CABIOS

Vol.9. no.6. 1993
Pages 729 – 734

# An efficient algorithm to locate all locally optimal alignments between two sequences allowing for gaps

Geoffrey J.Barton

## Abstract

*An efficient algorithm is described to locate locally optimal alignments between two sequences allowing for insertions and deletions. The algorithm is based on that of Smith and Waterman which returns the single best local alignment. However, the algorithm described here permits all non-intersecting locally optimal alignments to be determined in a single pass through the comparison matrix. The algorithm simplifies the location of repeats, multiple domains and shuffled motifs, and is fast enough to be used on a conventional workstation to scan large sequence databanks.*

## Introduction

Dynamic programming algorithms that locate optimal alignments of two sequences are central techniques for the comparison of biological sequences (Needleman and Wunsch, 1970; Sellers, 1974; Smith and Waterman, 1981) or three-dimensional structures (Barton and Sternberg, 1988; Taylor and Orengo, 1989; Sali and Blundell, 1990; Russell and Barton, 1992). The algorithms can be divided broadly into those that seek to find a *global* alignment between the sequences (e.g. Needleman and Wunsch, 1970) and those that find *local* alignments (e.g. Smith and Waterman, 1981; Erickson and Sellers, 1983). Global alignment methods optimize the score for alignment over the full length of both sequences, and are most appropriate when the sequences are known to be similar over their entire length. Local alignment methods allow the common subregions of the two sequences to be identified and are appropriate when it is not known in advance if the sequences being compared are similar. Local alignment methods are effective in locating common subdomains between long sequences that otherwise share little similarity. This feature makes such algorithms suitable for scanning large sequence databanks for similarities to a newly determined sequence.

The Smith and Waterman (1981) algorithm is perhaps the most widely used local similarity algorithm for biological sequence comparison. The algorithm identifies the single highest scoring subsequence alignment and allows for gaps (insertions/deletions). However, it is often true that there may be more than one biologically important alignment between two sequences. For example, a protein domain may be repeated, or domains may be shuffled within multidomain proteins. Waterman and Eggert (1987) have shown how the Smith–Waterman algorithm may be extended to locate the second-best and subsequent local alignments with minimal recalculation, subject to the primary restriction that the different alignments should not intersect. Here, I describe an algorithm that allows all such locally optimal alignments to be determined without the need for recalculation. The algorithm is similar in principle to that developed by Coulson *et al.* (1987) for the parallel processing Distributed Array Processor (DAP). However, the algorithm described here is general and has been implemented in C for widely available computers. The algorithm may be applied to any problem that is amenable to the Smith–Waterman dynamic programming algorithm.

## Efficient determination of best score

In order to simplify the explanation of the 'all local alignment' algorithm I shall first recapitulate a well-known method for efficient computer implementation of the Smith–Waterman algorithm. For a full introduction to dynamic programming algorithms in sequence comparison see Kruskal (1983).

The best score for a local alignment between two sequences $A$ and $B$ of length $m$ and $n$ is determined by calculating the comparison matrix $H_{m,n}$ starting with $H_{1,1}$ and working forwards through the matrix column by column. The value of each cell $H_{i,j}$ is given by the equation:

$$H_{i,j} = \max \begin{Bmatrix} H_{i-1,j-1} + w_{A_i,B_j} \\ H_{i,j-1} + w_{A_i,\Delta} \\ H_{i-1,j} + w_{\Delta,B_j} \\ 0 \end{Bmatrix}$$

where $w_{A_i,B_j}$ is the score for equivalencing $A_i$ and $B_j$, and $w_{A_i,\Delta}, w_{\Delta,B_j}$ are the scores for aligning with a gap in $B$ or $A$ respectively. To calculate the value of $H_{i,j}$, it is only necessary to know the contents of the three predecessor cells $(H_{i-1,j-1}, H_{i,j-1}, H_{i-1,j})$. If just the best score is required, and no alignment, then only the previous column, **C**, of the matrix need to be stored together with the score for the current cell, $r$ and previous cell, $p$ in the column. This is illustrated in Figure 1 for the comparison of A = C-C-A-A-T-C-T-A-C-T-A-C-T-G-C-T-T-G-C-A-G-T-A-C and B = A-G-T-C-C-G-A-

