Fast Approximation to the NP-hard Problem of Multiple Sequence Alignment

Sören W. Perrey
Mathematics Department
Massey University, Private Bag 11-222, Palmerston North, New Zealand
e-mail: S.W.Perrey@massey.ac.nz

Jens Stoye
Research Center for Interdisciplinary Studies on Structure Formation (FSPM)
University of Bielefeld, Postfach 10 01 31, D-33501 Bielefeld, Germany
e-mail: stoye@mathematik.uni-bielefeld.de

Abstract

The study and comparison of several sequences of characters from a finite alphabet is relevant to various areas of science, in particular molecular biology. It has been shown that multiple sequence alignment with the *sum-of-pairs score* is NP-hard. Recently a fast heurstic method was proposed based on a *Divide-and-Conquer* technique. Recursively, all sequences were cut at some suitable positions. Eventually, the sets of subsequences were aligned optimally. In general, the (time) complexity of searching for good cutting points is $O(l^n)$ (n the number and l the maximal length of the sequences involved). By a simple $(n \cdot l)$-time technique, the base l was reduced, leading to a reasonable fast alignment algorithm for up to $n = 7$ and $l \approx 500$.

We refine the base-reducing technique by spending computational time quadratic in n (and still linear in l). This improves the alignment procedure regarding the number of sequences managable up to $n = 9$ (of same length l). Moreover, we present two natural extensions of this technique. One is an iterative application of a $(n^2 l)$-time technique and therefore still of that complexity. The other needs time $O(n^2 l^{s+1})$, where s is the number of sequences simultaneously considered during a minimization procedure.

Key words: dynamic programming; secondary matrix

1 Introduction

A fundamental problem in molecular biology is the construction of biologically plausible alignments of a given family of DNA or protein sequences. Consequently, the design and study of alignment procedures is presently a very active area of research, with an abundance of papers and software contributions (see [28], [6], or [22] for a survey). Recently, it was shown that multiple sequence alignment with the *sum-of-pairs* score, on which we will focus, is NP-complete (cf. [38]). 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, but sufficiently reliable heuristics.

In this section, we assume $s_1, s_2, .., s_n$ to be sequences of (finite) lengths $l_1 = l(s_1), l_2 = l(s_2), \ldots, l_n = l(s_n)$ respectively, whose entries have been taken from a finite alphabet $\mathcal{A}$, and $'-'$ to denote the *gap letter* not contained in $\mathcal{A}$.
The multiple alignment problem can be formalized as follows:

Definitions

1. An *alignment* of $s_1, \ldots, s_n$ is a matrix

$$M = (m_{pj})_{1 \le p \le n, 1 \le j \le N_M}$$

where $N_M \in \mathbf{N}$ $(\max\{l_p \mid p = 1, \ldots, n\} \le N_M \le \sum_{p=1}^{n} l_p)$ and, for every $p = 1, \ldots, n$ and every $j = 1, \ldots, N_M$

(i) $m_{pj} \in \mathcal{A} \cup \{-\}$,

(ii) $(m_{p1}, m_{p2}, \ldots, m_{pN_M}) \,/\, \{\, m_{pk} | m_{pk} = '-' \,\} \;\; = s_p$,

(iii) $\{m_{1j}, m_{2j}, \ldots, m_{nj}\} \cap \mathcal{A} \neq \emptyset$.

2. Let $w : (\mathcal{A} \cup \{-\})^n \longrightarrow \mathbf{R}$ be a (real-valued) *weight function.* The *w-score* of an alignment M is defined by

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

3. The *multiple sequence comparison problem* is to calculate

$$w_{OP} := \min\{w(M) \mid M \text{ is an alignment of } s_1, \ldots, s_n\}.$$

The *multiple sequence alignment problem* is to calculate an alignment M with $w(M) = w_{OP}$. Such an alignment is called *optimal.*

4. An *alignment graph* of $s_1, \ldots, s_n$ is a directed graph $G = (V, E)$ with vertex set $V := \{ (v_1, \ldots, v_n) \mid v_p \in \{0, 1, \ldots, l_p\} ; \; p = 1, \ldots, n \}$ and edge set E defined by

$$[(v_1, .., v_n), (w_1, .., w_n)] \in E \quad \Longleftrightarrow$$
$$v_p \leq w_p \leq v_p + 1, \; \forall \; p = 1, .., n \,, \text{ and } \textstyle\sum_{p=1}^n v_p \neq \sum_{p=1}^n w_p.$$

5. An *alignment path* is a (directed) path P from the unique source $s := (0, \ldots, 0)$ of G to the unique sink $t := (l_1, \ldots, l_n)$ of G.

Obviously the number of edges at each alignment path is at least $\max_p\{l_p\}$ and at most $\sum_{p=1}^n l_p$. There is a one-to-one correspondence between alignment paths and multiple alignments of the sequences involved.

