PERGAMON

Applied Mathematics Letters 15 (2002) 11–16

Applied Mathematics Letters

www.elsevier.com/locate/aml

A Simple and Space-Efficient Fragment-Chaining Algorithm for Alignment of DNA and Protein Sequences

B. MORGENSTERN
GSF–National Research Center for Environment and Health
Institute of Biomathematics and Biometry
Ingolstädter Landstr. 1, 85764 Neuherberg, Germany
and
Aventis Pharma Limited
Rainham Road South, Essex RM10 7XS, U.K.

(Received February 2001; accepted March 2001)

Communicated by A. Dress

Abstract—In the segment-based approach to sequence alignment, nucleic acid, and protein sequence alignments are constructed from *fragments*, i.e., from *pairs of ungapped segments* of the input sequences. Given a set F of *candidate* fragments and a weighting function $w : F \to \mathbf{R}_0^+$, the score of an alignment is defined as the sum of weights of the fragments it consists of, and the optimization problem is to find a *consistent collection of pairwise disjoint fragments with maximum sum of weights*. Herein, a sparse dynamic programming algorithm is described that solves the pairwise segment-alignment problem in $O(L + N_{\max})$ space where L is the maximum length of the input sequences while $N_{\max} \leq \#F$ holds. With a recently introduced weighting function w, small sets F of candidate fragments are sufficient to obtain alignments of high quality. As a result, the proposed algorithm runs in essentially linear space.

Keywords—Sequence alignment, String algorithm, Fragment chaining, Dynamic programming, Complexity.

1. INTRODUCTION

Sequence alignment is a fundamental problem in molecular bioinformatics. The goal is to develop computer programs that produce *biologically* meaningful alignments. This requires

(a) an appropriate objective function P assigning a quality score to every possible alignment of a given sequence set, and
(b) an efficient algorithm capable of finding P-optimal or near-optimal alignments.

Most standard alignment programs are based on the NW-objective function that had been proposed in 1970 by Needleman and Wunsch [1]. They defined the score of a pairwise alignment as

Present address: MIPS, Max-Planck-Institut für Biochemie, Am Klopferspitz 18a, 82152 Martinsried, Germany.
I would like to thank A. Dress and S. Kurtz for helpful comments on the manuscripts.

0893-9659/01/$ - see front matter Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX
PII: S0893-9659(01)00085-4

the sum of individual similarity scores of aligned residue pairs from which a penalty is subtracted for every gap introduced into the sequences.

Unlike these methods, the program DIALIGN constructs pairwise and multiple alignments of nucleic acid and protein sequences from ungapped pairs of sequence segments [2]. Such segment pairs are referred to as *(alignment) fragments*, and a *pairwise* alignment can be defined as a *chain* of fragments $f_1 \ll \cdots \ll f_k$ where $f_i \ll f_j$ means that, in both sequences, the end positions of f_i are strictly smaller than the respective beginning positions of f_j. Note that fragments may contain *mismatches* and may have varying length. In order to find 'good' alignments, every possible fragment f is given a nonnegative score $w(f)$ reflecting the degree of similarity among the two segments, and the overall score of an alignment is defined as the sum of fragment scores. Thus, for pairwise alignment, our optimization problem is to find a *chain* of fragments with maximal overall score. In the DIALIGN approach, such optimal pairwise alignments are also the first step for a greedy multiple-alignment procedure [3,4]; they are also used in alternative multiple-alignment procedures [5,6].

There are well-known algorithms that solve this problem. If a set F of admissible fragments is *known*, the problem can be solved in $O(\#F)$ space [7], so if the set F is generated in a first step, the whole problem can be solved in $O(L+\#F)$ space where L is the length of the longer sequence. Chao and Miller [8] proposed an efficient algorithm for the case where *maximal* pairs of *identical* segments are considered and an affine penalty is charged for connecting two fragments. By combining a *sparse* dynamic programming algorithm by Eppstein *et al.* [9] with Hirschberg's [10] and Myers and Miller's [11] divide-and-conquer approach, they obtained an algorithm that runs in space proportional to the length of the input sequences. Recently, we outlined a much simpler fragment-chaining algorithm for the case where *general* gap-free segment pairs are allowed and gaps between fragments are *not* penalized [12]. Herein, this algorithm is described in detail. Its *worst-case* space complexity is discussed, and some techniques are proposed to improve its *real* space efficiency.