*Laboratory of Molecular Biophysics, University of Oxford, Rex Richards Building, South Parks Road, Oxford OX1 3QU, UK*

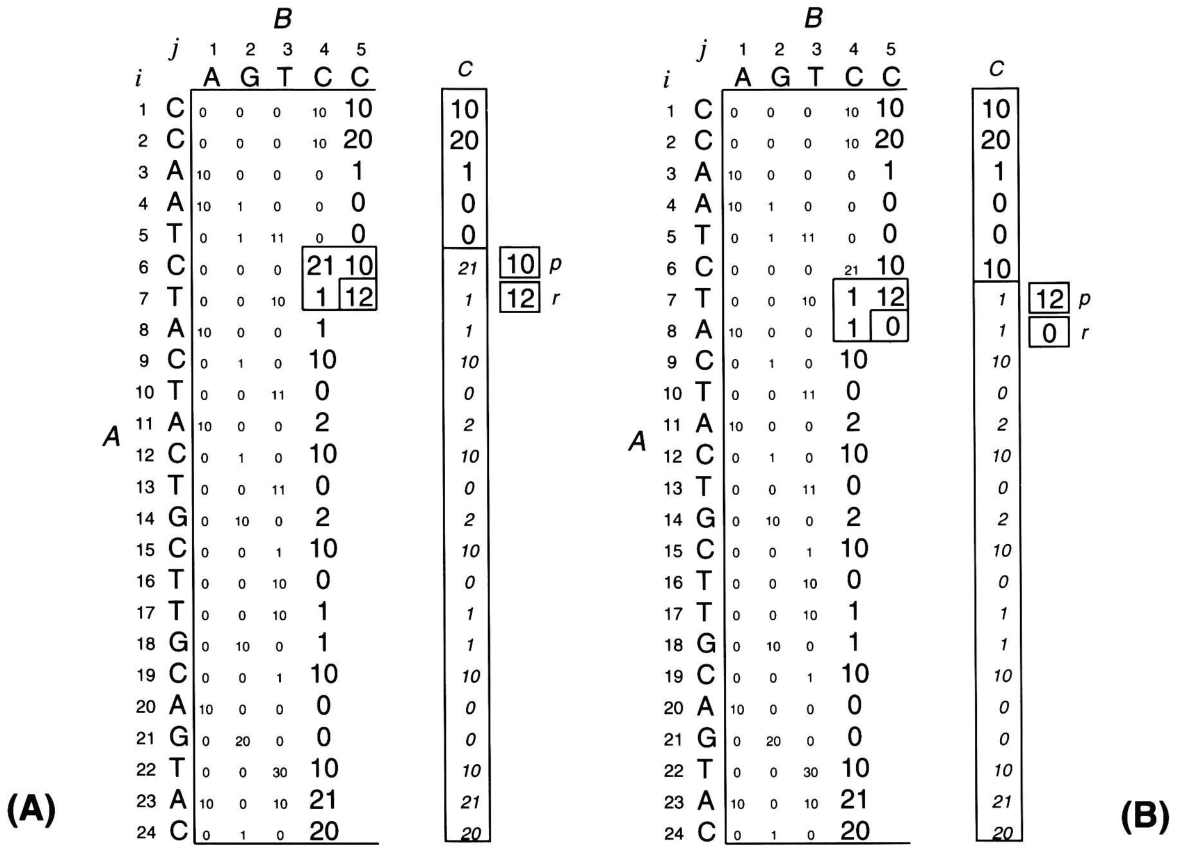
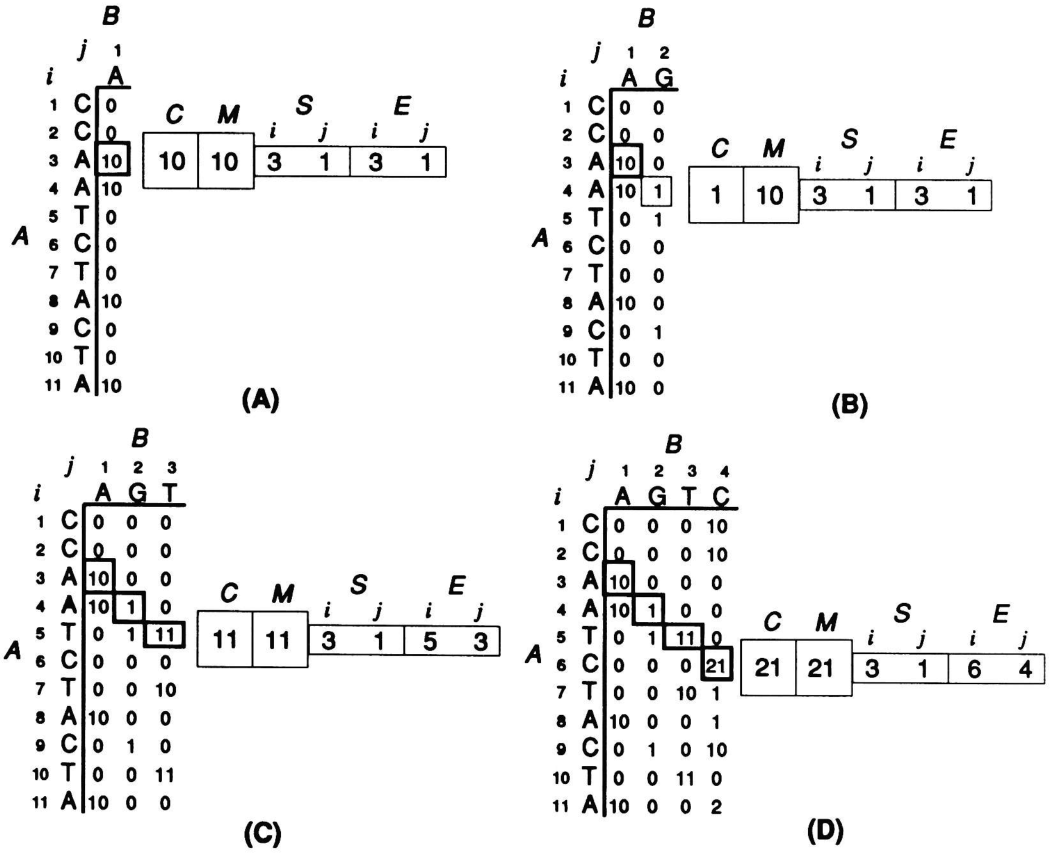


