures used in `MSA`, – a task we intend to work on in the near future.

L	5	10	20	40	60	120	600
CPU time (in seconds)	0.45	0.54	0.87	2.14	7.21	27.20	402.56
absolute score	11439	11444	11445	11449	11449	11455	11455

References

[1] L. Allison. A Fast Algorithm for the Optimal Alignment of Three Strings. *J. theor. Biol.*, 164:261–269, 1993.

[2] S. F. Altschul. Gap Costs for Multiple Sequence Alignment. *J. theor. Biol.*, 138:297–309, 1989.

[3] S. F. Altschul, R. J. Carroll, and D. J. Lipman. Weights for Data Related by a Tree. *J. Mol. Biol.*, 207:647–653, 1989.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *J. Mol. Biol.*, 215:403–410, 1990.

[5] S. F. Altschul and D. J. Lipman. Trees, Stars, and Multiple Biological Sequence Alignment. *SIAM J. Appl. Math.*, 49(1):197–209, 1989.

[6] V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation Algorithms for Multiple Sequence Alignment. In Crochemore, M. and Gusfield, D., editor, *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, CA, USA, June 5-8, 1994. Proceedings*, number 807 in Lecture Notes in Computer Science, pages 43–53, Berlin, 1994. Springer Verlag.

[7] H.-J. Bandelt and J. P. Barthelemy. Medians in Median Graphs. *Discrete Applied Mathematics*, 8:131–142, 1984.

[8] H. Carrillo and D. Lipman. The Multiple Sequence Alignment Problem in Biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.

[9] S. C. Chan, A. K. C. Wong, and D. K. Y. Chiu. A Survey of Multiple Sequence Comparison Methods. *Bull. Math. Biol.*, 54(4):563–598, 1992.

[10] R. F. Doolittle, editor. *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*. Academic Press, Inc., San Diego, CA, USA, 1990.

[11] A. W. M. Dress, G. Füllen, and S. W. Perrey. A Divide and Conquer Approach to Multiple Alignment. In *Proc. of the Third Conference on Intelligent Systems for Molecular Biology, ISMB 95*, pages 107–113. AAAI Press, Menlo Park, CA, USA, 1995.

[12] D.-F. Feng and R. F. Doolittle. Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees. *J. Mol. Evol.*, 25:351–360, 1987.

[13] O. Gotoh. Alignment of Three Biological Sequences with an Efficient Traceback Procedure. *J. theor. Biol.*, 121:327–337, 1986.

[14] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile Analysis: Detection of Distantly Related Proteins. *Proc. Natl. Acad. Sci. USA*, 84(13):4355–4358, 1987.

[15] S. K. Gupta, J. D. Kececioglu, and A. A. Schäffer. Improving the Practical Space and Time Efficiency of the Shortest-Paths Approach to Sum-of-Pairs Multiple Sequence Alignment. *J. Comp. Biol.*, 2(3):459–472, 1995.

[16] J. Hein. Unified Approach to Alignment and Phylogenies. In Doolittle, R. F., editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 39, pages 626–645. Academic Press, Inc., San Diego, CA, USA, 1990.

[17] D. G. Higgins and P. M. Sharp. CLUSTAL: A Package for Performing Multiple Sequence Alignment on a Microcomputer. *Gene*, 73:237–244, 1988.

[18] M. Hirosawa, M. Hoshida, M. Ishikawa, and T. Toya. MASCOT: Multiple Alignment System for Protein Sequences Based on Three-Way Dynamic Programming. *CABIOS*, 9(2):161–167, 1993.

[19] D. S. Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[20] X. Huang. Alignment of Three Sequences in Quadratic Space. *Applied Computing Review*, 1(2):7–11, 1993.

[21] M. S. Johnson and R. F. Doolittle. A Method for the Simultaneous Alignment of Three or More Amino Acid Sequences. *J. Mol. Evol.*, 23:267–278, 1986.

[22] S. Karlin, M. Morris, G. Ghandour, and M.-Y. Leung. Efficient Algorithms for Molecular Sequence Analysis. *Proc. Natl. Acad. Sci. USA*, 85:841–845, 1988.

