# Improving the Divide-and-Conquer Approach to Sum-of-Pairs Multiple Sequence Alignment

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

S. W. PERREY
Department of Mathematics, Massey University
Palmerston North, New Zealand
S.W.Perrey@massey.ac.nz

A. W. M. DRESS
Research Center for Interdisciplinary Studies on Structure Formation (FSPM)
University of Bielefeld
Postfach 10 01 31, D-33501 Bielefeld, Germany
dress@mathematik.uni-bielefeld.de

*(Received May 1996; accepted August 1996)*

**Abstract**—We consider the problem of multiple sequence alignment: given $k$ sequences of length at most $n$ and a certain scoring function, find an alignment that minimizes the corresponding "sum of pairs" distance score.

We generalize the divide-and-conquer technique described in [1,2], and present new ideas on how to use efficient search strategies for saving computer memory and accelerating the procedure for three or more sequences. Resulting running times and memory usage are shown for several test cases.

**Keywords**—Multiple sequence alignment, Dynamic programming, Divide-and-conquer.

## 1. INTRODUCTION

Multiple sequence alignment is an important problem in computational molecular biology, and many algorithms have been presented in this area of research (for a recent comparison see [3]). Since the problem of computing optimal alignments with respect to the "sum-of-pairs" criterion and most of its variants are NP-hard [4], many approximative algorithms have been proposed (e.g., [5–8]), Unfortunately, almost all of these methods either exhibit a prohibitive computational complexity or yield biologically unplausible results. With our algorithm, we try to contribute towards improving this situation.

## 2. THE PROBLEM

Let us consider a finite alphabet $\mathcal{A}$, $k$ sequences $s_1, s_2, \dots, s_k$ over $\mathcal{A}$ of length $n_1, n_2, \dots, n_k$, respectively, and an additional letter, say '$-$', not contained in $\mathcal{A}$, which symbolizes gaps. An alignment of $s_1, s_2, \dots, s_k$ is given by a $k \times N$ matrix $M = (m_{ij})_{1 \le i \le k, 1 \le j \le N}$ for some $N \le \sum_{i=1}^{k} n_i$, with entries $m_{ij} \in \mathcal{A} \cup \{-\}$ subject to the following constraints: it does not contain any column consisting of gaps only, and for each $i = 1, 2, \dots, k$, the row $(m_{i1}, m_{i2}, \dots, m_{iN})$ reproduces the sequence $s_i$ upon eliminating all of its gap letters.

The *weighted sum of pairs* multiple sequence alignment problem can now be described as follows (cf. [9]): given $s_1, s_2, \dots, s_k$, and given a scoring function $D : (\mathcal{A} \cup \{-\})^2 \to \mathbb{R}$, defined on all possible pairs of letters, find an *optimal* alignment $M$, i.e., an alignment that minimizes

$$w(M) := \sum_{1 \le p < q \le k} \left( \alpha_{p,q} \cdot \sum_{j=1}^{N} D(m_{pj}, m_{qj}) \right),$$

where the $\alpha_{p,q}$ are sequence dependent (nonnegative) weight factors reflecting, e.g., phylogenetic relationship. The resulting score is also denoted by $w_{\text{op}}(s_1, \dots, s_k)$.

It is well known that the multiple sequence alignment problem can be solved optimally by *dynamic programming* [10,11] with running time proportional to $2^k \cdot \prod_{i=1}^{k} n_i$, searching for a shortest *alignment path* in a directed graph with $\prod_{i=1}^{k} n_i$ vertices. Faster variants and a number of speedups of dynamic programming have, therefore, been proposed (e.g., [9,12]). Unfortunately, most of these approaches still need exceedingly large computing time and computer memory when applied to more than, say, six protein sequences of average length 300.

## 3. THE BASIC ALGORITHM

Our algorithm, previously described in [1,2] for the restricted case of three sequences, works according to the well-known *divide-and-conquer* principle: each of the given $k$ sequences is being cut at an appropriately chosen site somewhere near to its midpoint, this way reducing the original alignment problem to the two subproblems of aligning the two resulting groups of prefix and suffix subsequences, respectively. These will be handled by the same procedure in a recursive manner. The recursion stops when the remaining subsequences are short enough (e.g., shorter than a pregiven threshold $L$) to be aligned by a standard procedure—in our implementation, we use MSA [13,14].