2. PRELIMINARIES

It is well known that for two sequences $\mathbf{a} = a_1 \dots a_{L_1}$ and $\mathbf{b} = b_1 \dots b_{L_2}$, an NW-optimal alignment can be calculated in two steps: first, for all positions (i, j) in the comparison matrix $(i, j)_{1 \le i \le L_1, 1 \le j \le L_2}$, the score $\mathrm{Sc}[i, j]$ of an optimal alignment of the *prefixes* $a_1 \dots a_i$ and $b_1 \dots b_j$ is calculated *recursively* from the values $\mathrm{Sc}[i, j-k]$, $\mathrm{Sc}[i-k, j]$, $k \ge 1$, and $\mathrm{Sc}[i-1, j-1]$ using equation

$$\mathrm{Sc}[i,j] = \max \left\{ \begin{array}{c} \max_{k \ge 1}\{\mathrm{Sc}[i, j-k] - g(k)\}, \\ \max_{k \ge 1}\{\mathrm{Sc}[i-k, j] - g(k)\}, \\ \mathrm{Sc}[i-1, j-1] + s(a_i, b_j)\}. \end{array} \right\} \tag{1}$$

Here, $g(k)$ is the penalty for a gap of length k, and $s(a_i, b_j)$ is the score for aligning a_i and b_j. During the recursive procedure, one has to store at every position (i, j) where the optimal alignment of the *prefixes* $a_1 \dots a_i$ and $b_1 \dots b_j$ 'comes from', i.e., if a_i is aligned to b_j or if a_i or b_j are aligned to a gap of a certain length. This allows, in a second step, to find an optimal Needleman-Wunsch alignment of the sequences $\mathbf{a}$ and $\mathbf{b}$ by a *back-tracing* procedure.

It is important to bear in mind that both the recurrence formula and the back-tracing procedure rely on the *additivity* of the NW-function: if it is known that a_i is aligned to b_j in an optimal alignment of the prefixes $a_1 \dots a_i$ and $b_1 \dots b_j$, then the score of this alignment is the corresponding score for the prefixes $a_1 \dots a_{i-1}$ and $b_1 \dots b_{j-1}$ plus the similarity value $s(a_i, b_j)$; the same holds for residues that are aligned to gap characters. This fact is used in the first (recursive) part of the algorithm. Moreover, if one knows that a_i is aligned to b_j in an optimal alignment of the sequences $\mathbf{a}$ and $\mathbf{b}$, then it is possible to calculate how the prefixes $a_1 \dots a_i$ and $b_1 \dots b_j$ are aligned *without knowing the suffixes* $a_{i+1} \dots a_{L_1}$ and $b_{j+1} \dots b_{L_2}$. This allows in the

second (back-tracing) part of the algorithm to construct an optimal alignment of **a** and **b** using the information gained during the first part of the algorithm.

In our segment-based approach, this additivity does *not* hold—to be precise, the objective function we are using is additive at the level of segment pairs but *not* at the level of individual residue pairs. As mentioned in [2] the *score* $\mathrm{Sc}[i,j]$ of an optimal segment-to-segment alignment of the prefixes $a_1 \ldots a_i$ and $b_1 \ldots b_j$ can be calculated from the values $\mathrm{Sc}[i,j-1]$, $\mathrm{Sc}[i-1,j]$, $\mathrm{Sc}[i-l,j-l]$, $l \geq 1$, and the weights of the fragments *ending* in (i,j). Note that in this context, $\mathrm{Sc}[i,i]$ refers, of course, to our fragment-based objective function rather than to the Needleman-Wunsch scoring scheme. For our fragment-chaining problem, the direct analogue to formula is

