Algorithms for pairwise alignment of biological sequences

Peter Sestoft
sestoft@dina.kvl.dk

Department of Mathematics and Physics, KVL

2000-04-05

Literature:

Durbin et al: Biological Sequence Analysis, Cambridge University Press 1998, chapter 2


Algorithms for pairwise alignment

Consider two amino acid sequences HEAGAWGHEE and PAWHEAE. Call them X and Y, and denote their lengths by m and n.

We discuss two alignment problems:

Global alignment: all of X must be aligned with all of Y (Needleman-Wunsch):

HEAGAWGHEE -- PAWHEAE

Local alignment: a subsequence of X must be aligned with a subsequence of Y (Smith-Waterman):

AWGHE -- AWHE

As we can see, an alignment may contain gaps. We consider two kinds of gap costs:

Simple linear gap costs: a gap of length h has score h.

Affine gap costs: a gap of length h has score h.

Warming up: Similarity of two strings; substitution matrices

Consider two strings, e.g. RLKAE and KNKGE of the same length.

In an ungapped alignment, an amino acid in X must be matched by an amino acid in Y.

The score of a match between two amino acids is given by a substitution matrix.

We can see an improvement in our current gap. We consider two kinds of gap costs:

Simple linear gap costs: a gap of length h has score h.

Affine gap costs: a gap of length h has score h.

Various substitution matrices are used in alignment algorithms, e.g. BLOSUM and the PAM matrices.

The similarity of the strings X and Y is given by the sum of the scores:

\[ S(X, Y) = \sum_{i=1}^{m} \sum_{j=1}^{n} \begin{cases} s(i,j) & \text{if } X_i = Y_j \\ -g & \text{if } X_i \neq Y_j \\ 0 & \text{if } X_i \text{ or } Y_j \text{ is a gap} \end{cases} \]

where s(i,j) is the score of the substitution matrix and g is the gap score.

The gap length is limited to some number, typically 5 or 10.

A gap score matrix allows for the occurrence of gaps, which is reflected in the substitution matrix.

We will discuss several alignment algorithms.

The aim is to find the alignment with the highest score.

As an example, consider the amino acid sequences:

| A | N | D | C | O | G | H | I | L | M | F | S | T | W | Y | V |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0 | -1 | -2 | -1 | -2 | 0 | -1 | -2 | -1 | -2 | 0 | -1 | -2 | 0 | -1 | -2 | 0 | -1 | -2 | 0 |
| N | -1 | 0 | -3 | -2 | -3 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| D | -2 | -3 | 0 | -1 | -3 | 2 | -2 | -2 | -1 | -2 | 2 | -2 | -2 | 2 | -2 | -2 | 2 | -2 | -2 | 2 |
| C | -1 | -2 | -1 | 0 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| Q | -2 | -3 | 2 | -1 | 0 | -2 | 1 | -2 | -1 | -2 | 1 | -2 | -2 | 1 | -2 | -2 | 1 | -2 | -2 | 1 |
| E | -1 | -2 | -2 | 1 | -2 | 0 | -1 | -2 | -1 | -2 | 0 | -1 | -2 | 0 | -1 | -2 | 0 | -1 | -2 | 0 |
| G | 0 | -1 | -2 | 1 | -2 | 0 | -1 | -2 | -1 | -2 | 0 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| H | -1 | -3 | -1 | 2 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| I | -1 | -2 | -2 | 1 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| L | -1 | -2 | -1 | 2 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| K | -1 | -2 | -2 | 1 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| M | -2 | -3 | -1 | 2 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| F | -1 | -2 | -1 | 0 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| P | -1 | -2 | -1 | 0 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| S | -2 | -3 | -1 | 2 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| T | -1 | -2 | -1 | 1 | -2 | 0 | -1 | -2 | -1 | -2 | 0 | -1 | -2 | 0 | -1 | -2 | 0 | -1 | -2 | 0 |
| W | -2 | -3 | -1 | 2 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| Y | -1 | -2 | -1 | 0 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |
| V | -2 | -3 | -1 | 2 | -2 | 1 | -1 | -2 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 | -1 | -2 | 1 |

Dina Research School Workshop April 2000

Page 4
In a global alignment of a protein X and a protein Y, we want to find an optimal alignment between them. The score of a match in a global alignment is given by:

\[ \text{score of a match} = \sum_{i=1}^{m} \sum_{j=1}^{n} \text{match score}(X_i, Y_j) \]

where \( \text{match score}(X_i, Y_j) \) is the score of matching amino acid \( X_i \) and amino acid \( Y_j \).

To find the optimal alignment, we can build a table in which we compute the scores for all possible alignments. The value in each cell is the maximum score that can be achieved by aligning the prefixes of the two proteins up to that cell.