Lemma 1 Let $\mathcal{M} = \{M = (m_{pj})_{1 \leq p \leq n, 1 \leq j \leq N_M}\}$ be the set of all possible alignments and $\mathcal{P} = \{P = ([(v_{1j}, \ldots, v_{nj}), (w_{1j}, \ldots, w_{nj})])_{1 \leq j \leq N_P}\}$ be the set of all possible alignment paths.
Then, $M \in \mathcal{M}$ corresponds to $P \in \mathcal{P}$ if and only if

(i) $N_M = N_P \; (=: N)$ and

(ii) $\forall \, p \in \{1, \ldots, n\}, \; \forall \, j \in \{1, \ldots, N\}$

$$\begin{array}{lll} (1) \;\; w_{pj} = v_{pj} & \Longleftrightarrow & m_{pj} = '-' \\ (2) \;\; w_{pj} = v_{p+1,j} & \Longleftrightarrow & m_{pj} \neq '-'. \end{array}$$

Defining lengths for the edges of the alignment graph by using the weight function w on the corresponding columns of an alignment M, the total length of an alignment path coincides with the w-score of the associated alignment. Consequently, the task to compute the length of a shortest path in G from s to t is equivalent to the multiple sequence comparison alignment problem.
It is well known that the multiple comparison problem, in principle, can be solved optimally by the so-called *dynamic programming* procedure (cf. [24], [26],[29]) with a computational space and time complexity proportional to $\prod_{p=1}^n (l_p + 1)$, resp. $(2^n - 1) \cdot \prod_{p=1}^n (l_p + 1)$. The space complexity can be reduced by one order, at least for the case of two and three sequences [15],[21],[16], [12], while, in general, the order of the time complexity could not be reduced. But some regions of the whole alignment graph can be cut off using pairwise alignments because the optimal multiple alignment path cannot

go through these regions [5],[19]. To find an optimal alignment (a shortest path) itself, a simple backtrack procedure can be applied.

The following scoring function is commonly used (cf. [20],[10],[5]).

Definition 6 Let $\alpha_{pq} \in \mathbf{R}$ $(1 \leq p < q \leq n)$ be sequence-dependent weights and, $d : (\mathcal{A} \cup '-')^2 \to \mathbf{R}$ a score function, defined on all possible pairs of matrix entries.

The *weighted sum-of-pairs* score is defined by

$$D(m_{1j}, m_{2j}, .., m_{nj}) = \sum_{1 \leq p < q \leq n} \alpha_{pq} \cdot d(m_{pj}, m_{qj}) \qquad \forall\, j \in \{1, \ldots, N\}.$$

2 Divide-and-Conquer Algorithm

In this section we present the *Divide-and-Conquer* approach to multiple sequence alignment developed by A.W.M. Dress et al., by recalling the introduction given in [8] and [35].

Let d denote a (distance) score function defined on pairwise alignments calculated from applying the standard dynamic programming scheme. [1]

Then, for all pairs of sequences s_p, s_q $(1 \leq p < q \leq n)$ the entries of the *secondary matrix* containing the additional charges $C_{s_p,s_q}[i_p, i_q]$ imposed by forcing the alignment path to run through the vertex (i_p, i_q) $(0 \leq i_\mu \leq l_\mu;\ \mu = p, q)$ are defined by

$$C_{s_p,s_q}[i_p, i_q] := d(s_p(\leq i_p), s_q(\leq i_q)) + d(s_p(> i_p), s_q(> i_q)) - d(s_p, s_q),$$

where $s_p(\leq i_p)$ denotes the prefix subsequence of s_p with indices running from 0 to i_p and $s_p(> i_p)$ denotes the suffix subsequence of s_p running from $i_p + 1$ to l_p $(p = 1, \ldots, n)$ (cf. [15]). Obviously, each secondary matrix fulfills the following properties:

- $C_{s_p,s_q}[i_p, i_q] \geq 0$ for all i_p, i_q;
- $C_{s_p,s_q}[i_p, i_q] = 0$, if and only if (i_p, i_q) lies on an optimal alignment path of s_p and s_q; and
- for all vertices $i_p \in \{0, \ldots, l_p\}$ there exists a vertex $i_q \in \{0, \ldots, l_q\}$, such that

$$C_{s_p,s_q}[i_p, i_q] = 0.$$

[1]Typically, a $|\mathcal{A}| \times |\mathcal{A}|$ scoring matrix containing scores for each possible substitution together with some gap penalty function g, is used. For simplicity Dress et al. focused on the simple so-called homogeneous gap penalty functions $g_k : \mathbf{N} \to \mathbf{R}^{<0}$ $(k \in \mathbf{R}^{>0})$, penalizing a gap of length n by $-n \cdot k$. To include more sophisticated gap penalties is discussed in [31].

Calculation of the additional charge imposed by forcing the *multiple* alignment path of n sequences through a particular vertex $(i_1, \ldots, i_n)$ in the whole alignment graph associated with the corresponding alignment problem, would have the same space and time complexity as *Dynamic Programming.* Therefore, they used an *estimate* $C(i_1, \ldots, i_n)$ of this (multi-dimensional) additional charge, which was defined by the (weighted) sum of secondary charges over all projections (i_p, i_q) $(p \neq q)$, that is, for any n integers $i_1, \ldots, i_n$ with $0 \leq i_p \leq l(s_p)$ $(p = 1, \ldots, n)$,

$$C(i_1, \ldots, i_n) \ := \sum_{1 \leq p < q \leq n} \alpha_{pq} \cdot C_{s_p, s_q}[i_p, i_q],$$

where the weight factors α_{pq} are calculated by the optimal (distance) score of the pairwise alignments

$$\alpha_{ij} := \frac{\max\limits_{1 \leq p < q \leq n} \{d[s_p, s_q]\}}{d[s_i, s_j]}.$$

Thereby, more similar pairs of sequences (i.e., pairs with small distance score d) get higher weight factors in order to align them closer to their optimal pairwise alignment than less similar pairs.