$$\mathrm{Sc}[i,j] = \max \left\{ \begin{array}{l} \mathrm{Sc}[i,j-1], \\ \mathrm{Sc}[i-1,j], \\ \max_{l \geq 1}\{\mathrm{Sc}[i-l,j-l] + w(f_{i,j,l})\}, \end{array} \right\} \tag{2}$$

where $f_{i,j,l}$ denotes the fragment of length l ending in (i,j). In a previous paper, we used equation (2) to define a straight-forward dynamic-programming procedure that solves the fragment-chaining problem [2]. Since here, the values $\mathrm{Sc}[i,j]$ need to be stored for the entire dynamic-programming matrix simultaneously, the space complexity of this part of the algorithm was OL^2.

In order to describe a more space-efficient algorithm, we first introduce some more definitions. For a fragment $f \in F$, we define

$$W(f) := \max \left\{ \sum_{i=1}^{K} w(f_i) : f_1 \ll \cdots \ll f_K = f \right\}, \tag{3}$$

where the maximum is taken over all possible chains *ending* in f. If $f_1 \ll \cdots \ll f_K$ is a chain reaching the maximum in equation (3), we call $P(f) = f_{K-1}$ the *predecessor* of f. Finally, we define $\mathrm{Pr}[i,j]$ to be the *last* fragment in an optimal chain of the prefixes $a_1 \ldots a_i$ and $b_1 \ldots b_j$[1], so for a fragment f *starting* in (i,j), we have

$$W(f) = \mathrm{Sc}[i-1,j-1] + w(f), \tag{4}$$

$$P(f) = \mathrm{Pr}[i-1,j-1], \tag{5}$$

$$\mathrm{Sc}[i,j] = \max \left\{ \begin{array}{l} \mathrm{Sc}[i,j-1], \\ \mathrm{Sc}[i-1,j], \\ \max\{W(f) : f \text{ ending in } (i,j)\}, \end{array} \right\} \tag{6}$$

and $\mathrm{Pr}[i,j]$ can be established together with $\mathrm{Sc}[i,j]$ depending on where the maximum in equation (6) is reached as

$$\mathrm{Pr}[i,j] = \left\{ \begin{array}{ll} \mathrm{Pr}[i,j-1], & \text{if } \mathrm{Sc}[i,j] = \mathrm{Sc}[i,j-1], \\ \mathrm{Pr}[i-1,j], & \text{if } \mathrm{Sc}[i,j] = \mathrm{Sc}[i-1,j], \\ \hat{f}, & \text{if } \mathrm{Sc}[i,j] = \max\{W(f) : f \text{ ending in } (i,j)\}(*), \end{array} \right\} \tag{7}$$

where $\hat{f}$ is a fragment maximizing $(*)$.

Once $\mathrm{Sc}[i,j]$ and $\mathrm{Pr}[i,j]$ have been calculated for all positions (i,j) in the comparison matrix, a fragment $f_{\max}$ with $W(f_{\max}) = \max\{W(f), f \in F\}$ is given as $f_{\max} = \mathrm{Pr}[L_1, L_2]$. Now, a trace-back procedure can be used to find an optimal fragment chain by defining

$$f_0 = f_{\max} \quad \text{and} \quad f_{k+1} = P(f_k), \qquad k \geq 0. \tag{8}$$

[1]Strictly spoken, $P(f)$ and $\mathrm{Pr}[i,j]$ are not well defined since there may be several fragments with these properties. For our algorithm, however, this ambiguity is irrelevant.