Fig. 1. Efficient determination of best score. Two steps in the processing of the $H$ matrix for the comparison of sequences $A$ and $B$ (see text). Only the values shown in large numerals are stored using a single vector $C$ and two scalar values $p$ and $r$.



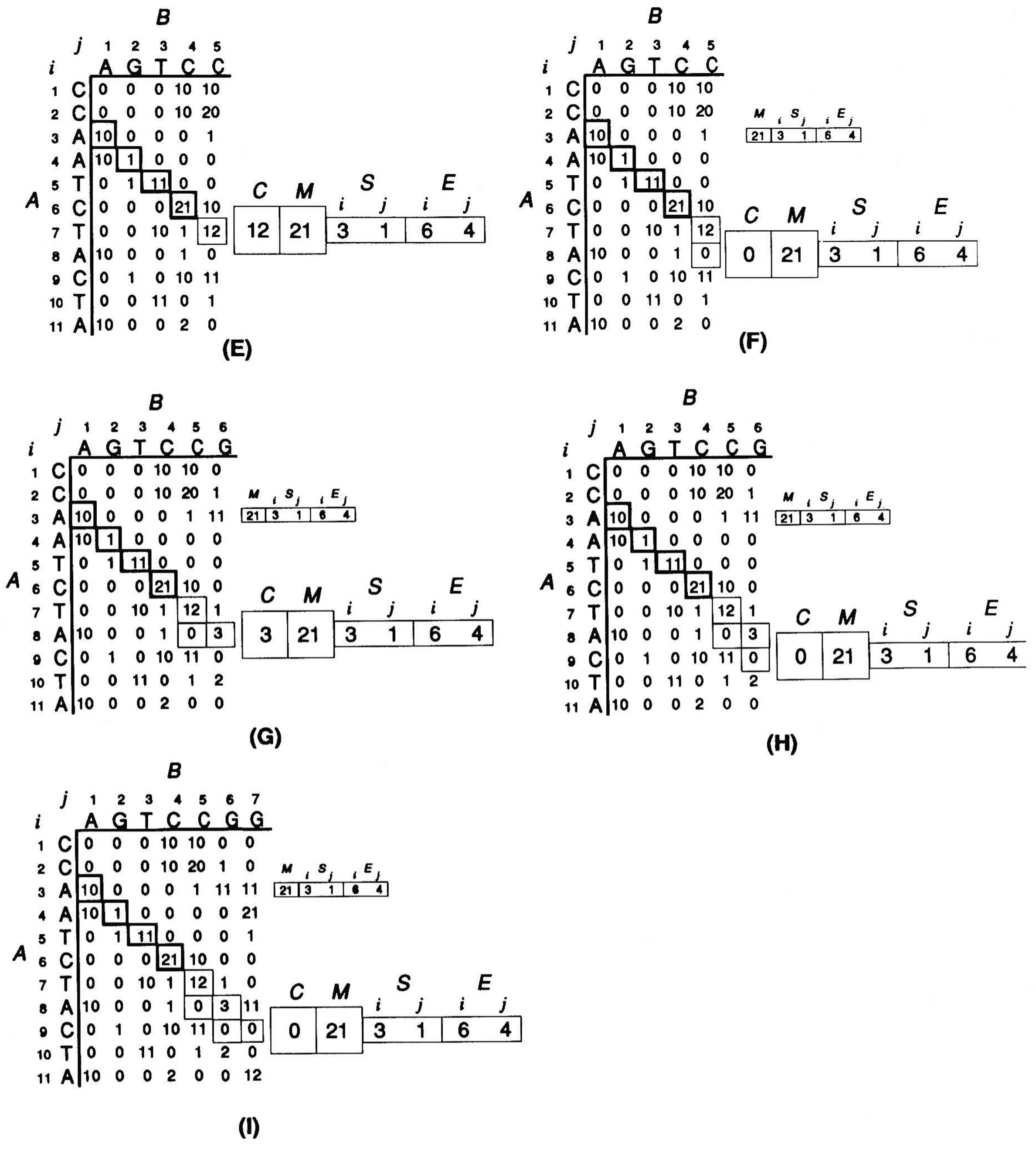


**Fig. 2.** Finding the first locally optimal alignment without recalculation of $H$. Calculation of the $H$ matrix. The vector $C$ shown in Figure 1 is joined by a further five vectors to store the maximum score $M$ on the current path, and the start $S$ and current end-point $E$ for the optimal alignment. To simplify the figure, only one element of each vector is illustrated. The subfigures show the building up of the score, start and end point for the first locally optimal alignment to be found when processing the $H$ matrix. In each subfigure, the heavy-boxed cells of $H$ have been assigned to the optimal alignment. The lightly boxed cells lie on the alignment path, but may follow the current maximum cell $M$. In (f), the score, start-points and end-points of the locally optimal alignment has been stored in the results list, indexed by the row in which the alignment starts (3).

G-G-G-C-T-A-C-T-C-T-A-C-T-G-A-A-C with $w_{A_i,B_j} = 10$ if $A_i = B_j$, $w_{A_i,B_j} = -9$ if $A_i \neq B_j$, and $w_{A_i,\Delta} = w_{\Delta,B_j} = -20$. This example is used here to allow a direct comparison of the 'all local alignment' algorithm with the work of Waterman and Eggert (1987).

Figure 1 illustrates the processing of $H_{7,5}$ and $H_{8,5}$. The cells of $H$ that are stored are shown in large numerals. The cells shown in small numerals are discarded. The best overall score is updated if the value of the current cell is greater than the maximum so far. This is the simplest efficient implementation of the 'best score only' local similarity algorithm, and may be coded to run very fast. It is also straightforward, by the addition of a further one-dimensional array, to adapt the algorithm to cope with a gap-penalty function having the form $w_k = u_k + v$ where $k$ is the gap-length (Gotoh, 1982).

