Algorithmen der Bioinformatik I
WS 2017/2018

Burkhard Morgenstern
Peter Meinicke

Dept. Bioinformatics
Institute of Microbiology and Genetics (IMG)
University of Göttingen

October 24, 2017

Optimal pairwise alignment

$F(i,j) =$ Score of optimal alignment of prefixes up to i und j

Recursion to calculate $F(i,j)$:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) & + & s(X_i, Y_j) \\ F(i-1,j) & - & g \\ F(i,j-1) & - & g \end{cases} \quad (1)$$

Optimal pairwise alignment

Initialize values:

- $F(0, j) = -j \times g$ (alignment only consists of j gaps in sequence X)
- $F(i, 0) = -i \times g$ (alignment only consists of i gaps in sequence Y)

Optimal pairwise alignment

To find best alignment: *Trace Back* !

For each (i, j), store which of the three alternatives in (1) is the optimum.

Complexity of algorithm:

- Time: $O(m \times n)$
- Space: $O(m \times n)$

Optimal pairwise alignment

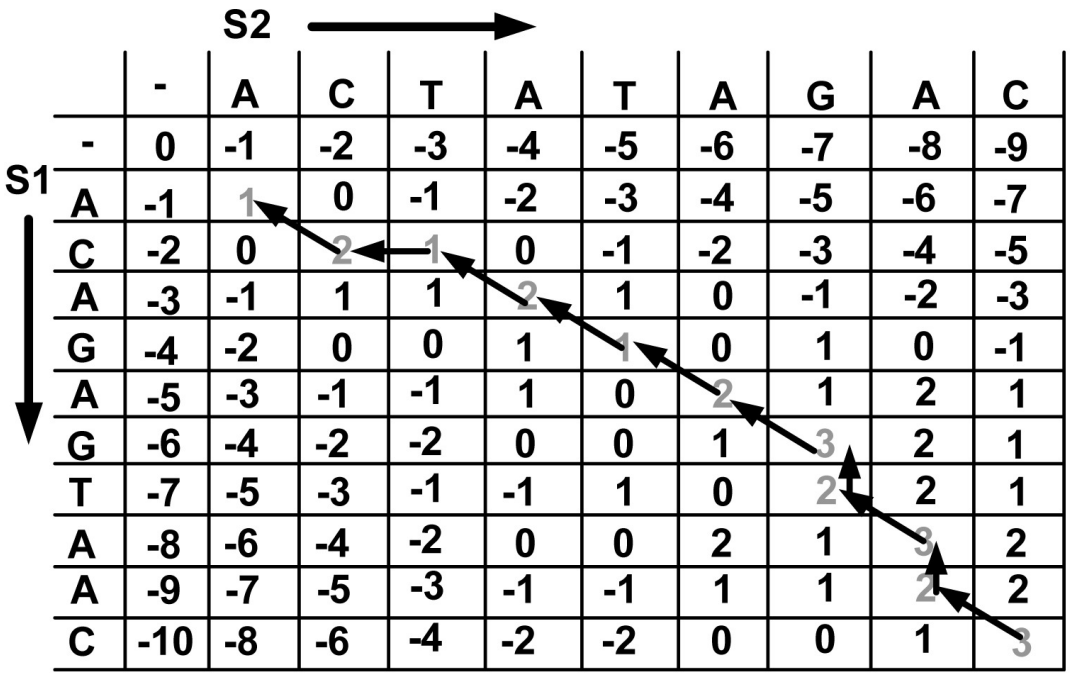


Figure: Alignment as *path* in *DP matrix* (IEEE Computer Society, `https://www.computer.org/`).

Local sequence alignment

So far: *global alignment*, i.e. alignment of full input sequences, but:

- Many sequences not *globally* related, only *local* homologies detectable.
- *Local* alignment searches one *segment* from each of the input sequences such that alignment score of for these segments is maximized.

Smith-Waterman algorithm finds optimal *local* alignment.

Local sequence alignment

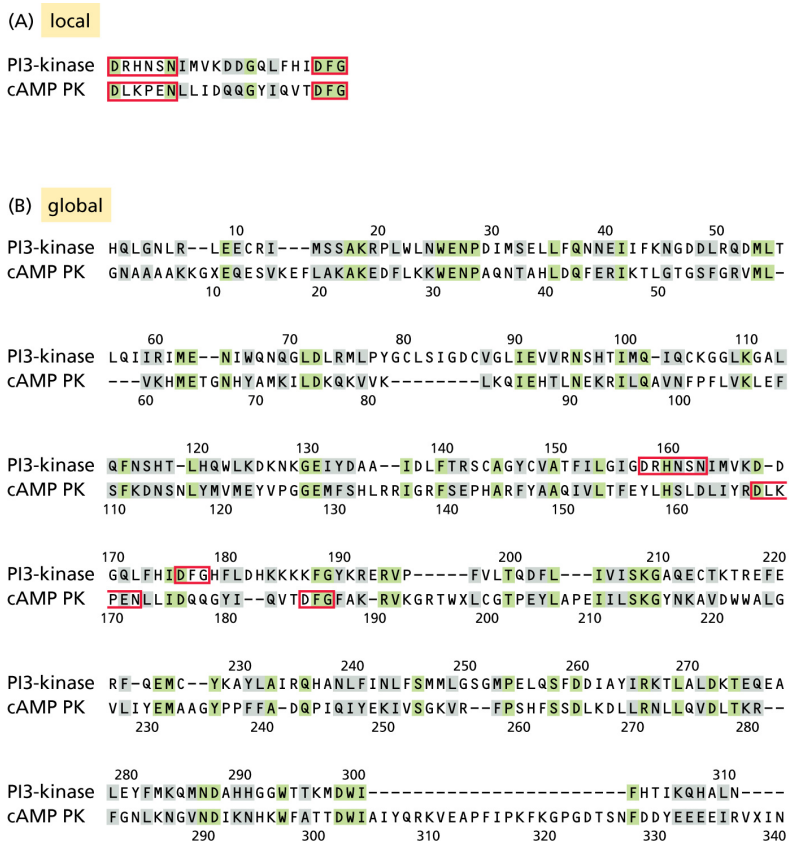


Figure: (A) Local and (B) global alignment of two kinase sequences. Locally conserved funktional region shown in red (Zvelebil & Baum, *Understanding Bioinformatics*

