Creating Optimal Sequence Alignment in Order N^2 Time.

Paul Haeberli

University of Wisconsin - Madison

August 26,1981

ABSTRACT

This short paper describes an algorithm which allows the optimal alignment of two DNA sequences of length N to be determined in order N^2 time using the Needleman-Wunsch or Sellers methods. The classical implementation of these methods are of order N^3. This new algorithm removes some limitations on the longest sequences which can be compared. Although this algorithm is applicable to either method, this paper focuses on that of Needleman and Wunsch.

Key words: Alignment, Deletion, Gaps, Homology, Insertion, Longest common substring, Matrix, Metric, Needleman-Wunsch, Sellers, Sequences.

The ablility to align biological sequences plays an important role in investigations of sequence relatedness. The question often asked is what are the minimum number of substitutions, insertions, and deletions needed to transform one sequence into another seemingly related sequence. To address this question, the concept of alignment quality is used to arrive at an optimal alignment. The optimal alignment will have the largest number of matched bases and the fewest insertions and deletions. These insertions and

deletions correspond to placing gaps in one sequence or the other. The actual weighting of gaps and gap lengths is up to the researcher.

If the gap weight is 3 and the gap length weight is 0, The insertion of any gap (independent of its length) must result in at least 3 more matches between the two sequences to be justified. Fig 1. is an example showing how gaps are inserted to arrive a the best alignment.

```
                Before                              After

Seq 1:  AAGCCCATGTATCAATGAGTA     -->     AAGCCCATGTATCAA..TGAGTA
        |||||  | |     ||||  |             |||||  ||||||||  ||||  |
Seq 2:  AAGCCTGTATCAACGTGAGCA     -->     AAGCC..TGTATCAACGTGAGCA
```

Fig 1. The insertion of gaps to obtain the best alignment of two sequences.

To create the optimal alignment of two sequences S1 and S2 of length N, all possible alignments and gap positions are considered. The quality of a particular alignment may be quantitized by summing the matches, while deducting for every gap and its length. The optimal alignment is found by a relatively simple search procedure.

We create a matrix of the form shown in fig 2. Each vertex $M_{i,j}$ in this matrix corresponds to the pairing of one base from the horizontal sequence S1(i) and one base from the vertical sequence S2(j). The result of the search indicates which bases should be paired. This in turn determines where gaps should be introduced.

We proceed row by row from the bottom to the top. The alignment quality $Q_{i,1}$ of every pairing in row 1 is determined by the match of corresponding bases from the two sequences $M_{i,1}$.

$$Q_{i,1} = \text{Match}(S1(i), S2(1)) \qquad 1 <= i <= N$$

$$\text{Where: Match}(\text{base1}, \text{base2}) = \begin{cases} 1 & \text{If base1 = Base2} \\ 0 & \text{If base1} \neq \text{Base2} \end{cases}$$

The match qualities $Q_{i,j}$ are shown subscripting each pairing. The first (left most) pairing of each row is handled similarly to those in the first row.

For the remaining N-1 pairings of the remaining N-1 Rows, the following comparisons are made to determine the highest cumulative alignment quality $Q_{i,j}$ that can be attained from the pairing of these two bases. In this way, the results generated in previous rows are used to determine $Q_{i,j}$. When $Q_{i,j}$ is calculated, a path is recorded in another matrix $P_{i,j}$ connecting the current pairing to the pairing which was found to contribute to $Q_{i,j}$.

$$\text{BestAcross} = \underset{1 < m < i}{\text{Maximum}}(Q_{m-1,j-1} - \text{GapWgt} - \text{GapLenWgt}*(i-m))$$

$$\text{BestDown} = \underset{1 < m < j}{\text{Maximum}}(Q_{i-1,m-1} - \text{GapWgt} - \text{GapLenWgt}*(j-m))$$

$$Q_{i,j} = \text{Match}(S1(i), S2(j)) + \text{Maximum}(\text{BestAcross}, \text{BestDown}, Q_{i-1,j-1})$$