**Algorithm:**

1. Initialize a matrix \( M \) of size \( (m+1) \times (n+1) \), where \( m \) and \( n \) are the lengths of proteins X and Y, respectively.
2. Set the first row and column of \( M \) to -1.
3. For each cell \( (i, j) \) in \( M \), compute the score:
   - If \( X_i = Y_j \), then \( M_{i,j} = M_{i-1,j-1} + \text{identity score} \)
   - If \( X_i \neq Y_j \), then \( M_{i,j} = \max(M_{i-1,j} + \text{gap penalty}, M_{i,j-1} + \text{gap penalty}) \)
4. The optimal alignment is the path from the bottom-right corner of \( M \) back to the top-left corner, with the traceback points indicating the optimal alignment.

**Traceback:**

1. Start at the bottom-right cell of \( M \) with the maximum score.
2. If the current cell \( (i, j) \) matches the previous cell \( (i-1, j-1) \), go left-up or left-right.
3. If the current cell \( (i, j) \) does not match the previous cell, go left-up or left-right based on the cell with the maximum score.
4. The traceback points at the cell that led to the maximal score.

**Global Alignment Example:**

Consider the two proteins:

- Protein X: HEAGAWGHEE
- Protein Y: FGEKFD

Using the BLOSUM50 matrix, compute the scores for all possible alignments and find the optimal alignment.

**Observation:**

- The optimal alignment between the two proteins is:
  
  HEAGAWGHEE
  |
  FGEKFD

  The score of this alignment is:

  \[ \sum_{i=1}^{11} \text{match score}(X_i, Y_i) \]

**Tree:**

```
A
  |
  G
  |
  W
  |
  H
  |
  E
  |
  F
  |
  G
  |
  E
  |
  K
  |
  D
```

**Tree:**

```
A
  |
  G
  |
  W
  |
  H
  |
  E
  |
  F
  |
  G
  |
  E
  |
  K
  |
  D
```

**Conclusion:**

- In a global alignment of proteins X and Y, we construct a table of scores for all possible alignments.
- The optimal alignment is found by tracing back from the cell with the maximum score.
- The score of the optimal alignment is the sum of the scores of the matches between the proteins.

**Useful observation:**

- The alignment is computed recursively, with the score at each cell depending on the scores of the previous cells.
- The traceback points indicate the optimal alignment between the two proteins.
The filled-in matrix for global alignment of \( \text{HEAGAWGHEE} \) and \( \text{PAWHEAE} \). The traceback is recorded in a matrix with the same shape as \( F \) for \( i \), \( j \):

\[
F[i][j] = \begin{cases} \text{score of } i \text{th and } j \text{th positions} & \text{if match} \\ \text{max}(-d, F[i-1][j-1] + \text{score of } i \text{th and } j \text{th positions}) & \text{if mismatch} \end{cases}
\]

The traceback pointer at \( F[i][j] \) points to \( F[i-1][j-1] \) if there is a match, \( F[i-1][j] \) if there is a gap in the first sequence, or \( F[i][j-1] \) if there is a gap in the second sequence.

Hence we initialize the borders as follows:

```java
for(int i=1;i<=n;i++)
    F[i][0]=-d*i;
for(int j=1;j<=m;j++)
    F[0][j]=-d*j;
```

Implementing global alignment: Initialization

Upper border: position \( i \) represents the alignment of \( \text{HEAGAWGHEE} \) to the empty prefix of \( \text{PAWHEAE} \). That is, the prefix \( \text{HEAGAWGHEE} \) has been matched with gaps in \( \text{PAWHEAE} \). With simple linear gap costs, the score is \( -d \times \text{number of gaps} \). The left-hand border is similar.

Hence we initialize the borders as follows:

```java
for(int i=1;i<=n;i++)
    F[i][0]=-d*i;
for(int j=1;j<=m;j++)
    F[0][j]=-d*j;
```

### Implementing global alignment: Filling in the matrix

Position \( (i,j) \) may be reached from \( (i-1,j-1) \) with a match, \( (i-1,j) \) with a gap in the first sequence, or \( (i,j-1) \) with a gap in the second sequence.

The traceback matrix is recorded in a matrix \( B \) with the same shape as \( F \) for \( i \), \( j \):

```java
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        B[i][j]=new Traceback2(i-1,j-1);
```

The start of the traceback is cell \( (n,m) \).

Local alignment of \( \text{HEAGAWGHEE} \) and \( \text{PAWHEAE} \) (Smith-Waterman 1981)

A subsequence of \( \text{HEAGAWGHEE} \) must be aligned with a subsequence of \( \text{PAWHEAE} \):

\[ \text{AWGHE} \]

Requirement: the expected score of a random match must be negative. If the score of a random match were positive, then many local alignments could be profitably extended to longer subsequences.

