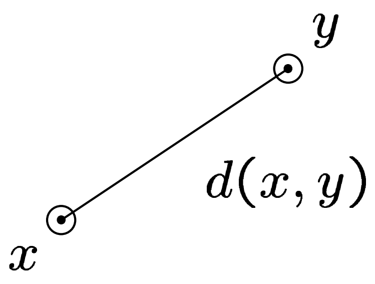
Today's program

$\frac{1}{2}$h Sequence comparison



Jyrki Katajainen

Department of Computing
University of Copenhagen

My main sources

William R. Pearson, Protein sequence comparison and protein evolution, *ISMB95 Tutorial* (1995)

João Setubal and João Meidanis, *Introduction to Computational Molecular Biology*, PWS Publishing Company (1997)

Mike Paterson and Vlado Dančík, Longest common subsequences, *Proceeding of the 19th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science* **841**, Springer-Verlag (1994), 127–142

James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy, Delta algorithms: an empirical analysis, *ACM Transactions on Software Engineering and Methodology* **7** (1998)

Symbols and sequences

In biological applications the symbols can be

- amino acids;

  a **protein** is a sequence of amino acids;
- bases in DNA molecules;

  each cell of an organism has a few very long DNA molecules; such a molecule is called a **cromosome**.

In a computer these symbols are often represented using one-letter codes.

Similarity of molecular sequences

Definitions: Let U and V be two sequences, and let - denote a space (not in U or V).

An **alignment** between U and V is defined as the insertion of spaces in arbitrary locations so that they end up with the same size. No space in U should be aligned with a space in V.

Let $\sigma\begin{pmatrix} x \\ y \end{pmatrix}$ denote the **score** between two symbols x and y. Given an alignment, its *score* is the sum of the scores of the corresponding symbols.

Problem: Compute the **similarity** between U and V, i.e., the score of the best alignment (maximum score).

Example: Assume that $\sigma\begin{pmatrix} x \\ x \end{pmatrix} = +1$, $\sigma\begin{pmatrix} x \\ y \end{pmatrix} = -1$, and $\sigma\begin{pmatrix} - \\ x \end{pmatrix} = -2$. Then the optimal alignment between `A B D D E F G H I` and `A B D E G K H I` has score 2.

```
A B D D E F G H I
A B D - E G K H I
```

Dynamic programming

Let $s[i,j]$ denote the similarity between the prefixes of $X[1{:}i]$ and $Y[1{:}j]$. That is, we are trying to compute $s[m,n]$.

A recursive definition for matrix s, exploiting an obvious optimal subproblem property, is given as follows.

$$s[i,j] \quad = \quad \max \left\{ \begin{array}{l} s[i-1,j-1]+\sigma\begin{pmatrix} X[i] \\ Y[j] \end{pmatrix}, \\ s[i-1,j]+\sigma\begin{pmatrix} X[i] \\ - \end{pmatrix}, \\ s[i,j-1]+\sigma\begin{pmatrix} - \\ Y[j] \end{pmatrix} \end{array} \right\}$$

Computing the similarity

The following algorithm fills the similarity matrix row by row.

```
Global-similarity($X$,$Y$) ▷Runtime: $O(mn)$
1   $m \leftarrow length(X)$
2   $n \leftarrow length(Y)$
3   $s[0,0] \leftarrow 0$
4   for $j \leftarrow 1$ to $n$
5       $s[0,j] \leftarrow s[0,j-1]+\sigma\begin{pmatrix} - \\ Y[j] \end{pmatrix}$
6   for $i \leftarrow 1$ to $m$
7       $s[i,0] \leftarrow s[i-1,0]+\sigma\begin{pmatrix} X[i] \\ - \end{pmatrix}$
8       for $j \leftarrow 1$ to $n$
9           $s[i,j] \leftarrow \max\left\{\begin{array}{l} s[i-1,j-1]+\sigma\begin{pmatrix} X[i] \\ Y[j] \end{pmatrix}, \\ s[i-1,j]+\sigma\begin{pmatrix} X[i] \\ - \end{pmatrix}, \\ s[i,j-1]+\sigma\begin{pmatrix} - \\ Y[j] \end{pmatrix} \end{array}\right\}$
10  return $s[m,n]$
```