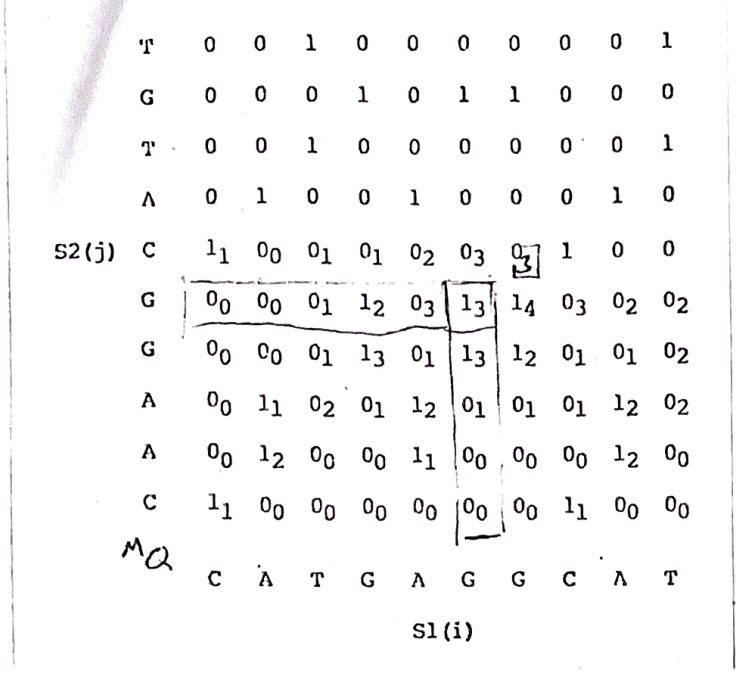

Fig 2. Determining $Q_{7,6}$. The gap weight is 1 and the gap length weight is 0. We apply the equations:

BestAcross = Maximum(0-1, 0-1, 1-1, 2-1, 3-1) = 2

BestDown = Maximum(0-1, 0-1, 1-1, 3-1) = 2

$Q_{7,6}$ = Match(G, C) + Maximum(2, 2, 3) = 0 + 3 = 3

$P_{7,6}$ is set to point to the diagonal.

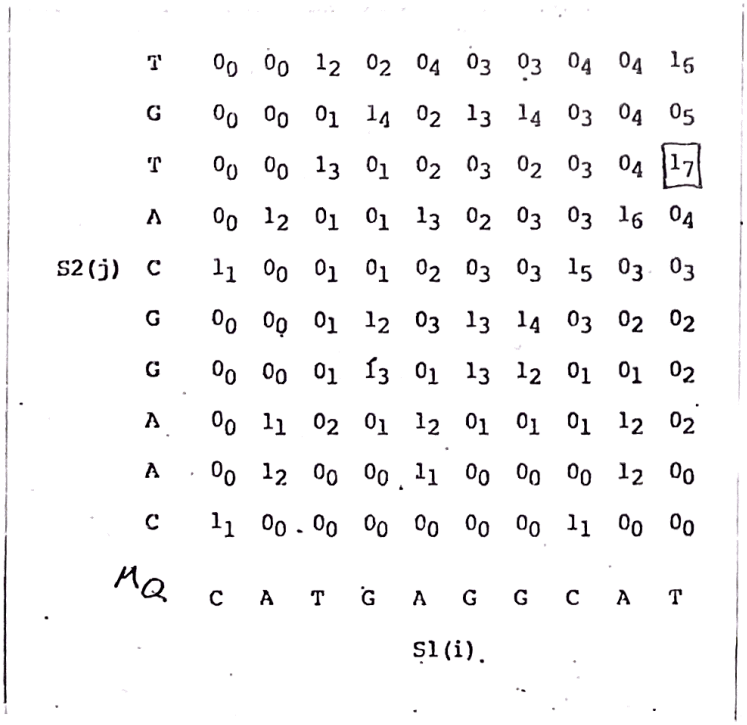

Fig 3. The completed Q and P matricies. The optimal alignment path (darkened) starts at the Q element with the largest value. Base pairs are created as this path is followed to generate the corresponding alignment of fig 4.

When all rows are complete as in fig 3., the actual alignment is created by starting at the pairing with the highest alignment quality (this will always be in the top row, or the right column), making these a pair, then following the best out going path $P_{i,j}$ to the next pairing. If the path is not directly diagonal a gap is created in one sequence. This builds the alignment of fig 4. from the right to the left.

```
                Before                      After

S1:  CATGAGGCAT      -->      CATGAGGCAT..
     || |     |                ||  ||||||
S2:  CAAGGCATGT      -->      CA..AGGCATGT
```

Fig 4. The insertion of gaps to obtain the best alignment of S1 and S2.

The number of computations required to perform the algorithm above increases as N^3, since the number of alignment qualities to be found is N^2, and often more than N quantities need comparison for each result. If N is 1000, About 10^9 comparisons are needed. However, by eliminating some redundancy in the comparisons, the identical result can be found in order N^2 time.

To see how this can be done, think about the comparisons performed as we calculate $Q_{i,j}$ on one row. Suppose the gap weight is 3 and the gap length weight is 1.