[23] J. Kececioglu. The Maximum Weight Trace Problem in Multiple Sequence Alignment. In Apostolico, A. and Crochemore, M. and Galil, Z. and Manber, U., editor, *4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993. Proceedings*, number 684 in Lecture Notes in Computer Science, pages 106–119, Berlin, 1993. Springer Verlag.

[24] U. Lessel and D. Schomburg. Similarities Between Protein 3D Structures. *Protein Engng.*, 7(10):1175–1187, 1994.

[25] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A Tool for Multiple Sequence Alignment. *Proc. Natl. Acad. Sci. USA*, 86:4412–4415, 1989.

[26] M. Murata. Three-Way Needleman-Wunsch Algorithm. In Doolittle, R. F., editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 22, pages 365–375. Academic Press, Inc., San Diego, CA, USA, 1990.

[27] M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous Comparison of Three Protein Sequences. *Proc. Natl. Acad. Sci. USA*, 82:3073–3077, 1985.

[28] E. W. Myers. An Overview of Sequence Comparison Algorithms in Molecular Biology. Technical Report TR 91-29, University of Arizona, Tucson, Department of Computer Science, 1991.

[29] E. W. Myers and W. Miller. Optimal Alignments in Linear Space. *CABIOS*, 4(1):11–17, 1988.

[30] D. Naor and D. L. Brutlag. On Near-Optimal Alignments of Biological Sequences. *J. Comp. Biol.*, 1(4):349–366, 1994.

[31] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[32] W. R. Pearson and D. J. Lipman. Improved Tools for Biological Sequence Comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448, 1988.

[33] D. Sankoff, R. J. Cedergren, and G. LaPalme. Frequency of Insertion-Deletion, Transversion, and Transition in the Evolution of 5S Ribosomal RNA. *J. Mol. Evol.*, 7:133–149, 1976.

[34] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, USA, 1983.

[35] P. R. Sibbald and P. Argos. Weighting Aligned Protein or Nucleic Acid Sequences to Correct for Unequal Representation. *J. Mol. Biol.*, 216:813–818, 1990.

[36] T. F. Smith and M. S. Waterman. Comparison of Biosequences. *Adv. Appl. Math.*, 2:482–489, 1981.

[37] T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[38] W. R. Taylor. Multiple Sequence Alignment by a Pairwise Algorithm. *CABIOS*, 3:81–87, 1987.

[39] W. R. Taylor. A Flexible Method to Align Large Numbers of Biological Sequences. *J. Mol. Evol.*, 28:161–169, 1988.

[40] W. R. Taylor. Motif-biased Protein Sequence Alignment. *J. Comp. Biol.*, 1(4):297–310, 1994.

[41] W. R. Taylor and K. Hatrick. Compensating Changes in Protein Multiple Sequence Alignments. *Protein Engng.*, 7(3):341–348, 1994.

[42] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice. *Nucl. Acids Res.*, 22(22):4673–4680, 1994.

[43] M. Vingron and P. Argos. Determination of Reliable Regions in Protein Sequence Alignments. *Protein Engng.*, 3(7):565–569, 1990.

[44] M. Vingron and P. Argos. Motif Recognition and Alignment for Many Sequences by Comparison of Dot-Matrices. *J. Mol. Biol.*, 218:33–43, 1991.

[45] M. Vingron and P. R. Sibbald. Weighting in Sequence Space: A Comparison of Methods in Terms of Generalized Sequences. *Proc. Natl. Acad. Sci. USA*, 90:8777–8781, 1993.

[46] M. Vingron and A. von Haeseler. Towards Integration of Multiple Alignment and Phylogenetic Tree Construction. Arbeitspapiere 852, GMD, 1994.

[47] L. Wang and T. Jiang. On the Complexity of Multiple Sequence Alignment. *J. Comp. Biol.*, 1(4):337–348, 1994.

[48] M. S. Waterman. *Introduction to Computational Biology. Maps, Sequences and Genomes.* Chapman & Hall, London, UK, 1995.

[49] M. S. Waterman and M. D. Perlwitz. Line Geometries for Sequence Comparisons. *Bull. Math. Biol.*, 48(4):567–577, 1984.

[50] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some Biological Sequence Metrics. *Adv. Math.*, 20:367–387, 1976.

[51] M. S. Waterman and M. Vingron. Sequence Comparison Significance and Poisson Approximation. *Statistical Science*, 9(3):367–381, 1994.