Hence, putting the *slicing point* $i_1 := \lceil l_1/2 \rceil$ (or close to this value), all $\binom{n-1}{2}$ secondary matrices

$$C_{s_p, s_q} = \Big(C_{s_p, s_q}[x, y] \Big)_{1 \leq x \leq l_p, 1 \leq y \leq l_q} \qquad (2 \leq p < q \leq n)$$

as well as the row

$$\Big(C_{s_1, s_q}[i_1, x] \Big)_{1 \leq x \leq l_q}$$

of each of the $n-1$ remaining secondary matrices, were used to find those values for the $i_2, \ldots, i_n$, that is *slicing points* for the other sequences, which minimize $C(i_1, \ldots, i_n)$.

By iterating the procedure, the original multiple alignment problem is replaced by the two alignment problems posed by the n prefix sequences $s_1(\leq i_1), \ldots, s_n(\leq i_n)$ and by the n suffix sequences $s_1(> i_1), \ldots, s_n(> i_n)$. At some iteration step the dividing process is stopped and some score-optimal alignment procedure (e.g. standard dynamic programming, or faster versions of it, like `MSA` (cf. [5]) is used to align the remaining (sub)sequences of short length.
There are several alternatives for a stopping criterion, for example a threshold L for the shortest length of the (sub)sequences under consideration, which leads to the following general algorithm:

Divide-and-Conquer alignment algorithm

```
$D\&C - Align\,(\,s_1,\ldots,s_n,L\,)$

If  $\min_{p\in\{1,\ldots,n\}}\{l_p\}\leq L$
    then  return the optimal alignment of $s_1,\ldots,s_n$;
    else  $i_1 := \lceil l_1/2\rceil$ and
          search for indices $i_p\in\{0,..,l_p\}$ $(p=2,\ldots,n)$
          which minimize $C(i_1,\ldots,i_n)$;
          return the Concatenation of $D\&C - Align\,(\,s_1(\leq i_1),\ldots,s_n(\leq i_n),L\,)$
          and                         $D\&C - Align\,(\,s_1(> i_1),\ldots,s_n(> i_n),L\,)$;
```

In the following sections we describe some methods for the remaining problem of efficiently searching for slicing points $i_2,\ldots,i_n$ to cut the sequences.

3 Searching for Slicing Points

The first search for slicing points is, of course, the most time consuming one. In order to calculate a tuple $(i_2,\ldots,i_n)$ which minimizes $C(\hat{i}_1,i_2,\ldots,i_n)$ one has to consider all $\binom{n-1}{2}$ secondary matrices

$$C_{s_p,s_q}=\Big(C_{s_p,s_q}[i_p,i_q]\Big)_{0\leq i_p\leq l_p,0\leq i_q\leq l_q}\qquad(2\leq p<q\leq n)$$

as well as all rows

$$\Big(C_{s_1,s_q}[\hat{i}_1,i_q]\Big)_{0\leq i_q\leq l_q}\qquad(2\leq q\leq n)$$

of the sequences of full length.

Obviously, the calculation of the secondary matrices has a computational time complexity proportional to $2\cdot\sum_{2\leq p<q\leq n}(l_p+1)(l_q+1)$, and searching *exhaustively* all secondary matrices C_{s_p,s_q} $(2\leq p,q\leq n)$ for a tuple $(\hat{i}_1,i_2,...,i_n)$ which minimizes the additional cost C needs $O\left(\prod_{p=2}^{n}(l_p+1)\right)$ time, which is exponential in the number n of sequences.

The following speed-up technique as well as the algorithm presented in subsection 4.1 recently was introduced by Stoye et al. [31]:
Because the estimate of the additional charge is a sum of non-negative numbers $\alpha_{p,q}\cdot C_{s_p,s_q}[i_p,i_q]\geq 0$, it is possible to exclude a tuple of slicing points $(i_1,...,i_n)$, whenever a partial sum of C is larger than the minimum found so far. In particular, for fixed $\hat{i}_1$, any i_q with

$$\alpha_{1,q}\cdot C_{s_1,s_q}[\hat{i}_1,i_q]\geq C$$

can never lead to a smaller sum C.

Therefore, a speed-up of the search is achieved by the following simple idea [31]:

- First an *estimate* $\hat{C}$ of the estimate $C(\hat{i}_1, i_2, \ldots, i_n)$ of the additional charge is calculated.
- The constant $\hat{C}$ leads to lower and upper bounds in the following way:
  - For all $q = 2, \ldots, n$ calculate m_q and u_q such that

$$\alpha_{1,q} \cdot C_{s_1,s_q}[\hat{i}_1, i_q] > \hat{C} \quad \text{for all } m_q < l_q \text{ and } i_q > u_q.$$

This calculation needs time proportional to $\sum_{p=2}^{n} l_p$. The intermediate segment

$$C_{s_1,s_q}[\hat{i}_1, m_q], C_{s_1,s_q}[\hat{i}_1, m_q + 1], \ldots, C_{s_1,s_q}[\hat{i}_1, u_q]$$

forms the *relevant part* of each column $\Big(C_{s_1,s_q}[\hat{i}_1, i_q]\Big)_{0 \le i_q \le l_q}$. So, the search for a better tuple of slicing sites can be restricted to a search space of size proportional to

$$\prod_{p=2}^{n} (u_p - m_p + 1) \qquad \text{rather than} \qquad \prod_{p=2}^{n} (l_p + 1). \tag{1}$$

Therefore, we call $\hat{C}$ the *speed-up constant.*

Note that even the best possible speed-up constant

$$C_{OP} := \min_{(i_2,\ldots,i_n)} \Big\{ C(\hat{i}_1, i_2, \ldots, i_n) \Big\}$$

does not imply

$$u_p = m_p \quad \text{for all } p = 2, \ldots, n.$$

So, even for C_{OP} the search space grows exponentially with n. Therefore, the alignment procedure is impractical for large n with this technique of reducing the search space for cutting points.