The main problem is to find a tuple of *ideal* slicing sites $(c_1, c_2, \dots, c_k)$ so that the simple concatenation of the two optimal alignments of the prefixes[1] $s_1(\le c_1)$, $s_2(\le c_2), \dots, s_k(\le c_k)$ and the suffixes $s_1(> c_1)$, $s_2(> c_2), \dots,$ $s_k(> c_k)$ forms an optimal alignment of the original sequences.

Obviously, for any fixed site $\hat{c}_1$ $(1 \le \hat{c}_1 \le n_1)$, there exists a $(k-1)$-tuple $(c_2(\hat{c}_1), \dots, c_k(\hat{c}_1))$, such that $(\hat{c}_1, c_2(\hat{c}_1), \dots, c_k(\hat{c}_1))$ forms a $k$-tuple of ideal slicing sites. Unfortunately, finding these points requires searching the whole $k$-dimensional hypercube, requiring as much time as the standard dynamic programming procedure. So, of course, this is not the method of choice.

Instead, our algorithm tries to find so-called *C-optimal* slicing sites that are based on pairwise sequence comparisons, only. More precisely, we use the dynamic programming procedure which we apply to all pairs of sequences $(s_p, s_q)$. The resulting score matrices for pairwise alignment give rise to *additional cost matrices*

$$C_{s_p,s_q}[c_p, c_q] := w_{\text{op}}(s_p(\le c_p), s_q(\le c_q)) + w_{\text{op}}(s_p(> c_p), s_q(> c_q)) - w_{\text{op}}(s_p, s_q),$$

[1]Here, $s_p\,(\le c_p)$ denotes the prefix subsequence of $s_p$ with indices running from 1 to $c_p$, and $s_p\,(> c_p)$ denotes the suffix subsequence of $s_p$ with indices running from $c_p + 1$ to $n_p$, $1 \le p \le k$.

which contain the additional charge imposed by forcing the alignment path to run through a particular vertex $(c_p, c_q)$ $(1 \leq p < q \leq k)$. The calculation of $C_{s_p,s_q}$ can be performed by computing *forward* and *reverse* matrices in a similar way as it is described in [15,16], respectively.

Note that there exists, for every fixed $\hat{c}_p$, at least one slicing site $c_q(\hat{c}_p)$ with $C_{s_p,s_q}[\hat{c}_p, c_q(\hat{c}_p)] = 0$. This follows from the facts that the vertices on an optimal pairwise alignment path are precisely those with no additional cost, and that every alignment path meets at least once every position of the two sequences.

To search for a good $k$-tuple of slicing sites, we try to estimate the *multiple additional cost* imposed by forcing the multiple alignment path of the sequences through the particular vertex $(c_1, c_2, \ldots, c_k)$ in the whole ($k$-dimensional) hypercube associated with the corresponding alignment problem. To this end, we use a weighted sum of additional costs over all projections $(c_p, c_q)$ as such an estimate: we put

$$C(c_1, c_2, \ldots, c_k) := \sum_{1 \leq p < q \leq k} \alpha_{p,q} \cdot C_{s_p,s_q}[c_p, c_q],$$

where the $\alpha_{p,q}$ are the same sequence dependent weight factors as above.

Our proposition is now that *C-optimal* slicing sites, i.e., $(c_1, c_2, \ldots, c_k)$ that minimize $C(c_1, c_2, \ldots, c_k)$, result in very good, if not optimal multiple alignments. In conclusion, this leads to the following general procedure.