## All local alignment algorithm

Smith and Waterman (1981) identified the best local alignment by storing the entire $H$ matrix, finding the maximum element, then tracing back through the matrix. Waterman and Eggert (1987) described an algorithm to identify alternative locally optimal alignments by partial recalculation of the $H$ matrix subject to the condition that the alignment paths do not intersect, and that the first and last equivalenced pairs have a positive score. Here, I show that for a length-dependent gap-penalty the scores for all such locally optimal alignments can be obtained

```
1: Score: 62.0

   1 CCAATCTACT    10
  11 CTACTCTACT    20

2: Score: 61.0

   6 CTACTACTGCT   16
  11 CTACT CTACT   20

3: Score: 60.0

   9 CTACTG    14
  16 CTACTG    21

4: Score: 50.0

   9 CTACT    13
  11 CTACT    15

5: Score: 31.0

  20 AGTAC    24
   1 AGTCC     5

6: Score: 30.0

  22 TAC    24
  12 TAC    14

7: Score: 30.0

  14 GCT    16
  10 GCT    12

8: Score: 30.0

  22 TAC    24
  17 TAC    19

9: Score: 21.0

   3 AATC    6
   1 AGTC    4

10: Score: 21.0

   1 CCAA    4
   4 CCGA    7

11: Score: 21.0

  12 CTGC    15
  11 CTAC    14

12: Score: 20.0

   8 AC    9
  23 AC   24

13: Score: 20.0

  18 GC    19
  10 GC    11

14: Score: 20.0

  20 AG    21
   7 AG     8

15: Score: 20.0

   3 AA     4
  22 AA    23

16: Score: 12.0

   8 ACTAC    12
   1 AGTCC     5

17: Score: 12.0

   1 CCAAT     5
  16 CTACT    20

18: Score: 12.0

   7 TACTA    11
   3 TCCGA     7

19: Score: 12.0

  11 ACTGC    15
   1 AGTCC     5

20: Score: 11.0

  19 CAG    21
   4 CCG     6

21: Score: 11.0

   2 CAA     4
  16 CTA    18

22: Score: 11.0

  17 TGC    19
  12 TAC    14

23: Score: 11.0

  17 TGC    19
   3 TCC     5

24: Score: 11.0

   6 CTA     8
   5 CGA     7

25: Score: 11.0

   2 CAA     4
  11 CTA    13

26: Score: 11.0

  12 CTG    14
   4 CCG     6

27: Score: 11.0

   4 ATC     6
  22 AAC    24

28: Score: 11.0

   1 CCA     3
   5 CGA     7
```

**Fig. 3.** The 28 locally optimal alignments found between *A* and *B*. The boxed alignments all score ≥20 and their paths are shown in Figure 4.

on a single pass through $H$. The essential observation is that since alignment paths are not allowed to intersect, it is only possible to have one path passing through each cell $H_{i,j}$. Therefore, when processing $H$ it is simply necessary to maintain a record of the starting residues and best score for the alignment that passes through the current cell $H_{i,j}$. The algorithm requires storage for the column scores $C$ as for the single best-score algorithm. In addition, the current maximum path scores $M$, and the start and end of the local alignment, $S$, $E$, where $S$ and $E$ hold the coordinates of the cells in $H$ where the alignment starts and ends must also be stored.

Figure 2(a)−(i) illustrates the processing of $H$ to find the first, but not the highest scoring, locally optimal alignment between the sequences. For the sake of clarity, only the elements of $C$, $M$, $S$ and $E$ that are relevant to this alignment are illustrated in each figure. Figure 2(a) shows the first cell in the alignment $H_{3,1} = 10$, this is also the maximum score for the alignment so far, and the alignment starts and ends with the same pair of residues $S_3 = E_3 = (3,1)$. In Figure 2(b), the alignment is continued, but since the current cell, $H_{4,2} = 1(<[M_4 = 10])$, $S_4$, $E_4$ and $M_4$ are left unchanged. In Figure 2(c), $H_{5,3} = 11$, so $E_5$ and $M_5$ are updated to 11 and (5,3) respectively. Similar processing is shown in Figure 2(d) and (e) where $C$, $M$, $S$ and $E$ are updated to the values 12, 21, (3,1), (6,4), while Figure 2(f) illustrates termination of the path when $H_{8,5} = 0$. Now that this local alignment path has decayed to zero, the starting and ending coordinates are saved together with the alignment score. However, since we have not completed the processing of $H$, we do not yet know if this is the best possible score for an alignment starting in $H_{3,1}$. Figure 2(f)−(i) show that the alignment cannot be extended further, so the maximum score of 21 for an alignment starting in $H_{3,1}$ stands. Had it been possible for the alignment to be extended, then the new best-score and end-point ($M$ and $E$) would have replaced the 21, (6,4) currently saved on the results list. Once the entire matrix has been calculated (but not stored), the results list contains the score, start and end coordinates for all local alignments between the two sequences. Although not essential for the functioning of the algorithm, it is often desirable to set a minimum score threshold $T_s$ such that only those local alignments where $M > T_s$ will be saved on the results list.