4 Speed-Up Constants

The following algorithm efficiently calculates a reasonable speed-up constant $\hat{C}$ (see [31]) for fixed $\hat{i}_1$ $\left(\hat{i}_1 := \lceil \frac{l_1}{2} \rceil\right)$.

FirstRow-zero:

1. For all $2 \leq q \leq n$ search for an $\hat{i}_q$ so that
$$C_{s_1,s_q}[\hat{i}_1, \hat{i}_q] = 0.$$
2. Define
$$\begin{aligned} \hat{C} &:= C(\hat{i}_1, \ldots, \hat{i}_n) \\ &= \sum_{2 \leq p < q \leq n} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{i}_p, \hat{i}_q]. \end{aligned}$$

Beside the calculation of $\hat{C}$, this algorithm has computational time of $O(\sum_p l_p)$.

On some simulated data, *FirstRow-zero* leads to an alignment procedure for up to seven sequences of length up to 500, while for 8 and more sequences (of that length) the speed-up constant is insufficient to make the alignment procedure practical (cf. [31], see also section 5). For the same data set, C_{OP} is insufficient to make the alignment procedure practical for more than 10 sequences. [2] Using speed-up constants close to C_{OP}, the *Divide-and-Conquer* procedure is able to align up to eight or nine sequences (or significantly longer sequences).

In this section we propose natural generalisations of the *FirstRow-zero* algorithm. Some of them lead to *provable* better, that is, smaller speed-up constants. If the calculation of better speed-up constants is fast enough, it leads to faster versions of the alignment algorithm *D&C − Align*.

4.1 Some $O(n^2)$-time methods

In the *FirstRow-zero* procedure each index $\hat{i}_q$ $(2 \leq q \leq n)$ is chosen depending only on $\hat{i}_1$. Because we focus on the sum-of-pairs score, each index i_q in a tuple with minimal

[2]In case of biologically related sequences the procedure often is practical for more than 10 sequences as well as for much longer sequences [31].

additional charges $C(i_1, \ldots, i_n)$ depends on all other indices i_p $(p \neq q)$. Therefore, we consider the sum of all corresponding columns

$$\sum_{p=1}^{q-1} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{i}_p, i_q] \qquad (q = 2, \ldots, n)$$

in order to minimize them:

SC-min:

1. For all $2 \leq q \leq n$ search for an $\hat{i}_q$ so that $C_{s_1,s_q}[\hat{i}_1, \hat{i}_q] = 0$.
2. For all $2 \leq q \leq n$ search for $0 \leq \bar{i}_q \leq l_q$ such that
$$\sum_{p=1}^{q-1} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{i}_p, i_q] \qquad \text{is minimal for } i_q = \bar{i}_q.$$
3. Define $\bar{C} := C(\hat{i}_1, \bar{i}_2, \ldots, \bar{i}_n)$.

Thereby we ensure that each calculated cutting position $\bar{i}_q$ depends on all indices $\hat{i}_p$ $(p = 2, \ldots, q-1)$.

A simpler and more natural variant of the *SC-min* procedure (with $\tilde{i}_1 := \hat{i}_1$ fixed) is

nSC-min:

1. For all $q = 2$ up to n, search for $0 \leq \tilde{i}_q \leq l_q$ such that
$$\sum_{p=1}^{q-1} \alpha_{p,q} \cdot C_{s_p,s_q}[\tilde{i}_p, i_q] \qquad \text{is minimal for } i_q = \tilde{i}_q.$$
2. Define $\tilde{C} := C(\tilde{i}_1, \ldots, \tilde{i}_n)$.

Of course, *nSC-min* is sensitive to the whole input order, while *SC-min* depends only on the choice of s_1. We have observed, that both algorithms mostly lead to smaller speed-up constants than *FirstRow-zero* (see section 5). The running time of both algorithms is proportional to $\binom{n}{2} \max_p\{l_p\}$.

Another variant of the *SC-min* procedure simply is obained by taking into account the sum of all rows $\left(C_{s_q,s_p}[i_q, \hat{i}_p]\right)_{i_q}$ for $p > q$ in the minimization step, so that the second step of *SC-min* is substituted by

2'. For all $2 \le q \le n$ search for $0 \le \grave{i}_q \le l_q$ such that

$$\sum_{p=1}^{q-1} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{i}_p, i_q] + \sum_{p=q+1}^{n} \alpha_{q,p} \cdot C_{s_q,s_p}[i_q, \hat{i}_p] \qquad \text{is minimal for } i_q = \grave{i}_q.$$

We call this variant *SCR-min* (short for *S*um of corresponding *C*olumns and *R*ows). Obviously, *SCR-min* has a computational time complexity proportional to $(n-1)^2 \max_p\{l_p\}$.