**Algorithm** *d&c-align* $(s_1, s_2, \ldots, s_k, L)$

```
    If $\max\{n_1, n_2, \ldots, n_k\} \leq L$,
        then return the optimal alignment of $s_1, s_2, \ldots, s_k$ (using, e.g., MSA);
        else return the concatenation of
                        $d\&c\text{-}align\,(s_1(\leq c_1), s_2(\leq c_2), \ldots, s_k(\leq c_k), L)$,
                    and $d\&c\text{-}align\,(s_1(> c_1), s_2(> c_2), \ldots, s_k(> c_k), L)$,
            where $(c_1, c_2, \ldots, c_k) := calc\text{-}cut(s_1, s_2, \ldots, s_k)$.
```

In the following section, we describe how to realize *calc-cut*, which computes a $k$-tuple of $C$-optimal slicing sites.

## 4. EFFICIENTLY CALCULATING THE SLICING SITES

In a naive implementation, the search *calc-cut* for $C$-optimal slicing sites $(c_1, c_2, \ldots, c_k)$ needs time $\mathcal{O}(k^2n^2 + n^{k-1})$, where $n := \max\{n_1, n_2, \ldots, n_k\}$: the computation of all pairwise additional cost matrices takes $\mathcal{O}(k^2n^2)$ time and, for given $\hat{c}_1$, all possible combinations of $c_2, \ldots, c_k$ have to be checked to find the tuple that minimizes $C$ in $\mathcal{O}(n^{k-1})$.

We reduce this running time and the required memory ($\mathcal{O}(k^2n^2)$ for the naive version) by the following approach: we precalculate an estimation $\widehat{C}$ for $C(c_1, c_2, \ldots, c_k)$, which allows us to prune the search space enormously. Because the multiple additional cost $C(c_1, c_2, \ldots, c_k)$ is a sum of nonnegative numbers $\alpha_{p,q} \cdot C_{s_p,s_q}[c_p, c_q]$, it is possible to exclude a tuple of slicing sites $(c_1, c_2, \ldots, c_k)$, whenever one of the summands $\alpha_{p,q} \cdot C_{s_p,s_q}[c_p, c_q]$ is larger than the minimum $\widehat{C}$ found so far. In particular, for fixed $\hat{c}_1$, any $c_p$ with $\alpha_{1,q} \cdot C_{s_1,s_p}[\hat{c}_1, c_p] \geq \widehat{C}$ can never lead to a smaller sum $C$.

With this in mind, a tuple of $C$-optimal slicing sites can be calculated as follows.

**Function** *calc-cut* $(s_1, s_2, \ldots, s_k)$

1. Fix $\hat{c}_1 := \lceil (n_1/2) \rceil$.
2. Calculate and save columns $\mathrm{col}_{1,q}^{\hat{c}_1}[j] := C_{s_1,s_q}[\hat{c}_1, j]$ $(2 \leq q \leq k, 1 \leq j \leq n_q)$.
3. Locate slicing sites $\hat{c}_2, \ldots, \hat{c}_k$ such that $\mathrm{col}_{1,q}^{\hat{c}_1}[\hat{c}_q] = 0$ $(2 \leq q \leq k)$.

4. Calculate the estimate

$$\widehat{C} := \sum_{1\le p<q\le k} \alpha_{p,q} \cdot C_{s_p,s_q}\left[\hat{c}_p, \hat{c}_q\right] = \sum_{2\le p<q\le k} \alpha_{p,q} \cdot C_{s_p,s_q}\left[\hat{c}_p, \hat{c}_q\right],$$