If it is necessary to generate the alignments rather than just report the best scores and end-points, then a direction matrix $e_{m,n}$ (Smith *et al.*, 1981; Gotoh, 1982) is saved during the calculation of $H$ such that each element $e_{i,j}$ indicates which of $H_{i-1,j-1}$, $H_{i,j-1}$ or $H_{i-1,j}$ contributed to the value of $H_{i,j}$. Accordingly, $e_{m,n}$ may be a compact data type of as few as three bits per element, so the memory overhead for finding all locally optimal alignments between long sequences is modest compared to algorithms that require storage of $H$. Extension of the algorithm to allow gap-penalties of the form $u_k + v$ is straightforward if only best scores and end-points are required,

| | *j* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *i* | | A | G | T | C | C | G | A | G | G | G | C | T | A | C | T | C | T | A | C | T | G | A | A | C |
| 1 | C | 0 | 0 | 0 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 10 | 0 | 10 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 10 |
| 2 | C | 0 | 0 | 0 | 10 | 20 | 1 | 0 | 0 | 0 | 0 | 10 | 1 | 0 | 10 | 1 | 10 | 1 | 0 | 10 | 1 | 0 | 0 | 0 | 10 |
| 3 | A | 10 | 0 | 0 | 0 | 1 | 11 | 11 | 0 | 0 | 0 | 0 | 1 | 11 | 0 | 1 | 0 | 1 | 11 | 0 | 1 | 0 | 10 | 10 | 0 |
| 4 | A | 10 | 1 | 0 | 0 | 0 | 0 | 21 | 2 | 0 | 0 | 0 | 0 | 11 | 2 | 0 | 0 | 0 | 11 | 2 | 0 | 0 | 10 | 20 | 1 |
| 5 | T | 0 | 1 | 11 | 0 | 0 | 0 | 1 | 12 | 0 | 0 | 0 | 10 | 0 | 2 | 12 | 0 | 10 | 0 | 2 | 12 | 0 | 0 | 1 | 11 |
| 6 | C | 0 | 0 | 0 | 21 | 10 | 0 | 0 | 0 | 3 | 0 | 10 | 0 | 1 | 10 | 0 | 22 | 2 | 1 | 10 | 0 | 3 | 0 | 0 | 11 |
| 7 | T | 0 | 0 | 10 | 1 | 12 | 1 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | 0 | 20 | 2 | 32 | 12 | 0 | 20 | 0 | 0 | 0 | 0 |
| 8 | A | 10 | 0 | 0 | 1 | 0 | 3 | 11 | 0 | 0 | 0 | 0 | 0 | 30 | 10 | 0 | 11 | 12 | 42 | 22 | 2 | 11 | 10 | 10 | 0 |
| 9 | C | 0 | 1 | 0 | 10 | 11 | 0 | 0 | 2 | 0 | 0 | 10 | 0 | 10 | 40 | 20 | 10 | 2 | 22 | 52 | 32 | 12 | 2 | 1 | 20 |