Combining the *nSC-min* and *SCR-min* procedures leads to a calculation of the speed-up constant which is always better than that calculated by *FirstRow-zero*. The *SCR-min* algorithm improves the speed-up constant calculable from the tuple $(\hat{i}_1, \dots, \hat{i}_n)$ by one minimization ($q = 2$) to that calculable from $(\hat{i}_1, \bar{i}_2, \hat{i}_3, \dots, \hat{i}_n)$. Using these indices as an input for the same procedure, but now minimizing over $\hat{i}_3$, leads to a smaller speed-up constant

$$C(\hat{i}_1, \bar{i}_2, \bar{i}_3, \hat{i}_4, \dots, \hat{i}_n).$$

MinC:

1. Put $\hat{i}_1 = \bar{i}_1 := \lceil \frac{l_1}{2} \rceil$.

2. For all $2 \le q \le n$ search for an $\hat{i}_q$ so that $C_{s_1,s_q}[\hat{i}_1, \hat{i}_q] = 0$.

3. For $k = 2$ up to n search for $0 \le \bar{i}_k \le l_k$ such that

$$\sum_{p=1}^{k-1} \alpha_{p,k} \cdot C_{s_p,s_k}[\bar{i}_p, i_k] + \sum_{q=k+1}^{n} \alpha_{k,q} \cdot C_{s_k,s_q}[i_k, \hat{i}_q] \qquad \text{is minimal for } i_k = \bar{i}_k.$$

4. Define $\bar{C} := C(\bar{i}_1, \dots, \bar{i}_n)$.

Obviously, **MinC** has the same time complexity as *SCR-min*. But this algorithm improves the speed-up constant calculable from the tuple $(\bar{i}_1, \dots, \bar{i}_{k-1}, \hat{i}_k, \dots, \hat{i}_n)$ by the $k-$th run of step 3 to

$$C(\bar{i}_1, \dots, \bar{i}_k, \hat{i}_{k+1}, \dots, \hat{i}_n).$$

Lemma 2 $\bar{C} \le \hat{C}$.

Proof Let $n \in \mathbf{N}^{>2}$ and $2 \le k \le n$. By definition of $C(i_1, \dots, i_n)$ we have

$$\begin{aligned} & C(\bar{i}_1, \dots, \bar{i}_{k-1}, \hat{i}_k, \dots, \hat{i}_n) \\ &= \sum_{p=1}^{k-1} \sum_{q=k}^{n} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \hat{i}_q] + \sum_{1 \le p < q \le k-1} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \bar{i}_q] \end{aligned}$$

$$
\begin{aligned}
& + \sum_{k \le p < q \le n} \alpha_{pq}\, C_{s_p s_q}[\,\hat{i}_p, \hat{i}_q] \\
= & \sum_{p=1}^{k-1} \sum_{q=k+1}^{n} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \hat{i}_q] + \sum_{p=1}^{k-1} \alpha_{p,k}\, C_{s_p s_k}[\,\bar{i}_p, \hat{i}_k] \\
& + \sum_{1 \le p < q \le k-1} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \bar{i}_q] + \sum_{k+1 \le p < q \le n} \alpha_{pq}\, C_{s_p s_q}[\,\hat{i}_p, \hat{i}_q] \\
& + \sum_{q=k+1}^{n} \alpha_{k,q}\, C_{s_k s_q}[\,\hat{i}_k, \hat{i}_q] \\
\ge & \sum_{p=1}^{k-1} \sum_{q=k+1}^{n} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \hat{i}_q] + \sum_{1 \le p < q \le k-1} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \bar{i}_q] \\
& + \sum_{k+1 \le p < q \le n} \alpha_{pq}\, C_{s_p s_q}[\,\hat{i}_p, \hat{i}_q] \\
& + \min_{0 \le i_k \le l_k} \left\{ \sum_{p=1}^{k-1} \alpha_{p,k}\, C_{s_p s_k}[\,\bar{i}_p, i_k] + \sum_{q=k+1}^{n} \alpha_{k,q}\, C_{s_k s_q}[\,i_k, \hat{i}_q] \right\} \\
= & \sum_{p=1}^{k-1} \sum_{q=k+1}^{n} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \hat{i}_q] + \sum_{1 \le p < q \le k-1} \alpha_{pq}\, C_{s_p s_q}[\,\bar{i}_p, \bar{i}_q] \\
& + \sum_{k+1 \le p < q \le n} \alpha_{pq}\, C_{s_p s_q}[\,\hat{i}_p, \hat{i}_q] \\
& + \sum_{p=1}^{k-1} \alpha_{p,k}\, C_{s_p s_k}[\,\bar{i}_p, \bar{i}_k] + \sum_{q=k+1}^{n} \alpha_{k,q}\, C_{s_k s_q}[\,\bar{i}_k, \hat{i}_q] \\
= & \; C(\bar{i}_1, \ldots, \bar{i}_k, \hat{i}_{k+1}, \ldots, \hat{i}_n)
\end{aligned}
$$

For $k = n$ the above inequality is equivalent to Lemma 2.

In the following subsections we present two extensions of the above procedure. Analogous to Lemma 2, we can show that they improve the speed-up constant further more.