Local sequence alignment

Example (Local protein alignment)

```
seq1  A D E Q M S C V W M I P H K R
seq2  L M V L V I S C M I P A G S G
```

Locally related protein sequences

Local sequence alignment

Example (Local protein alignment)

```
seq1 A D E Q   M S C V W M I P H K R
seq2 L M V L V I S C - - M I P A G S G
```

Local alignment: only *segments* (red) aligned, non-related parts of sequences ignored.

Sounds more complicated . . .

. . . but: algorithm almost the same as for *global* alignment

Local sequence alignment

Now $F(i, j)$ score of best *local* alignment, with segments *ending* in i, j.

Recursion:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) & + & s(X_i, Y_j) \\ F(i-1, j) & - & g \\ F(i, j-1) & - & g \\ 0 \end{cases}$$

Initialize:

- $F(0, j) = 0$
- $F(i, 0) = 0$

Optimal alignment with linear memory

Observation: values $F(i, j)$ can be calculated by DP in

$$O(m \cdot n)$$

time and with

$$O(\max\{m, n\})$$

memory. Only values in previous colums (or previous line) necessary!

Optimal alignment with linear memory

I.e. *score* of an optimal (global) alignment of two sequences can be computed with *linear* memory complexity.

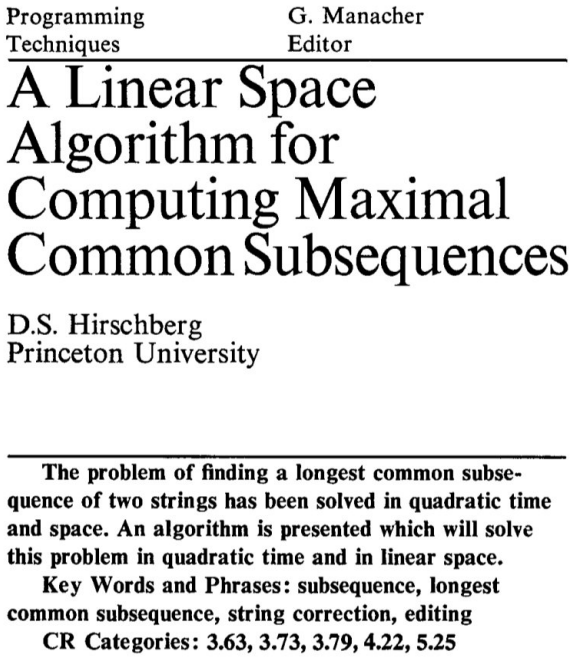
Space

$$O(m \cdot n)$$

only necessary for *trace back* to find optimal alignment.

Divide-and-conquer algorithm allows to compute optimal alignment in linear time!

Optimal alignment with linear memory



Programming Techniques — G. Manacher Editor

A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

Key Words and Phrases: subsequence, longest common subsequence, string correction, editing
CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25

Figure: D. Hirschberg, 1975: find *longest common subsequences* of two strings with *linear space* using divide-and-conquer

Optimal alignment with linear memory

Definition (Subsequence)

A subsequence Y of a sequence (string) X is obtained by omitting characters from X

Example (Subsequence)

Sequence

$$X = \text{BANANA}$$

Subsequence of X

$$Y = \text{ANNA}$$

Optimal alignment with linear memory

Definition (Subsequence)

A subsequence Y of a sequence (string) X is obtained by omitting characters from X

Example (Subsequence)

Sequence

$$X = BANANA$$

Subsequence of *X*

$$Y = ANNA$$

Optimal alignment with linear memory

Definition (Common subsequence)

A string Z is a common subsequence *of strings X and Y, if Z is a subsequence of X and a subsequence of Y*

Example (Common subsequence)

X = A C B D E G C E D B G

Y = B E G C F E U B K

Z = B E E

Optimal alignment with linear memory

Definition (Common subsequence)

A string Z is a common subsequence *of strings X and Y, if Z is a subsequence of X and a subsequence of Y*

Example (Common subsequence)

X = A C B D E G C E D B G

Y = B E G C F E U B K

Z = B E E

Optimal alignment with linear memory

Definition (Common subsequence)

A string Z is a common subsequence *of strings X and Y, if Z is a subsequence of X and a subsequence of Y*

Example (Longest common subsequence, LCS)

X = A C B D E G C E D B G

Y = B E G C F E U B K

Z' = B E G C E B

Optimal alignment with linear memory

Theorem

For strings X and Y of length m and n, respectively, the longest common subsequence (LCS) *of X and Y can be found in*

$$O(m \cdot n)$$

time

Proof: Apply algorithm for optimal pairwise alignment with suitable parameters for substitution scores $s(a, b)$ and gap penalty g.

Optimal alignment with linear memory

Define

$$
\begin{aligned}
s(a,b) &= \left\{ \begin{array}{ll} 1 & \text{if } a = b \\ -1 & \text{if } a \neq b \end{array} \right\} \\
g &= 0
\end{aligned}
$$

In this case:

- Optimal alignment will only align identical characters (‘matches’).
- Score of optimal alignment = number of ‘matches’