| 10 | T | 0 | 0 | 11 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | 20 | 50 | 30 | 20 | 2 | 32 | 62 | 42 | 22 | 2 | 0 |
| 11 | A | 10 | 0 | 0 | 2 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 30 | 10 | 30 | 41 | 21 | 30 | 12 | 42 | 53 | 52 | 32 | 12 |
| 12 | C | 0 | 1 | 0 | 10 | 12 | 0 | 0 | 3 | 0 | 0 | 10 | 0 | 10 | 40 | 20 | 40 | 32 | 12 | 40 | 22 | 33 | 44 | 43 | 42 |
| 13 | T | 0 | 0 | 11 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 20 | 0 | 20 | 50 | 30 | 50 | 30 | 20 | 50 | 30 | 24 | 35 | 34 |
| 14 | G | 0 | 10 | 0 | 2 | 0 | 11 | 0 | 10 | 10 | 10 | 0 | 0 | 11 | 0 | 30 | 41 | 30 | 41 | 21 | 30 | 60 | 40 | 20 | 26 |
| 15 | C | 0 | 0 | 1 | 10 | 12 | 0 | 2 | 0 | 1 | 1 | 20 | 0 | 0 | 21 | 10 | 40 | 32 | 21 | 51 | 31 | 40 | 51 | 31 | 30 |
| 16 | T | 0 | 0 | 10 | 0 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 30 | 10 | 1 | 31 | 20 | 50 | 30 | 31 | 61 | 41 | 31 | 42 | 22 |
| 17 | T | 0 | 0 | 10 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 21 | 1 | 11 | 22 | 30 | 41 | 21 | 41 | 52 | 32 | 22 | 33 |
| 18 | G | 0 | 10 | 0 | 1 | 0 | 10 | 0 | 10 | 10 | 10 | 0 | 0 | 1 | 12 | 0 | 2 | 13 | 21 | 32 | 21 | 51 | 43 | 23 | 13 |
| 19 | C | 0 | 0 | 1 | 10 | 11 | 0 | 1 | 0 | 1 | 1 | 20 | 0 | 0 | 11 | 3 | 10 | 0 | 4 | 31 | 23 | 31 | 42 | 34 | 33 |
| 20 | A | 10 | 0 | 0 | 0 | 1 | 2 | 10 | 0 | 0 | 0 | 0 | 11 | 10 | 0 | 2 | 0 | 1 | 10 | 11 | 22 | 14 | 41 | 52 | 32 |
| 21 | G | 0 | 20 | 0 | 0 | 0 | 11 | 0 | 20 | 10 | 10 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 32 | 21 | 32 | 43 |
| 22 | T | 0 | 0 | 30 | 10 | 0 | 0 | 2 | 0 | 11 | 1 | 1 | 10 | 0 | 0 | 11 | 0 | 10 | 0 | 0 | 11 | 12 | 23 | 12 | 23 |
| 23 | A | 10 | 0 | 10 | 21 | 1 | 0 | 10 | 0 | 0 | 2 | 0 | 0 | 20 | 0 | 0 | 2 | 0 | 20 | 0 | 0 | 2 | 22 | 33 | 13 |
| 24 | C | 0 | 1 | 0 | 20 | 31 | 11 | 0 | 1 | 0 | 0 | 12 | 0 | 0 | 30 | 10 | 10 | 0 | 0 | 30 | 10 | 0 | 2 | 13 | 43 |