4.2 An iterative method

A simple extension of **MinC** is to apply the third step iteratively:
given a number I of iterations, in each run j $(1 \le j \le I)$ we calculate the minimizing indices $\bar{i}_{k_j}$ $(k = 2, \ldots, n)$ by using the indices $\bar{i}_{k_{j-1}}$, produced in run $j-1$, as for $\hat{i}_k$ $(k = 2, \ldots, n)$ in **MinC**. Clearly, as starting points we put $\bar{i}_{k_0} := \hat{i}_k$ for all $k = 2, \ldots, n$, so that the first run of the iterated procedure is **MinC** itself.

itMinC(I):

1. Put $\hat{i}_1 = \bar{i}_1 := \lceil \frac{l_1}{2} \rceil$.

2. For all $2 \le q \le n$ search for $0 \le \hat{i}_q$ $(=: \bar{i}_{q_0}) \le l_q$ so that $C_{s_1,s_q}[\hat{i}_1, \hat{i}_q] = 0$.

3. For $j = 1$ up to I
   for all $k = 2$ up to n search for $0 \le \bar{i}_{k_j} \le l_k$ such that

$$\sum_{p=1}^{k-1} \alpha_{p,k} \cdot C_{s_p,s_k}[\bar{i}_{p_j}, i_k] + \sum_{q=k+1}^{n} \alpha_{k,q} \cdot C_{s_k,s_q}[i_k, \bar{i}_{p_{j-1}}] \quad \text{is minimal for } i_k = \bar{i}_{k_j}.$$

4. Define $\bar{C}_{it_I} := C(\bar{i}_{1_I}, \bar{i}_{2_I}, \ldots, \bar{i}_{n_I})$.

Lemma 3 $\bar{C}_{it_I} \le \bar{C}$ for all $I \in \mathbf{N}$.

The **proof** is analogous to the proof of Lemma 2 by showing that each iteration j improves the speed-up constant calculable from the tuple $(\bar{i}_{1_j}, \ldots, \bar{i}_{n_j})$ to a value smaller than

$$C(\bar{i}_{1_{j-1}}, \ldots, \bar{i}_{n_{j-1}}).$$

One can easily adopt some stopping criterion to save running time, e.g. an iteration $j+1$ should only be carried out if the previous iteration j has obtained some change $\bar{i}_{q_j} \ne \bar{i}_{q_{j-1}}$.

Beside this, for calculating the tuple $(\bar{i}_{1_I}, \ldots, \bar{i}_{n_I})$ the algorithm needs computational time proportional to $I(n-1)^2 l$ and is very useful to reduce the speed-up constant.

4.3 A polynomial-time method

At this stage we only have tried to minimize $C(i_1, i_2, \ldots, i_n)$ by varying one index at a time and keeping all the others fixed. In principle, one can carry out the minimization of sums of corresponding rows and columns over s indices *simultaneously* ($s \in \{2, \ldots, n\}$), which reduces the probability of running into bad local minima. This leads to algorithms with a computational time complexity polynomial in $\max_p l_p$ (and quadratic in n). So they also can be helpful in speeding up the *Divide-and-Conquer* alignment procedure. For simplicity, we first describe the method in the case $s = 2$:

siMinC(2):

1. Put $\hat{i}_1 = \bar{i}_1 := \lceil \frac{l_1}{2} \rceil$.

2. For all $2 \leq q \leq n$ search for an $\hat{i}_q$ so that $C_{s_1,s_q}[\hat{i}_1, \hat{i}_q] = 0$.
3. For $k = 1$ up to $\lfloor \frac{n-1}{2} \rfloor$ search for $(i_{2k}, i_{2k+1}) \in \{0, \ldots, l_{2k}\} \times \{0, \ldots, l_{2k+1}\}$ such that

$$\sum_{l=1}^{2k-1} \left\{ \alpha_{l,2k} C_{s_l s_{2k}}[\bar{i}_l, i_{2k}] + \alpha_{l,2k+1} C_{s_l s_{2k+1}}[\bar{i}_l, i_{2k+1}] \right\}$$
$$+ \alpha_{2k,2k+1} C_{s_{2k} s_{2k+1}}[i_{2k}, i_{2k+1}]$$
$$+ \sum_{l=2(k+1)}^{2 \cdot \lfloor \frac{n-1}{2} \rfloor} \left\{ \alpha_{2k,l} C_{s_{2k} s_l}[i_{2k}, \hat{i}_l] + \alpha_{2k+1,l} C_{s_{2k+1} s_l}[i_{2k+1}, \hat{i}_l] \right\}$$

   is minimal for $(i_{2k}, i_{2k+1}) = (\bar{i}_{2k}, \bar{i}_{2k+1})$.
4. If n is even, search for $0 \leq \bar{i}_n \leq l_n$ such that

$$\sum_{l=1}^{n} \alpha_{l,n} C_{s_l s_n}[\bar{i}_l, i_n] \quad \text{is minimal for } i_n = \bar{i}_n.$$

5. Define $\bar{C}_{si_2} := C(\bar{i}_1, \ldots, \bar{i}_n)$.

The algorithm needs time $O(n^2 l^3)$.

Lemma 4 $\bar{C}_{si_2} \leq \bar{C}$.

The **proof** is straightforward, analogous to the proof of Lemma 2.