3. SPACE-EFFICIENT FRAGMENT CHAINING

In this section, a *sparse* dynamic-programming algorithm is described that finds an optimal fragment chain in $O(L+N_{\max})$ space where $N_{\max}$ is a number that is upper-bounded by $\#F$ but is in practical applications far smaller than $\#F$. The comparison matrix is processed column-by-column from the lower-left to the upper-right corner. At every position (i,j), fragments *starting* at (i,j) are considered, and for each fragment $f \in F$, equations (4) and (5) are used to establish $W(f)$ and $P(f)$. To do so, $\mathrm{Sc}[i-1,j-1]$ and $\mathrm{Pr}[i-1,j-1]$ have to be known. The central idea behind our algorithm is to store these values not for the entire comparison matrix simultaneously, but only for one column at a time, so Sc and Pr are encoded as one-dimensional arrays rather than as two-dimensional matrices. Before the fragments starting in column $i+1$ are processed, $\mathrm{Sc}[i,j]$ and $\mathrm{Pr}[i,j]$ are established for all $1 \leq j \leq L_2$ according to equations (6) and (7) using the corresponding values from column $i-1$ together with the sets of all fragments *ending* in (i,j). Therefore, by the time the algorithm reaches column i, $W(f)$ and $P(f)$ have to be know for all fragments ending in this column.

To this end, once a fragment f has been processed, i.e., once $W(f)$ and $P(f)$ have been established, a pointer to f is added to a list $F_{i'}$ that is associated with the column i' where f is *ending*. By the time fragments starting in column $i+1$ are processed, the set F_i of all fragments $f \in F$ ending in column i is therefore already known and can be used to calculate $\mathrm{Sc}[i,j]$ and $\mathrm{Pr}[i,j], 1 \leq j \leq L_2$. Once these values have been established, the corresponding values for column j can be deleted. After the whole dynamic-programming matrix has been processed in this way, the score $\mathrm{Sc}[L_1,L_2]$ of an optimal fragment chain of the input sequences **a** and **b** is known as well as the last fragment $\mathrm{Pr}[L_1,L_2]$ of this chain, and the optimal chain itself can be found by the trace-back procedure (8).

During the described procedure, the values of Pr and Sc are stored for one column at a time. In addition, the sets $F_i, 1 \leq i \leq L_1$ are to be stored, so in the *worst case*, our algorithm requires computer memory proportional to $L + \sum_i \#F_i = L + \#F$.

for $j \leftarrow 0$ **to** L_2 **do**
  $\mathrm{Sc}[0,j] \leftarrow 0$
for $i \leftarrow 1$ **to** L_1 **do**
  for $j \leftarrow 1$ **to** L_2 **do**
    for all $f \in F$ starting in (i,j) **do**
      $W(f) \leftarrow w(f) + \mathrm{Sc}[i-1,j-1]$
      $P(f) \leftarrow \mathrm{Pr}[i-1,j-1]$
      $F_{i'} \leftarrow F_{i'} \cup \{f\}$ where i' is the column where f is ending
    delete $\mathrm{Sc}[i-1,j]$ and $\mathrm{Pr}[i-1,j]$

$$\mathrm{Sc}[i,j] \leftarrow \max \left\{ \begin{array}{l} \mathrm{Sc}[i-1,j], \\ \mathrm{Sc}[i,j-1], \\ \max\{W(f) : f \text{ ending in } (i,j)\} \end{array} \right\} (*)$$

    establish $\mathrm{Pr}[i,j]$ depending on what the maximum in $(*)$ is.
    for all $f \in F$ ending in (i,j) with $f \neq \mathrm{Pr}[i,j]$ **do**
      delete f
    delete $\mathrm{Sc}[i-1,j]$ and $\mathrm{Pr}[i-1,j]$
  $f_0 \leftarrow \mathrm{Pr}[L_1,L_2]$
  while $f_k \neq$ NIL **do** /* trace back */
    $f_{k+1} \leftarrow P(f_{k++})$

Figure 1. Dynamic programming algorithm that calculates fragment chain $f_0 \ll \cdots \ll f_K$ maximizing $\sum_i w(f_i)$ for two sequences of length L_1 and L_2.