New interpretation of \( F \):

The maximal score for an alignment between a suffix of \( \text{HEAGAWGHEE} \) and a suffix of \( \text{PAWHEAE} \) is

\[
F[n][m]
\]

Local alignment of \( \text{HEAGAWGHEE} \) and \( \text{PAWHEAE} \) (Smith-Waterman 1981)
Implementing local alignment: Initialization

The upper border represents the alignment of a suffix of $\text{seq1}$ to an empty sequence. An empty match, with score 0, is the best we can do (because gapshave negative scores). Then it is the start of a new local alignment, and the traceback pointer is somewhere.

Hence we initialize the border cells to 0 and the traceback to null (this is the default value in Java).

Reducing the space consumption of local alignment

Reducing the space consumption of global alignment

Implementing local alignment: Filling in the matrix

Implementing local alignment: Initialization

The start of the traceback must be at some cell $f(i,j)$ in $F$ with maximal score.

$$f(i,j) = \max\{0, F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d\}$$

The start $B_0$ of the traceback must be set to some cell in $F$ with maximal score.

How reconstruct the optimal global alignment in this case?

When $i < j$ or $i > j$, use the standard algorithm in the case (as these spaces already have a maximal score).

The traceback is the source of the maximal resulting score. Thus:

$$f(i,j) = \max\{0, F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d\}$$

For (int $i=1$; $i<=n$; $i++)$

$$f(i-1, j-1) = \max\{0, F[i-1][j-1]+s, F[i-1][j]-d, F[i][j-1]-d\}$$

$$f(i-1, j) = \max\{0, F[i-1][j]-d, F[i][j-1]-d\}$$

$$f(i, j-1) = \max\{0, F[i-1][j]-d, F[i][j-1]-d\}$$

$$f(i, j) = \max\{0, F[i-1][j]+s, F[i-1][j]-d, F[i][j-1]-d\}$$

For (int $i=1$; $i<=n$; $i++)$

$B_0[i,j] = \max\{0, F[i-1][j]+s, F[i-1][j]-d, F[i][j-1]-d\}$

In the above, $\text{seq1}$ and $\text{seq2}$ are in the order they appear in the input to the algorithm, and $f(i,j)$ is in the order they appear in the output.

Implementing local alignment: Filling in the matrix

Implementing local alignment: Initialization
Affine gap costs

Until now we used linear gap costs

$$H \rightarrow H$$, where

" \rightarrow ".

Thus a gap of length 7 has 4 times the cost of a gap of length 1.

This is unrealistic; too expensive, biologically speaking. A gap arises by an evolutionary event, and along a gap is nearly as likely to arise as a short one.

Better use affine gap costs of the form $$H \rightarrow H$$, where

$$6 \rightarrow 6$$ and $$6 \rightarrow 6$$.

Hence it is expensive to open a gap ($$6 \rightarrow 6$$) but inexpensive to extend it ($$6 \rightarrow 6$$).

Alignment with affine gap costs is done by dynamic programming using three matrices instead of one.

The matrices have the following meanings:

- The matrix in dynamic programming

When the strings and are identical, the traceback will follow the diagonal of the matrix.

In that case, only the diagonal need to be filled in (which takes much less time).

When the strings and are very similar, the traceback will follow a band along the diagonal.

In that case, only the elements of that band of the matrix need to be filled in.

We may speed up dynamic programming by filling in only a band along the diagonal of the matrix. But this may overlook a good (high-scoring) alignment whose traceback would go outside the band.

Databases

A sequenced database contains a large number of sequences (e.g. 100,000).

When searching a database we give a short query string, e.g. of length 100.

We then seek the best local, gapped alignment between the query string and each of the database strings.

We might use the Smith-Waterman algorithm for each sequence in the database.

This is too slow in practice.

Databases search programs (Blast and Fasta) do dynamic programming, but only after some preliminary work.

The Blast2 database search algorithm (Altschul et al. 1997)

Leta query string be given.

Find hits between 3-letter substrings of the query string and 3-letter substrings of the database strings.

A hit must have a score of at least I, e.g. I.

Find two non-overlapping hits on the same diagonal; they are close to each other (determined by J).

Let a query string be given.

The Blast2 database search algorithm (Altschul et al. 1997) has the following properties:

1. It is 100 times faster than Smith-Waterman on the computer and may be an order of magnitude faster.
2. It uses dynamic programming to do the alignment.
3. Its results are as sensitive and selective as Smith-Waterman.
4. It is a good method for finding biologically relevant alignments.