   where to calculate the entries $C_{s_p,s_q}[\hat{c}_p, \hat{c}_q]$ $(2 \le p < q \le k)$ only the memory for one column $\mathrm{col}\,^{\hat{c}_p}_{p,q}$ of order $\mathcal{O}(n)$ is needed at a time, since no part of the matrices $C_{s_p,s_q}$ has to be saved.
5. Calculate lower and upper bounds $l_q$ and $u_q$ such that $\alpha_{1,q} \cdot \mathrm{col}\,^{\hat{c}_1}_{1,q}[j] \ge \widehat{C}$, for all $j < l_q$, and for all $j > u_q$ $(2 \le q \le k)$. The intermediate segment $\mathrm{col}\,^{\hat{c}_1}_{1,q}[l_q], \ldots, \mathrm{col}\,^{\hat{c}_1}_{1,q}[u_q]$ forms the *relevant part* of each column $\mathrm{col}\,^{\hat{c}_1}_{1,q}$.
6. Given these bounds, compute and save the *relevant parts* of the matrices $C_{s_p,s_q}$, defined by $C_{s_p,s_q}[i,j]$ with $l_p \le i \le u_p$ and $l_q \le j \le u_q$.
7. Search for better slicing sites $(\hat{c}_1, c_2(\hat{c}_1), \ldots, c_k(\hat{c}_1))$ within the relevant parts of the columns $\mathrm{col}\,^{\hat{c}_1}_{1,q}$ and of the matrices $C_{s_p,s_q}$. Thereby, the sum $C$ can be computed step by step and the search can be stopped, if an intermediate result is larger than $\widehat{C}$. During this search, with decreasing values of $\widehat{C}$, the relevant part of the columns $\mathrm{col}\,^{\hat{c}_1}_{1,q}$ can possibly be further reduced, decreasing the search space even more.

Obviously, the worst case time and space complexity of this approach still remain $\mathcal{O}(k^2n^2 + n^{k-1})$ and $\mathcal{O}(k^2n^2)$, respectively, for the (very improbable) case where the bounds $l_p$ and $u_p$ can never be increased or decreased, respectively. But for biological sequence families, the effect is enormous: for calculating the first tuple of slicing sites in the recursion, which takes far the longest time of all cut-site computations, the reduction from $n$ to the length $r := \max_{p=2,\ldots,k}\{u_p - l_p + 1\}$ of the longest of the remaining relevant parts of the columns, is usually greater than $100:1$ for small $k$, yielding memory savings for the matrices of at least four orders of magnitude, and reducing the expected time and space complexity to $\mathcal{O}(k^2n^2 + r^{k-1})$ and $\mathcal{O}(kn + k^2r^2)$, respectively. More detailed results concerning exact running times and memory usage are presented in Section 6.

## 5. FURTHER IMPROVEMENTS

In the actual implementation of *d&c-align*, the program MSA is called for every single group of subsequences containing at least one subsequence of length at most $L$. It would be more elegant to calculate all the slicing sites first (which is possible without the knowledge of any of the subalignments) and then call MSA only once with the extra information concerning the slicing sites. Although MSA is able to handle fixed parts of the alignment of length one or longer, it still needs to be extended so as to accept fixed slicing sites, which from this point of view, can be seen as fixing an alignment of zero length. Upon our request, the authors of MSA are working on such an extension [17].

Another advantage expected from this approach, is that, with such a feature, MSA's ability of dealing with biologically more reasonable *quasinatural* gap costs [18] and with a score function, which does not penalize gaps at the beginning and the end of the alignment, will be carried over automatically to our procedure.

To improve the quality of the alignments further, we propose a *windowing approach* to correct the obtained alignments in the proximity of division sites. We suggest to choose a window width $W$ depending on the threshold $L$. Laying such a window across each slicing site, one may search for an optimal realignment of the corresponding subsequences, again using for instance MSA.

## 6. RESULTS

To determine exact time and memory usage, our algorithm has been applied to several sets of sequences obtained by a stochastical mutation process on random sequences of different lengths $n$ and alphabet size 20 denoting the set of amino acids. The sequences have a pairwise identity between 15 and 25 percent. All results shown are averages over ten different sequence sets. As scoring function $D$, we used the PAM-250 distance matrix [19] with positive entries between 0 and 25, and gap penalty 15.

From earlier measurements [2], we obtained a value of $L := 40$ for the stopping criterion to guarantee near-to-optimal results: the score of the alignments calculated in these tests differs by at most 1 percent from the score of the optimal alignment (in cases where we were able to check this with MSA). The values of the weight factors $\alpha_{p,q}$ are calculated from the pairwise optimal alignment scores $w_{\text{op}}(s_p, s_q)$, according to the formula given in [2].