For completeness we present the algorithm for arbitrary s ($s \in \{2, \ldots, n\}$).

siMinC(s):

1. Put $\hat{i}_1 = \bar{i}_1 := \lceil \frac{l_1}{2} \rceil$.
2. For all $2 \leq q \leq n$ search for an $\hat{i}_q$ so that $C_{s_1,s_q}[\hat{i}_1, \hat{i}_q] = 0$.
3. For $k = 1$ up to $\lfloor \frac{n-1}{s} \rfloor$ search for

$$(i_{ks}, i_{ks+1}, \ldots, i_{(k+1)s-1}) \in \bigotimes_{m=ks}^{(k+1)s-1} \{0, \ldots, l_m\}$$

such that

$$\sum_{l=1}^{ks-1} \sum_{m=ks}^{(k+1)s-1} \alpha_{l,m} C_{s_l s_m}[\bar{i}_l, i_m] + \sum_{ks \le m < m' \le (k+1)s-1} \alpha_{m,m'} C_{s_m s_{m'}}[i_m, i_{m'}] + \sum_{l=(k+1)s}^{s \cdot \lfloor \frac{n-1}{s} \rfloor} \sum_{m=ks}^{(k+1)s-1} \alpha_{m,l} C_{s_m s_l}[i_m, \hat{i}_l]$$

is minimal for $(i_{ks}, \ldots, i_{(k+1)s-1}) = (\bar{i}_{ks}, \ldots, \bar{i}_{(k+1)s-1})$.

4. If $r := n - 1 - s \cdot \lfloor \frac{n-1}{s} \rfloor \neq 0$, search for $(i_{n-r+1}, \ldots, i_n) \in \bigotimes_{m=n-r+1}^{n} \{0, \ldots, l_m\}$ such that

$$\sum_{l=1}^{n-r} \sum_{m=n-r+1}^{n} \alpha_{l,m} C_{s_l s_n}[\bar{i}_l, i_m]$$

is minimal for $(i_{n-k+1}, \ldots, i_n) = (\bar{i}_{n-k+1}, \ldots, \bar{i}_n)$.

5. Define $\bar{C}_{si_s} := C(\bar{i}_1, \ldots, \bar{i}_n)$.

Steps 3 and 4 of the algorithm have a computational time complexity proportional to $n(n-s)l^{s+1}$, which makes it feasible only for small s.

5 Results

To illustrate the improvement of our speed-up constants to $\hat{C}$ (calculated by *FirstRow-zero*) with an example, our algorithms have been applied to several sets of sequences, generated by a stochastical mutation process on random protein sequences of length about 100. The sequences have a pairwise identity between 15 and 25 percent. The following table contains the relative length of the relevant parts $\frac{u_p - m_p + 1}{l_p + 1}$, (see eq. (1) of section 3) averaged over $p = 1, \ldots, n$ caused by various methods, as well as by the optimal speed-up constant (calculated by an exponential-time search). All shown results are averages over 100 different sequence sets.

n	3	4	5	6	7	8	9
FirstRow-zero	0.033	0.078	0.132	0.215	0.298	0.423	0.531
nSC-min	0.026	0.059	0.106	0.183	0.260	0.357	0.465
SCR-min	0.029	0.066	0.111	0.179	0.258	0.366	0.463
MinC	0.025	0.057	0.098	0.163	0.239	0.334	0.424
itMinC(5)	0.025	0.055	0.095	0.156	0.231	0.319	0.409
$C_{OP}(s_k)$	0.024	0.052	0.091	0.149	0.226	0.312	0.402

Table 1: The proportion of one sequence which has to be searched for slicing points using the different speed-up constants, are shown, and compared with the proportion resulting by using the optimal speed-up constant.

6 Discussion

If an multiple sequence alignment procedure must be called up very often for a certain purpose (e.g. in [27],[37]), the *Divide-and-Conquer* procedure including the speed-up techniques proposed in this paper is particularly useful.

For simplicity, we have introduced these algorithms by applying them to the sequences of full length. Precalculating lower and upper bounds by using the *FirstRow-zero* procedure (see subsection 3.1, 3.2), the algorithms can be applied only to the relevant parts (see eq. (1)) which obviously leads to a further speed up of the resulting alignment procedure.

The algorithms presented in section 4 depend on the input order of the sequences under consideration. It should be useful to order them appropriately, e.g. by considering the pairwise distance score used to define the weight factors $\alpha_{p,q}$.

The running times of the above outlined algorithms differ significantly. Therefore, an integration to the *Divide-and-Conquer* alignment procedure should be done such that for *each search* for cutting points *the appropriate algorithm* is used, depending on the properties of the set of (sub)sequences under consideration, like the number of sequences, their length, and their pairwise distance scores. Especially, for more distantly related sequences (usually showing high pairwise distance scores) it should be better to use **siMinC** than **itMinC**.

Acknowledgements

The authors wish to thank Andreas W.M. Dress for introducing us to the whole subject, and Mike D. Hendy for helpful comments.