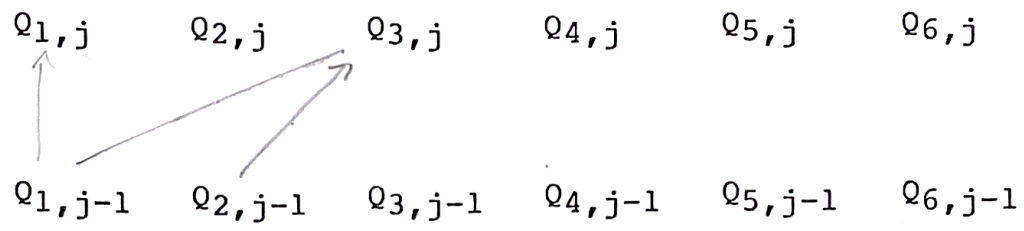

Fig 5. Two rows of the alignment quality matrix.

$Q_{1,j-1}$, $Q_{2,j-1}$,...$Q_{6,j-1}$ are the alignment qualities of the previous row. As part of the calculation to find the maximum alignment qualities for $Q_{1,j}$, $Q_{2,j}$, ..., $Q_{6,j}$ the following comparisons are made.

$$BestAcross_{3,j} = Max(Q_{1,j-1} - 4),$$
$$BestAcross_{4,j} = Max(Q_{1,j-1} - 5, Q_{2,j-1} - 4),$$
$$BestAcross_{5,j} = Max(Q_{1,j-1} - 6, Q_{2,j-1} - 5, Q_{3,j-1} - 4),$$
$$BestAcross_{6,j} = Max(Q_{1,j-1} - 7, Q_{2,j-1} - 6, Q_{3,j-1} - 5, Q_{4,j-1} - 4).$$

To evaluate these for one row takes on the order of $N^2/2$ operations, but these expressions can be evaluated in a way which uses the result of the previous comparison!

$$BestAcross_{3,j} = Max(Q_{1,j-1} - 4),$$
$$BestAcross_{4,j} = Max(BestAcross_{3,j} - 1, Q_{2,j-1} - 4),$$
$$BestAcross_{5,j} = Max(BestAcross_{4,j} - 1, Q_{3,j-1} - 4),$$
$$BestAcross_{6,j} = Max(BestAcross_{5,j} - 1, Q_{4,j-1} - 4),$$

In this way these comparisons for one row can be made in order N operations. In general if GapLenWgt is the gap length weight, and GapWgt is the gap weight, the incremental comparison can be written as

$$BestAcross_{i,j} = Max(BestAcross_{i-1} - GapLenWgt, Q_{i-2,j-1} - GapLenWgt - GapWgt)$$

Or, more concisely

$$BestAcross_{i,j} = Max(BestAcross_{i-1,j}, Q_{i-2,j-1} - GapWgt) - GapLenWgt$$

A similar technique may be used to calculate BestDown, using the result calculated on the previous row which is maintained in an array. Now the comparisons needed to obtain each alignment quality are simplified. One comparison to determine BestAcross, then one for BestDown. Finally we compare BestAcross, BestDown, and $Q_{i-1,j-1}$ to get $Q_{i,j}$. Since the complexity of the comparison for each of the N^2 alignment qualities is not dependent on N, The optimal alignment may be found in time proportional to N^2 not N^3.

This algorithm has been used in a program to optimally align DNA sequences in order N^2 time. The program running on a DEC Vax 11/780 computer under the VMS operating system can perform the Needleman-Wunsch procedure on two sequences of 1000 bases in under 2 minutes - just time enough to sip a little coffee. If this same problem was done in the classical way you could expect to get a good night's sleep before seeing the result.

This work was supported in part by a grant to Oliver Smithies from the National Institutes of Health (GM 20069). I would like to specially thank Oliver Smithies and John Devereux for introducing this problem and creating an environment in which this kind of understanding could take place. I am also grateful to Walter Fitch, Ron Niece, Temple Smith, and Mike Waterman for their helpful comments and encouragement.

References

Dolittle, R. F. Similar Amino acid Sequences: Chance or Common Ancestry? Science, Vol. 214, Oct 9, (1981)

Elleman, T. C. A Method for Detecting Distant Evolutionary Relationships Between Protein or Nucleic Acid Sequences in the Presence of Deletions or Insertions. Journal of Molecular Evolution Springer-Verlag (1978)

Needleman, S. B., Wunsch, C. D. A General Method Applicable to the search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology 48, 443-453, (1970)

Sankoff, D. Matching Sequences under Deletion/Insertion Constraints. Proceedings of The National Academy of Sciences Vol 69, No.1, pp. 4-6, Jan. (1972)

Waterman, M. S., Smith, T. F., Beyer, W. A. Some Biological Sequence Metrics. Advances in Mathematics 20, 367-387 (1976)

Appendix I

Implementation

In actually implementing this algorithm the storage space required can be greatly reduced by recognizing that the entire M and Q matrices need not be maintained. $M_{i,j}$ can be determined directly from the two sequences, while $Q_{i,j}$ values must be maintained only for the current row being processed. The only large data structure is the trace back path array $P_{i,j}$. If enough physical memory is not available, disk storage can be used efficiently. Please examine the fortran implementation which follows.