Example: global alignment

	j	0	1	2	3	4	5	6	7	8	9
i		Y	A	B	D	D	E	F	G	H	I
0	X	0	← -2	← -4	← -6	← -8	← -10	← -12	← -14	← -16	← -18
1	A	↑ -2	↖ 1	← -1	← -3	← -5	← -7	← -9	← -11	← -13	← -15
2	B	↑ -4	↑ -1	↖ 2	← 0	← -2	← -4	← -6	← -8	← -10	← -12
3	D	↑ -6	↑ -3	↑ 0	↖ 3	← 1	← -1	← -3	← -5	← -7	← -9
4	E	↑ -8	↑ -5	↑ -2	↑ 1	↖ 2	↖ 2	← 0	← -2	← -4	← -6
5	G	↑ -10	↑ -7	↑ -4	↑ -1	↖ 0	↖ 1	↖ 1	↖ 1	← -1	← -3
6	K	↑ -12	↑ -9	↑ -6	↑ -3	↖ -2	↖ -1	↖ 0	↖ 0	↖ 0	← -2
7	H	↑ -14	↑ -11	↑ -8	↑ -5	↑ -4	↑ -3	↑ -2	↖ -1	↖ 1	↖ -1
8	I	↑ -16	↑ -13	↑ -10	↑ -7	↑ -6	↑ -5	↑ -4	↑ -3	↑ -1	↖ 2

Local similarity scores

A **local alignment** between X and Y is an alignment between a substring of X and a substring of Y.

```
Local-similarity($X$,$Y$) ▷Runtime: $O(mn)$
1  $m \leftarrow length(X)$
2  $n \leftarrow length(Y)$
3  $best \leftarrow 0$
4  for $j \leftarrow 1$ to $n$
5      $s[0,j] \leftarrow s[0,j-1] + \sigma\begin{pmatrix} - \\ Y[j] \end{pmatrix}$
6  for $i \leftarrow 1$ to $m$
7      $s[i,0] \leftarrow s[i-1,0] + \sigma\begin{pmatrix} X[i] \\ - \end{pmatrix}$
8      for $j \leftarrow 1$ to $n$
9          $s[i,j] \leftarrow \max\left\{\begin{array}{l} 0, \\ s[i-1,j-1] + \sigma\begin{pmatrix} X[i] \\ Y[j] \end{pmatrix}, \\ s[i-1,j] + \sigma\begin{pmatrix} X[i] \\ - \end{pmatrix}, \\ s[i,j-1] + \sigma\begin{pmatrix} - \\ Y[j] \end{pmatrix} \end{array}\right\}$
10         $best \leftarrow \max\{best, s[i,j]\}$
11 return $best$
```

Example: local alignment

	j	0	1	2	3	4	5	6	7	8	9
i		Y	A	B	D	D	E	F	G	H	I
0	X	0	0	0	0	0	0	0	0	0	0
1	A	0	↖1	0	0	0	0	0	0	0	0
2	B	0	0	↖2	$\overset{\leftarrow}{0}$	0	0	0	0	0	0
3	D	0	0	↑0	↖3	$\overset{\leftarrow}{1}$	0	0	0	0	0
4	E	0	0	0	↑1	↖2	↖2	$\overset{\leftarrow}{0}$	0	0	0
5	G	0	0	0	0	↖0	↖1	↖1	↖1	0	0
6	K	0	0	0	0	0	0	↖0	↖0	↖0	0
7	H	0	0	0	0	0	0	0	0	↖1	0
8	I	0	0	0	0	0	0	0	0	0	↖2

Optimal local alignment (score 3):

```
A B D
A B D
```

Possible improvements

The amount of space used can be reduced to $O(m+n)$.