**Fig. 4.** The completed $H$ matrix with the paths for the top 15 alignments highlighted.

but generation of the corresponding alignments will require two passes through $H$ (Gotoh, 1982).

Figure 3 shows the 28 locally optimal alignments that are found between the sequences $A$ and $B$. Alignments of length 1 are normally uninteresting and so are excluded. The two alignments illustrated by Waterman and Eggert (1987) are ranked 1 and 2, with a further six alignments scoring $\geq 30$. A total of 15 alignments score $\geq 20$, with the remaining 13 optimal alignments scoring $\leq 12$. Figure 4 illustrates the full $H$ matrix with the paths highlighted that correspond to the 15 alignments scoring $\geq 20$.

## Implementation and efficiency

The 'all local alignment' algorithm has been implemented in C. This language allows the $C$, $M$, $S$ and $E$ arrays to be grouped into a single data structure array of length $m$. The resulting code is faster than when $C$, $M$, $S$ and $E$ are coded separately, since sequentially accessed values are adjacent in memory. The results list is also of length $m$ where each element points to a dynamically allocated 'ragged' list of structures containing values for $M$, $S$ and $E$.

As a check of the relative efficiency of the 'all local alignment' algorithm, the protein sequence of human $\alpha$-haemoglobin (141 residues, PIR code HAHU) was compared to a small sequence databank (PIR 14.0: 6858 sequences, 2 080 148 amino acids) using the Dayhoff MDM250 matrix (Dayhoff *et al.*, 1978) and gap-penalty of 8 ($w_{Ai,\Delta} = w_{\Delta,Bj} = -8$). When run on a Sun SPARCstation 2, the efficient 'best score only' algorithm determined one optimal local alignment for each databank sequence, and required 270 s to complete the scan. Another implementation of the Smith–Waterman algorithm SSEARCH (Pearson, 1992) which also only returns the top scoring alignment for each sequence pair, required 1100 s for the same scan. In contrast, the 'all local alignment' algorithm described here with $T_s = 35$ examined 30 690 local alignments in only 1000 s.

The value of $T_s$ has little effect on the execution time, except when set very small such that large numbers of local alignments must be stored and sorted for each sequence comparison. For example, comparison of HAHU (141 residues) to one of the longest protein sequences known, Twitchin from *Caenorhabditis elegans* (PIR code S07571, 6048 residues), finds 32 299 alternative local alignments when $T_s = 0$ and requires 16 s CPU time. Setting $T_s = 35$ reduces the number of alignments to 58 and takes only 2 s. The algorithms described here have also been implemented to work with *pscan* (Barton, 1991) to allow distributed processing on a network of workstations. *pscan* permits an approximately linear decrease in elapsed scan time with increasing numbers of processors.

## Availability

An ANSI-C subroutine library that includes the 'all local alignment' code and utility routines is available from the author. The files are quite small and can be e-mailed. Send requests to gjb@bioch.ox.ac.uk. Alternatively, please send a DOS formatted 3.5 in. disk with suitable packaging and a return address label.

## Acknowledgements

I thank the Royal Society and Professor L.N.Johnson for support, and a referee for pointing out the similarities between this work and that of Coulson *et al.* (1987).

## References

Barton,G.J. (1991) Scanning the protein sequence databank using a distributed processing workstation network. *Comput. Applic. Biosci.*, **7**, 85–88.

Barton,G.J. and Sternberg,M.J.E. (1988) Lopal and scamp: Techniques for the comparison and display of protein sequences. *J. Mol. Graph.*, **6**, 190–196.

Coulson,A.F.W., Collins,J.F. and Lyall,A. (1987) Protein and nucleic acid sequence database searching: a suitable case for parallel processing. *Comput. J.*, **30**, 420–424.

Dayhoff,M.O., Schwartz,R.M. and Orcutt,B.C. (1978) A model of evolutionary change in proteins. Matrices for detecting distant relationships. In Dayhoff,M.O. (ed.), *Atlas of Protein Sequence and Structure.* National Biomedical Research Foundation, Washington, DC, Vol. 5, pp. 345–358.

Erickson,B.W. and Sellers,P.H. (1983) Recognition of patterns in genetic sequences. In Sankoff,D. and Kruskal,J.B. (eds), *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, pp. 55–91.

Gotoh,O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.*, **162**, 705–708.

Kruskal,J.B. (1983) An overview of sequence comparison. In Sankoff,D. and Kruskal,J.B. (eds), *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, pp. 1–44.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Pearson,W. (1992) Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith–Waterman and FASTA algorithms. *Genomics*, **11**, 635–650.

Russell,R.B. and Barton,G.J. (1992) Multiple protein sequence alignment from tertiary structure comparison: assignment of global and residue confidence levels. *Proteins: Struct., Funct., Genet.*, **14**, 309–323.

Sali,A. and Blundell,T.L. (1990) Definition of general topological equivalence in protein structures a procedure involving comparison of properties and relationships through simulated annealing and dynamic programming. *J. Mol. Biol.*, **212**, 403–428.

Sellers,P.H. (1974) On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, **26**, 787–793.

Smith,T.F.and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.

Smith,T.F., Waterman,M.S. and Fitch,W.M. (1981) Comparative biosequence metrics. *J. Mol. Evol.*, **18**, 38–46.

Taylor,W. and Orengo,C. (1989) Protein structure alignment. *J. Mol. Biol.*, **208**, 1–21.

Waterman,M.S. and Eggert,M. (1987) A new algorithm for best subsequence alignments with application to trna-rrna comparisons. *J. Mol. Biol.*, **197**, 723–728.

*Received on April 30, 1993; accepted on June 29, 1993*

Circle No. 15 on Reader Enquiry Card