Figures 1 and 2 show plots of running time and memory usage, respectively, for sets of $k = 3, \ldots, 9$, sequences with different average length $n$ on a Silicon Graphics workstation with a MIPS R4000 CPU.

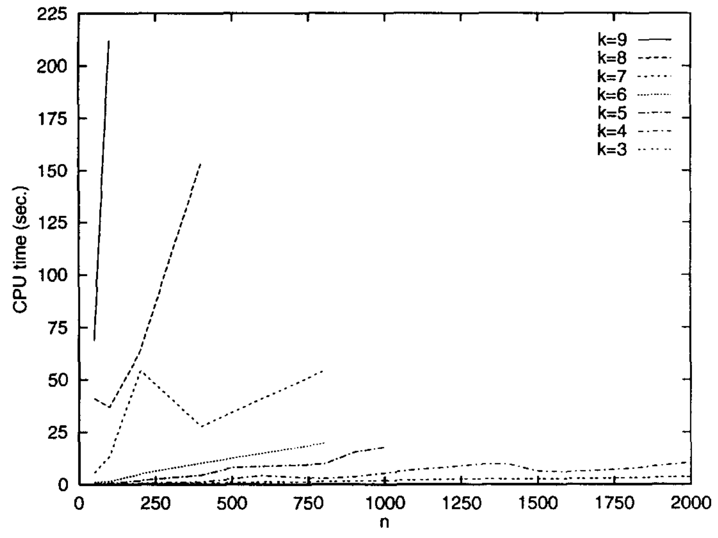

Figure 1. Running times of *d & c-align*.

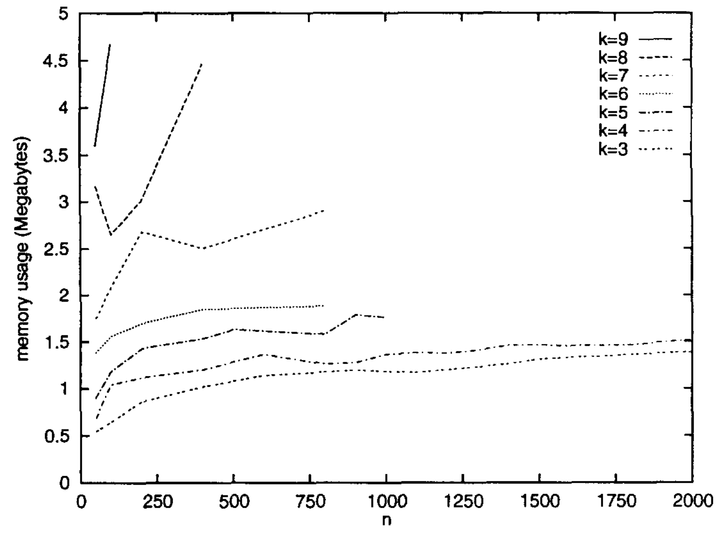

Figure 2. Memory usage of *d & c-align*.

As can be seen from these results, very fast calculation of high-quality alignments with rather moderate memory usage is possible with *d & c-align* for up to seven sequences. In particular, the near-to-linear growth of running time for sets with three and four sequences allows us to align three sequences of length up to 9000, and four sequences of length up to 6000 within 10 minutes, as further measurements have shown. In contrast, the comparatively long running time for eight and more sequences depends (in addition to longer MSA-runs) on the smaller reduction ($r/n$) in these cases (see Table 1), since the multiple additional cost $C$ is a sum of $\binom{k}{2}$ summands, of which only one is tested against $\widehat{C}$ when determining the bounds of the relevant regions.

Table 1. Ratio of relevant part to original size of columns $\mathrm{col}_{1,q}^{\hat{c}_1}$, averaged over 100 sets of $k = 3, \ldots, 9$, sequences.

| $k$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| average $\frac{r}{n}$ | 0.009 | 0.023 | 0.073 | 0.170 | 0.242 | 0.389 | 0.608 |

Finally, we have investigated whether the alignment scores improve upon using the windowing approach mentioned in Section 5. Table 2 shows the relative error of our score for four sets of random sequences with different $k$ and $n$ (i.e., the percentage of the optimal score by which the score obtained by our algorithm exceeds the optimal one), for different values of $L$ and $W$.

