Local similarity

- Up to this point we have focused on global comparison.
- That is, we have compared the complete sequence u with the complete sequence v.
- In DNA sequences we often have long non-coding regions and small coding regions.
- Thus if two coding regions in two large sequences are similar, this does not imply that the sequences have a small edit distance, see the Example 1.
- In an analogy, protein sequences often contain several domains which they share with related sequences.
- But if only the domains are similar, then a global comparison would not reveal the similarity of the domains.

Example 1

The following global alignment shows two sequences with similarities over the entire range of positions. Hence a global alignment is an appropriate means to represent the similarities.

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-AC
  |  || |  ||  | | | |||    || |  | |  |||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--TCAGAT--C
```

The middle line uses the symbol | to mark columns of identical characters. Sometimes only segments of the sequences show similarities.

```
CGGGGAGGCGCCCGCGCGGCCCAGTTAT-TCAGAATTAAAATCTTAGTAAACAT
                          ||||| | ||||
AATTAAAATCTTAGTAAACATCAGTT-TGTCAGCGGGGAGGCGCCCGCGCGGCC
```

Thus a global alignment does not make sense in this case and one should look for local similarities, displayed in a local alignment.

Local similarity

- As a consequence, when comparing biological sequences it is sometimes important to perform local similarity comparisons: This means to find pairs of similar *substrings* of u and v.
- It does not make sense to look for pairs of substrings with some minimum distance, as ε is a substring of any sequence and the sequences ε and ε have distance 0.
- To clarify the notion of sequence similarity, we introduce score functions.

Definition 1

A *score function* σ assigns to each edit operation $\alpha \to \beta$ a *score* $\sigma(\alpha \to \beta) \in \mathbb{R}$. For each alignment $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ we define the score $\sigma(A) = \sum_{i=1}^{h} \sigma(\alpha_i \to \beta_i)$. The similarity score of the sequences u' and v' is defined by

$$score_\sigma(u', v') = \max\{\sigma(A) \mid A \text{ is an alignment of } u' \text{ and } v'\}. \quad \square$$

Table 1: The BLOSUM62 similarity score matrix specifying a replacement score for each pair of amino acid. As the matrix is symmetric only the lower halve of the matrix is shown. With some additional scores for insertions and deletions we would obtain a score function.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4																			
R	−1	5																		
N	−2	0	6																	
D	−2	−2	1	6																
C	0	−3	−3	−3	9															
Q	−1	1	0	0	−3	5														
E	−1	0	0	2	−4	2	5													
G	0	−2	0	−1	−3	−2	−2	6												
H	−2	0	1	−1	−3	0	0	−2	8											
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	4										
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	4									
K	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	5								
M	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	5							
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6						
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7					
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4				
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	5			
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	11		
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	7	
V	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4

Local similarity

Definition 2

Let σ be a score function. We define

1. $loc_\sigma(u, v) = \max\{score_\sigma(u', v') \mid u' \text{ is substring of } u \text{ and } v' \text{ is substring of } v\}$

   That is, $loc_\sigma(u, v)$ is the maximum score over all pairs of substrings of u and v.

2. Let u' be a substring of u and v' be a substring of v such that

$$score_\sigma(u', v') = loc_\sigma(u, v)$$

   An alignment A of u' and v' satisfying $score_\sigma(u', v') = \sigma(A)$ is an *optimal local alignment* of u and v.

3. The *local alignment problem* is to compute $loc_\sigma(u, v)$ and an optimal local alignment of u and v. □

Example 2

Consider the sequences $u = \texttt{PQRAFADCSTVQ}$ and $v = \texttt{FYAFDACSLL}$ and the following similarity score function:

$$\sigma(\alpha \to \beta) = \begin{cases} -1 & \text{if } \alpha = \varepsilon \text{ or } \beta = \varepsilon \\ -2 & \text{if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha \neq \beta \\ +2 & \text{if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha = \beta \end{cases}$$

Then $loc_\sigma(u, v) = 8 = score_\sigma(u', v')$, where $u' = \texttt{AFADCS}$, $v' = \texttt{AFDACS}$ and