It is, however, not necessary to store the entire sets F_i simultaneously. F_i is needed to establish $\mathrm{Sc}[i,j]$ and $\mathrm{Pr}[i,j], 1 \leq j \leq L_2$, so if only the *score* of an optimal fragment chain is to be calculated, F_i can be deleted as soon as $\mathrm{Sc}[i,j]$ and $\mathrm{Pr}[i,j]$ are known. In order to retrieve an optimal chain by the described trace-back procedure, those fragments need to be retained that are potentially contained in this chain. A fragment f can belong to the optimal output chain only if $f = \mathrm{Pr}[i,j]$ holds for some position (i,j) in the comparison matrix which is the case precisely if $f = \mathrm{Pr}[i',j']$ holds for the position (i',j') where f is *ending*. Thus, as soon as $\mathrm{Sc}[i,j]$ and $\mathrm{Pr}[i,j]$ are calculated for a position (i,j), all fragments ending in (i,j) can be deleted except for $\mathrm{Pr}[i,j]$. The *real* space complexity of our algorithm is therefore $O(L+N_{\max})$ where $N_{\max} \leq \#F$ is the maximum number of fragments that are stored *simultaneously* during the dynamic-programming procedure.

Since only fragments with positive weights need to be taken into consideration, F depends, in turn, on the weighting function w. We are considering the function w used in the DIALIGN 2 program which is defined as follows: for a fragment f, $S(f)$ is defined as the sum of similarity values of aligned residue pairs, and $P(f)$ denotes the probability of finding a fragment of the same length as f with at least the same sum $S(f)$ of similarity values in a pair of *random sequences* of the same length as the input sequences **a** and **b**. The weight score of f is then defined as $w(f) = -\ln P(f)$.

It is clear that even for closely related sequences, the vast majority of fragments f in the comparison matrix are unrelated segment pairs with a sum of similarity values $S(f)$ not far from the expectation value. Since the probability $P(f)$ of finding such a fragment in a pair of random sequences is (*very* close to) 1, the weight of a random fragment will be $\approx -\ln 1 = 0$, so these fragments need not be considered for alignment. The set F is further reduced if a threshold T is employed such that only those fragments are considered for alignment that have a weight greater than T.

Moreover, the output of our algorithm is not affected if one ignores all fragments f that properly *contain* a fragment f' with $w(f') \geq w(f)$.[2] We call a fragment f with this property *redundant*. Obviously, if a redundant fragment f belongs to a chain $\mathcal{A}$, then replacing f by f' results in a chain $\mathcal{A}'$ with a score at least as high as $\mathcal{A}$. While it would be time-consuming to identify and exclude *all* redundant fragments, there are simple ways of excluding a substantial part of them without increasing the running time of the algorithm.

(a) Fragments starting at a position (i,j) are processed in order of increasing length, and a fragment is considered for alignment only if its weight exceeds the maximum weight so far of a fragment starting at (i,j).

(b) With our weighting function, a fragment f is redundant whenever its first (or its last) residue pair has the lowest possible similarity value—it is easy to see that omitting this residue pair would result in a fragment f' contained in f with $w(f') \geq w(f)$.

Therefore, for nucleic acid sequences only fragments starting with a match have to be considered. For proteins, one may define a threshold $s_{\min}$ and ignore all fragments with an initial similarity value smaller than $s_{\min}$.