Table 2. Effect of windowing on alignments obtained with *d & c-align*.

| $k$ | $n$ | $L$ | $W = 0$ | $W = \frac{1}{4}L$ | $W = \frac{1}{2}L$ | $W = L$ |
|---|---|---|---|---|---|---|
| 3 | 100 | 60 | 0.00% | 0.00% | 0.00% | 0.00% |
| | | 40 | 0.12% | 0.00% | 0.00% | 0.00% |
| | | 20 | 0.47% | 0.35% | 0.25% | 0.25% |
| 3 | 300 | 60 | 0.21% | 0.18% | 0.18% | 0.08% |
| | | 40 | 0.21% | 0.18% | 0.18% | 0.17% |
| | | 20 | 0.36% | 0.35% | 0.32% | 0.17% |
| 4 | 100 | 60 | 0.28% | 0.12% | 0.12% | 0.00% |
| | | 40 | 0.28% | 0.18% | 0.17% | 0.09% |
| | | 20 | 0.34% | 0.18% | 0.18% | 0.17% |
| 5 | 100 | 60 | 0.41% | 0.41% | 0.41% | 0.41% |
| | | 40 | 0.46% | 0.42% | 0.42% | 0.42% |
| | | 20 | 0.69% | 0.69% | 0.50% | 0.50% |

## REFERENCES

1. 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*, Menlo Park, CA, pp. 107–113, AAAI Press, (1995).
2. U. Tönges, S.W. Perrey, J. Stoye and A.W.M. Dress, A general method for fast multiple sequence alignment, *Gene* **172** (GC33–GC41), (1996).
3. M.A. McClure, T.K. Vasi and W.M. Fitch, Comparative analysis of multiple protein-sequence alignment methods, *Mol. Biol. Evol.* **11** (4), 571–592, (1994).
4. L. Wang and T. Jiang, On the complexity of multiple sequence alignment, *J. Comp. Biol.* **1** (4), 337–348, (1994).
5. D. Gusfield, Efficient methods for multiple sequence alignment with guaranteed error bounds, *Bull. Math. Biol.* **55** (1), 141–154, (1993).
6. J. Kececioglu, The maximum weight trace problem in multiple sequence alignment, In *Proc. of the 4th Symp. on Combinatorial Pattern Matching, LNCS 684*, pp. 106–119, (1993).
7. 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).
8. V. Bafna, E.L. Lawler and P.A. Pevzner, Approximation algorithms for multiple sequence alignment, In *Proc. of the 5th Symp. on Combinatorial Pattern Matching, LNCS 807*, pp. 43–53, (1994).
9. H. Carillo and D. Lipman, The multiple sequence alignment problem in biology, *SIAM J. Appl. Math.* **48** (5), 1073–1082, (1988).
10. 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).

11. M.S. Waterman, Introduction to computational biology, In *Maps Sequences and Genomes*, Chapman & Hall, London, UK, (1995).
12. M. Murata, J.S. Richardson and J.L. Sussman, Simultaneous comparison of three protein sequences, *Proc. Natl. Acad. Sci. USA* **82**, 3073-3077, (1985).
13. 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).
14. 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).
15. E.W. Myers and W. Miller, Optimal alignments in linear space, *CABIOS* **4** (1), 11–17, (1988).
16. D. Naor and D.L. Brutlag, On near-optimal alignments of biological sequences, *J. Comp. Biol.* **1** (4), 349–366, (1994).
17. A.A. Schäffer, Personal communication.
18. S.F. Altschul, Gap costs for multiple sequence alignment, *J. Theor. Biol.* **138**, 297–309, (1989).
19. M.O. Dayhoff, R.M. Schwartz and B.C. Orcutt, A model of evolutionary change in proteins, In *Atlas of Protein Sequences and Structure*, (Edited by M.O. Dayhoff), Volume 5, Suppl. 3, pp. 345–352, National Biomedical Research Foundation, Washington, DC, (1978).