```
A  F  -  A  D  C  S
A  F  D  A  -  C  S
2  2 -1  2 -1  2  2
```

of score 8 is an optimal local alignment (with the score of each column in the last line).

Local similarity

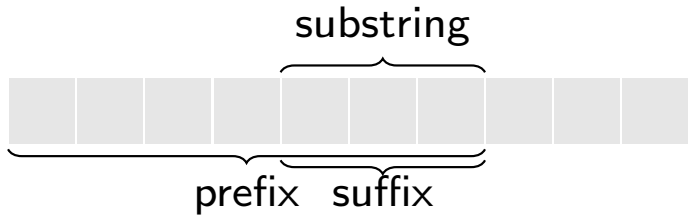
A brute force solution to the local alignment problem would be as follows:

for each pair (u', v') *of substrings* u' *of* u *and* v' *of* v *compute* $score_\sigma(u', v')$ *and determine the maximum over all such values*

- Computing $score_\sigma(u', v')$ is a global alignment problem where, instead of minimizing distances, similarities are maximized.
- Therefore, we can modify the DP Algorithm for computing the edit distance in such a way that a score function is used and maxima are computed instead of minima.
- Thus $score_\sigma(u', v')$ can be computed in $O(|u'||v'|) = O(mn)$ time.
- There are $O(m^2n^2)$ pairs (u', v') of substrings of u and v.
- Thus, this method would require $O(m^2n^2mn) = O(m^3n^3)$ time, which is, of course, too much.

Local similarity

– Note that a substring of a string is a suffix of a prefix of that string:

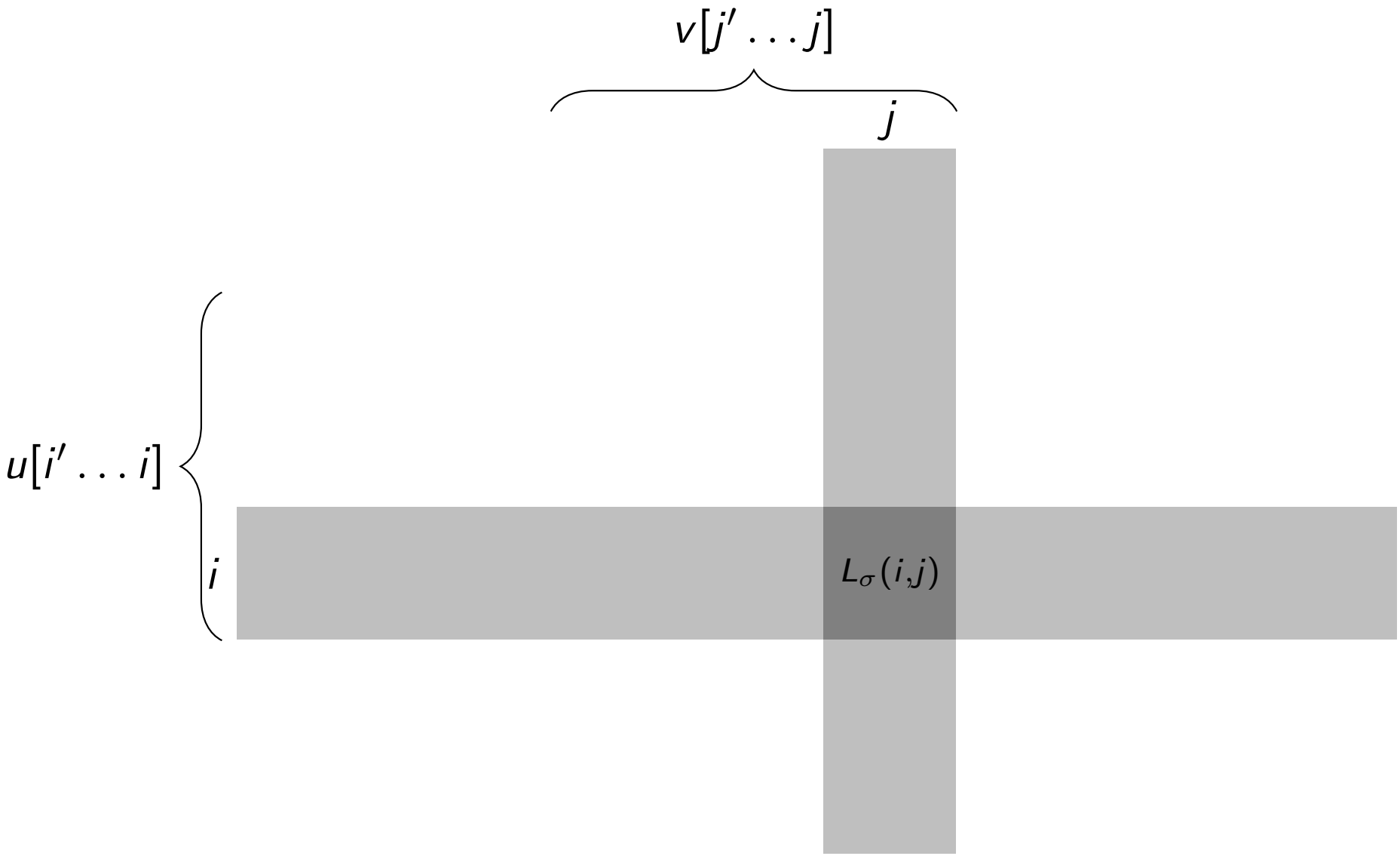


– So substring u' of u is a suffix of a prefix of u and substring v' of v is a suffix of a prefix of v.
– For example, if $u' = u[p \ldots q]$ (i.e. u' is a substring of u), then $u[1 \ldots q]$ is a prefix of u and u' is a suffix of this prefix.
– The idea is to compute a matrix where each entry (i, j) contains the maximal score for all pairs of suffixes of prefix $u[1 \ldots i]$ of u and prefix $v[1 \ldots j]$ of v.
– That is, we compute an $(m+1) \times (n+1)$-matrix L_σ defined by

$$L_\sigma(i,j) = \max\{score_\sigma(x,y) \mid x \text{ is suffix of } u[1 \ldots i] \text{ and } y \text{ is suffix of } v[1 \ldots j]\}$$

See also Figure 1 for an illustration.

Figure 1: The curly brackets mark the substrings of u and v an entry in L_σ refers to. In particular, $L_\sigma(i, j)$ refers to pairs of suffixes $u[i' \ldots i]$ of $u[1 \ldots i]$ and suffixes $v[j' \ldots j]$ of $v[1 \ldots j]$.



Local similarity

$$
\begin{aligned}
loc_\sigma(u,v) &= \max\{score_\sigma(u',v') \mid u' \text{ is substring of } u,\ v' \text{ is substring of } v\} \\
&= \max\{score_\sigma(x,y) \mid 0 \le i \le m, 0 \le j \le n, \\
&\qquad\qquad x \text{ is suffix of } u[1 \ldots i], \\
&\qquad\qquad y \text{ is suffix of } v[1 \ldots j]\} \\
&= \max\{\max\{score_\sigma(x,y) \mid x \text{ is suffix of } u[1 \ldots i], \\