This is left as a home exercise. (Hint: Use divide and conquer.)

Also, there exists an algorithm with time complexity $O(dn)$, where d is the difference between the maximum possible score and the optimal score. Thus, the higher the similarity, the faster the answer. Space-saving versions can also be derived.

Basic idea: compute only the similarities around the main diagonal; double the distance from the diagonal until one can be sure that the solution is optimal.

Scoring functions

- identity/non-identity function: $\sigma\begin{pmatrix}x\\x\end{pmatrix} = M$, $\sigma\begin{pmatrix}x\\y\end{pmatrix} = m$, and $\sigma\begin{pmatrix}-\\x\end{pmatrix} = g$ (usually $g < 0$, and naturally $m < M$).

  $$\sigma\begin{pmatrix}-\mathtt{A}\\\mathtt{C}-\end{pmatrix} < \sigma\begin{pmatrix}\mathtt{A}\\\mathtt{C}\end{pmatrix} \Rightarrow 2g < m$$

  $\sigma\begin{pmatrix}-\mathtt{AT}\\\mathtt{TA}-\end{pmatrix} < \sigma\begin{pmatrix}\mathtt{AT}\\\mathtt{TA}\end{pmatrix} \Rightarrow m$ closer to M than $2g$

  The values $M = 1$, $m = -1$, and $g = -2$ fulfil these restrictions.

- subadditive functions: $f(k_1+k_2+\cdots+k_n) \leq \sum_{i=1}^{k} f(k_i)$. The idea is that k spaces are more probable than k isolated spaces.

- For protein sequences, the use of PAMk matrices is commonplace. These take into account the relative replaceability in an evolutionary scenario.

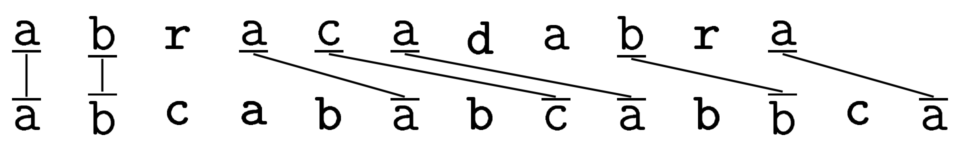
Longest common subsequence

Special case: $\sigma\begin{pmatrix} x \\ y \end{pmatrix} = 0$ and $\sigma\begin{pmatrix} x \\ x \end{pmatrix} = 1$.

Definitions: We say that $U = \langle u_1, u_2, \ldots, u_k \rangle$ is a **subsequence** of $X = \langle x_1, x_2, \ldots, x_m \rangle$ if there exist indices $1 \le i_1 < \cdots < i_k \le m$ such that $U = \left\langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \right\rangle$. U is a **common subsequence** of X and Y if U is a subsequence of both X and Y. We let $|U|$ denote the length of U.

Problem: Given two sequences X and Y, find their longest common subsequence.

Example: The LCS of the sequences



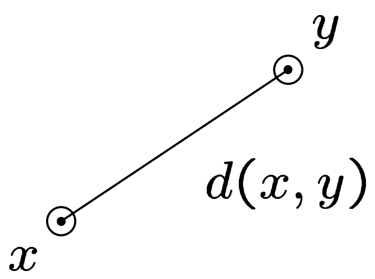
is `abcaaba` and is of length 7.

Related applications

- compression in revision control systems
- displaying of differences between files
- merging of the changes in two different files relative to a common base
- saving space when taking backups
- taking checkpoints in database systems
- updating terminal display over a slow network
- distributing updates for software and other data over the web
- speech processing
- spell checking
- improving I/O performance when processing data

Conclusion

The distance between our teaching and real world is a bit larger than ε, but not much larger.



Jyrki Katajainen

Department of Computing
University of Copenhagen