ALIGN arguments:

s1	Is a character array representing strand 1.
s1len	Is an integer indicating the length of strand 1.
s2	Is a character array representing strand 2.
s2len	Is an integer indicating the length of strand 2.
WgtGap	Is a real number which gives the penalty for each inserted gap.
WgtGapLen	Is a real number which provides a penalty for the length of each gap.
Ngaps	Is an integer which returns the number of gaps inserted.
TotalMatch	Is a real number which returns the percent of match between the two strands.

LOCAL Variables:

XskipCol	Is the X coorinate, or column of the best X skip.
XskipMatch	Is the number of matches in the best X skip for the current row.
YskipRow(n)	Is the Y coordinate, or row of the best Y skip.
YskipMatch(n)	Is the number of matches in the best Y skip for column n.
Diag(n)	Is the number of matches for a diagonal step from from the current row for column n.
Trace(col, row)	Is the traceback array. This is the trace encoding:

```
If Trace( col, row ) > 0 Then ! Down the column
  col' = Trace( col, row )
  row' = row - 1
End If

If Trace( col, row ) = 0 Then ! Diagonal step
  col' = col - 1
  row' = row - 1
End If

If Trace( col, row ) < 0 Then ! Across the row
  col' = col - 1
  row' = -Trace( col, row )
End If
```

```
C **  Align  ********************************************************************
C **
C **    Written 20-Aug-1981  By  paul haeberli
C **
C **                         For Oliver Smithies and John Devereux
C **
C **                         At The University of Wisconsin - Madison
C **
C **
C **    This produces the optimal alignment of two sequences up to
C **    1000 bases in length.
C **
C ********************************************************************************
C
      Subroutine Align( s1, s1len, s2, s2len, wgtgap, wgtgaplen,
     *                                         Ngaps, totalmatch )
C
      Character s1(2000),   s2(2000)
      Integer   s1len,   s2len
      Real wgtgap, wgtgaplen, totalmatch
      Integer Ngaps
      Integer Xmax, Ymax, ocol, orow, col, row, pos
      Integer TrStep
      Integer*2 Trace(1000,1000)
      Integer XskipCol, YskipRow(1000)
      Real XskipMatch, YskipMatch(1000), Diag(1000), MatSum(1000)
      Real Xskip, Yskip
      Integer BaseMatch
C
C ****  Check Sequence length
C
      If ((s1len.gt.1000).or.(s2len.gt.1000)) Then
        Type *,'** ERROR ** Align: length must be less than 1000'
        Stop
      End If
C
C ****  Do the bottom row
C
      Xmax = s1len
      Ymax = s2len
      Do col = 1, Xmax
        YskipRow(col) = 1.0
        YskipMatch(col) = BaseMatch(s1(col),s2(1))
        Diag(col) = YskipMatch(col)
      End Do
```

```
C
C ****  Create the trace array.
C
        Do row = 2, Ymax
          XskipCol = 1.0
          XskipMatch = BaseMatch(s1(1),s2(row))
C
          MatSum(1)=XskipMatch
          Do col = 2, Xmax
            Xskip = XskipMatch - WgtGap - WgtGapLen
            Yskip = YskipMatch(col-1) - WgtGap - WgtGapLen
C
            If (Xskip.gt.Diag(col-1)) Then
              If (Xskip.gt.Yskip) Then
                Trace(col,row) = XskipCol                ! Do an X skip
                MatSum(col) = Xskip + BaseMatch(s1(col),s2(row))
             Else
                Trace(col,row) = -YskipRow(col-1)        ! Do a Y skip
                MatSum(col) = Yskip + BaseMatch(s1(col),s2(row))
              End If
            Else
              If (Yskip.gt.Diag(col-1)) Then
                Trace(col,row) = -YskipRow(col-1)        ! Do a Y skip
                MatSum(col) = Yskip + BaseMatch(s1(col),s2(row))
              Else
                Trace(col,row)=0                         ! Do a diagonal step
                MatSum(col) = Diag(col-1) + BaseMatch(s1(col),s2(row))
              End If
            End If
C
C ****  Incrementally update XskipMatch and XskipCol
C
            XskipMatch = XskipMatch - WgtGapLen
            If (Diag(col).ge.Xskip+wgtgap) Then
              XskipCol = col
              XskipMatch = Diag(col)
            End If
          End Do
C
C ****  Now Incrementally update the YskipMatch and YskipRow arrays.
C