Finally, one may limit the length of fragments under consideration. In DIALIGN, the maximal length for fragments is $l_{\max} = 40$ residue pairs. While this clearly deteriorates the *numerical* score of the resulting output alignments—i.e., the sum of the respective fragment weights—their *biological* quality is unlikely to be affected. Usually, the length of conserved protein domains is far smaller than 40 residues, so only for closely related sequences, high scoring fragments of more than 40 residues in length can be expected. In these situations, however, long fragments in 'biologically correct' alignments can be replaced by consecutive shorter fragments, and 'correct' alignments are still likely to have relatively higher scores than alternative 'wrong' alignments. To

[2] We say that f' is *contained* in f if all residue pairs aligned by f' are also aligned by f—with our previously introduced definition of (partial) alignments as equivalence relations [2,13], this is the usual set-theoretical inclusion relation.

test the influence of these parameters, we have performed systematic test runs on the BAliBASE data base of benchmark alignments [14]. With parameter values $T = 0.5$, $s_{\min} = 8$, $l_{\max} = 30$, the size of the set F was reduced by 81% compared to the results with default parameters $T = 0$, $s_{\min} = 4$, $l_{\max} = 40$, while the quality of the output alignments was reduced only by 2.2%.

The number of fragments considered for alignment depends on the degree of similarity among the input sequences. In order to obtain lower and upper estimates for $\#F$ and $N_{\max}$, we aligned extreme dissimilar as well as extreme similar test sequences, namely pairs of *independent* random DNA sequences and pairs of *identical* random sequences using DIALIGN with default parameters, i.e., with a threshold $T = 0$ for fragment weights and with a maximum fragment length of $l_{\max} = 40$. For independent random sequences, $\#F$ and $N_{\max}$ were negligible compared to L while for identical sequences, $N_{\max}$ was in the order of $L \times l_{\max}$ [12]. Comparison of these results shows that for identical sequences, the vast majority of fragments are contained in the main diagonal of the comparison matrix, so their number will grow approximately linearly with the sequence length.

REFERENCES

1. 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).
2. B. Morgenstern, A.W.M. Dress and T. Werner, Multiple DNA and protein sequence alignment based on segment-to-segment comparison, *Proc. Natl. Acad. Sci. USA* **93**, 12098–12103, (1996).
3. B. Morgenstern, DIALIGN 2: Improvement of the segment-to-segment approach to multiple sequence alignment, *Bioinformatics* **15**, 211–218, (1999).
4. S. Abdeddaïm and B. Morgenstern, Speeding up the DIALIGN multiple alignment program by using the 'greedy alignment of biological sequences library' (GABIOS-LIB), In *Proceedings of the Journées Ouvertes: Biologie, Informatique et Mathématiques (JOBIM), Lecture Notes in Computer Science*, (in press).
5. H.-P. Lenhof, B. Morgenstern and K. Reinert, An exact solution for the segment-to-segment multiple sequence alignment problem, *Bioinformatics* **15**, 203–210, (1999).
6. J.D. Kececioglu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert and M. Vingron, A polyhedral approach to sequence alignment problems, *Discrete Applied Mathematics* **104**, 143–186, (2000).
7. J.W. Wilbur and D.J. Lipman, Rapid similarity searches of nucleic acid and protein data banks, *Proc. Natl. Acad. Sci. USA* **80**, 726–730, (1983).
8. K.-M. Chao and W. Miller, Linear-space algorithms that build local alignments from fragments, *Algorithmica* **13**, 106–134, (1995).
9. D. Eppstein, Z. Galil, R. Giancarlo and G. Italiano, Sparse dynamic programming I: Linear cost functions, *J. Assoc. Comput. Mach.* **39**, 519–545, (1992).
10. D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Commun. ACM* **18**, 314–343, (1975).
11. E.M. Myers and W. Miller, Optimal alignments in linear space, *CABIOS* **4**, 11–17, (1988).
12. B. Morgenstern, A space-efficient algorithm for aligning large genomic sequences, *Bioinformatics* **16**, 948–949.
13. B. Morgenstern, J. Stoye and A.W.M. Dress, Consistent equivalence relations: A set-theoretical framework for multiple sequence alignment, Materialien und Preprints 133, University of Bielefeld, (1999).
14. J.D. Thompson, F. Plewniak, and O. Poch, BAliBASE: A benchmark alignment database for the evaluation of multiple sequence alignment programs, *Bioinformatics* **15**, 87–88, (1999).