References

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

[2] S.F. Altschul, D.J. Lipman, Trees, Stars and Multiple Biological Sequence Alignment, *SIAM J. Appl. Math.* 49, pp.197-209, 1989

[3] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic Local Alignment Search Tool, *J. Mol. Biol.* 215, pp.403-410, 1990

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

[5] H. Carillo, D. Lipman, The Multiple Sequence Alignment Problem in Biology, *SIAM J. Appl. Math.* 48 no.5, pp.1073-1082, 1988

[6] S.C. Chan, A.K.C. Wong, D.K.Y. Chiu, A Survey of Multiple Sequence Comparison Methods, *Bull. Math. Biol.* 54 no.4, pp.563-598, 1992

[7] R.F. Doolittle, Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences, *Methods of Enzymology* Vol. 183, 1990

[8] A.W.M. Dress, G. Füllen, S.W. Perrey, A Divide and Conquer Approach to Multiple Alignment, *Proceedings of the 3rd International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, Menlo Park, California, pp.107-113, June 1995

[9] Da-Fei Feng, R.F. Doolittle, Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees, *J. Mol. Evol.* 25, pp.351-360, 1987

[10] O. Gotoh, Alignment of Three Biological Sequences with an Efficient Traceback Procedure, *J. Theo. Biol.* 121, pp.327-337, 1986

[11] M. Gribskov, A.D. McLachlan, D. Eisenberg, Profile Analysis: Detection of Distantly Related Proteins, *Proc. Natl. Acad. Sci. USA* 84, pp.4355-4358, 1987

[12] 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

[13] J.J. Hein, Unified Approach to Alignment and Phylogenies, in *Methods in Enzymology* Vol. 183, (ed. R.F. Doolittle), pp.626-645, 1990

[14] M. Hirosawa, M. Hoshida, M. Ishikawa, T. Toya, MASCOT: Multiple Alignment System for Protein Sequences Based on Three-Way Dynamic Programming, *CABIOS* 9, no.2, p.161-167, 1993

[15] D.S. Hirschberg, A Linear Space Algorithm for Computing Maximal Common Subsequences, *Comm. ACM* 18, pp.341-343, 1975

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

[17] M.S. Johnson, R.F. Doolittle, A Method for Simultaneous Alignment of Three or More Amino Acid Sequences, *J. Mol. Evol.* 23, pp.267-278, 1986

[18] J.D. Kececioglu, The Maximum Weight Trace Problem in Multiple Sequence Alignment, in *Proc. of the 4th Symp. on Comb. Pattern Matching, LNCS* 684, pp.106-119, 1993

[19] D.J. Lipman, S.F. Altschul, J.D. Kececioglu, A Tool for Multiple Sequence Alignment, *Proc. Natl. Acad. Sci. USA* 86, pp.4412-4415, 1989

[20] M. Murata, J.S. Richardson, J.L. Sussman, Simultaneous Comparison of Three Protein Sequences, *Proc. Natl. Acad. Sci. USA* 82, pp.3073-3077, 1985

[21] E.W. Myers, W. Miller, Optimal Alignments in Linear Space, *CABIOS* 4, no.1, pp.11-17, 1988

[22] E.W. Myers, An Overview of Sequence Comparison Algorithms in Molecular Biology, Tech.Rep. 91-29, Dept. of Comp. Sci., The Univ. of Arizona, Tucson, Arizona 85721, 1991

[23] D. Naor, D.L. Brutlag, On Near-Optimal Alignments of Biological Sequences, *J. Comp. Biol.* 1, no.4, pp.349-366, 1994

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

[25] W.R. Pearson, D.J. Lipman, Improved Tools for Biological Sequence Comparison, *Proc. Natl. Acad. Sci. USA* 85, pp.2444-2448, 1988

[26] P.H. Sellers, On the Theory ans Computation os Evolutionary Distances, *SIAM J. Appl. Math.* 26, no.4, pp.787-793, 1974

[27] D. Sankoff, R.J. Cedergren, G. LaPalme, Frequency of insertion-deletion, transversion, and transition in the evolution of 5s ribosomal RNA, *Journal of Molecular Evolution* 7, pp.133-149, 1976

[28] D. Sankoff, J.B. Kruskal (eds.), Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison, *Addison-Wesley*, London, 1983

[29] T.F. Smith, M.S. Waterman, Comparison of Biosequences, *Adv. Appl. Math.* 2, pp.482-489, 1981

[30] T.F. Smith, M.S. Waterman, Identification of Common Molecular subsequences, *J. Mol. Biol.* 147, pp.195-197, 1981

[31] J. Stoye, S.W. Perrey, U. Tönges, A.W.M. Dress, Improving the Divide-and-Conquer Approach to Sum-of-Pairs Multiple Sequence Alignment, manuscript

[32] W.R. Taylor, Motif-biased Protein Sequence Alignment, *J. Comp. Biol.* 1, no.4, pp.297-310, 1994

[33] W.R. Taylor, K. Hatrick, Compensating Changes in Protein Multiple Sequence Alignments, *Prot. .Eng.* 7, no.3, pp.341-348, 1994

[34] J.D. Thompson, D.G. Higgins, 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, no.22, pp.4673-4680, 1994

[35] U. Tönges, S.W. Perrey, J. Stoye, A.W.M. Dress, A General Method for Fast Multiple Sequence Alignment, Gene-COMBIS, accepted

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

[37] M. Vingron, A.v. Haeseler, Towards Integration of Multiple Alignment and Phylogenetic Tree Reconstruction, *Arbeitspapiere der GMD* 852, June 1994

[38] L. Wang, T. Jiang, On the Complexity of Multiple Sequence Alignment, *J. Comp. Biol.* 1, no.4, pp.337-348, 1994

[39] M.S. Waterman, T.F. Smith, W.A. Beyer, Some Biological Sequence Metrics, *Adv. Math.* 20, pp.367-387, 1976