          Do col = 1, Xmax
            YskipMatch(col) = YskipMatch(col) - WgtGapLen
            If (Diag(col).ge.YskipMatch(col)) Then
              YskipRow(col) = row-1
              YskipMatch(col) = Diag(col)
            End If
            Diag(col) = MatSum(col)
          End Do
        End Do
```

```
C
C ****  Determine trace starting point and start building the output string.
C
      Ngaps = 0        ! count gaps
      XskipCol=1.0
      XskipMatch=Diag(1)
      Do col = 2, Xmax                    ! find max across top.
        XskipMatch = XskipMatch - WgtGapLen
        If (Diag(col).ge.XskipMatch) Then
          XskipCol=col
          XskipMatch=Diag(col)
        End If
      End Do
      Xskip = XskipMatch
      Yskip = YskipMatch(Xmax)
C
      If (Xskip.gt.Diag(Xmax)) Then
        If (Xskip.gt.Yskip) Then
          col = XskipCol                          ! Do an X skip
          row = Ymax
          TotalMatch = Xskip
        Else
          col = Xmax                              ! Do a Y skip
          row = YskipRow(Xmax)
          TotalMatch = Yskip
        End If
      Else
        If (Yskip.gt.Diag(Xmax)) Then
          col = Xmax                              ! Do a Y skip
          row = YskipRow(Xmax)
          TotalMatch = Yskip
        Else
          col = Xmax                              ! Do a diagonal step
          row = Ymax
          TotalMatch = Diag(Xmax)
        End If
      End If
      pos = 2000
      ocol = Xmax + 1
      orow = Ymax + 1
      Call GapGen(s1,s2,pos,ocol,orow,col,row)
```

```
C
C ****  Trace out the best path from this starting point
C
        Do While((col.ne.1).and.(row.ne.1))
          TrStep=Trace(col,row)
          If (TrStep.eq.0) Then
            col = col - 1                          ! Follow diagonal step
            row = row - 1
          Else
            If (TrStep.gt.0) Then
              col = TrStep                         ! Follow X skip
              row = row - 1
              Ngaps = Ngaps + 1
            Else
              col = col - 1                        ! Follow Y skip
              row = -TrStep
              Ngaps = Ngaps + 1
            End If
          End If
          Call GapGen(s1,s2,pos,ocol,orow,col,row)
        End Do
        col = 0
        row = 0
        Call GapGen(s1,s2,pos,ocol,orow,col,row)
        pos = pos + 2                              ! that ol' fence post.
C
C ****  Shift gapped bases to the start of s1 and s2
C
        Do i = pos, 2000
          s1(i-pos+1) = s1(i)
          s2(i-pos+1) = s2(i)
        End Do
        s1Len = 2000 - pos + 1
        s2Len = s1Len
C
        TotalMatch = 100 * TotalMatch / s1len
        Return
        End
```

```
C **  GapGen  ***************************************************************
C **
C **    This adds bases and gaps (if needed) to strands for one step in the
C **    tracing of the path.
C **
C ********************************************************************************
C
      Subroutine GapGen( s1, s2, pos, ocol, orow, col, row )
C
      Character s1(*), s2(*)
      Integer pos, ocol, orow, col, row
C
      If (ocol-col.ne.1) Then
        Do i = ocol-1, col+1, -1              ! Make gap for X step
          s1(pos) = s1(i)
          s2(pos) = '.'
          pos = pos - 1
        End Do
      Else
        If (orow-row.ne.1) Then
          Do i = orow-1, row+1, -1            ! Make gap for Y step
            s1(pos) = '.'
            s2(pos) = s2(i)
            pos = pos - 1
          End Do
        End If
      End If
      s1(pos) = s1(col)
      s2(pos) = s2(row)
      pos = pos - 1
      ocol = col
      orow = row
      Return
      End

C **  BaseMatch  ************************************************************
C **
C **   This compares two bases, returning 1 if they are identical, 0 otherwise.
C **
C ********************************************************************************
C
      Function BaseMatch( Base1, Base2 )
C
      Character Base1, Base2
      Integer BaseMatch
C
      If (Base1.eq.Base2) Then
        BaseMatch = 1
      Else
        BaseMatch = 0
      End If
      return
      end
```