&\qquad\qquad y \text{ is suffix of } v[1 \ldots j]\} \mid 0 \le i \le m, \\
&\qquad\qquad 0 \le j \le n\} \\
&= \max\{L_\sigma(i,j) \mid 0 \le i \le m, 0 \le j \le n\}
\end{aligned}
$$

- So $loc_\sigma(u,v)$ can be computed by maximizing over all entries in L_σ.
- Consider an edit graph representing all local alignments.
- Since we are interested in alignments of all pairs of substrings of u and v, we are interested in each path.
- The paths do not necessarily have to start at $(0,0)$ or end at (m,n).
- Since a path can begin at any node, we have to allow the score 0 in any entry of the matrix.

Local similarity

These considerations lead to the following result:

Theorem 3

Let σ be a score function satisfying

$$loc_\sigma(s, \varepsilon) = loc_\sigma(\varepsilon, s) = 0 \tag{1}$$

for any sequence $s \in \mathcal{A}^*$. Then the following holds:

- *If $i = 0$ or $j = 0$, then $L_\sigma(i, j) = 0$.*
- *Otherwise,*

$$L_\sigma(i,j) = \max \left\{ \begin{array}{l} 0 \\ L_\sigma(i-1,j) + \sigma(u[i] \to \varepsilon) \\ L_\sigma(i,j-1) + \sigma(\varepsilon \to v[j]) \\ L_\sigma(i-1,j-1) + \sigma(u[i] \to v[j]) \end{array} \right\}$$

Local similarity

- In general, it is not easy to verify condition (1).
- However, one usually requires that σ is biased towards negative indel scores, that is, it holds: $\sigma(\alpha \to \beta) < 0$ for all insertions and deletions $\alpha \to \beta$.
- This condition then implies (1):

$$\begin{aligned} \mathit{loc}_\sigma(s, \varepsilon) &= \max\{\mathit{score}_\sigma(x, \varepsilon) \mid x \text{ is a substring of } s\} \\ &= \max\{\mathit{score}_\sigma(\varepsilon, \varepsilon)\} \cup \{\mathit{score}_\sigma(x, \varepsilon) \mid x \text{ is a substring of } s, x \neq \varepsilon\} \\ &= 0 \end{aligned}$$

One can similarly show $\mathit{loc}_\sigma(\varepsilon, s) = 0$.

Based on Theorem 3, we can derive Algorithm 1 which solves the local similarity search problem.

Algorithm 1 (Smith-Waterman Algorithm)

Input: sequences $u = u[1 \ldots m]$ and $v = v[1 \ldots n]$
score function σ satisfying (1)

Output: $loc_\sigma(u, v)$ and an optimal local alignment of u and v.

1. Compute matrix L_σ according to Theorem 3.
2. Compute a maximal entry, say $L_\sigma(i, j)$, in L_σ.
3. Compute optimal local alignments by a traceback from (i, j) on a maximizing path until some entry (i', j') satisfying $L_\sigma(i', j') = 0$ is found.

Algorithm 1 requires $O(mn)$ time and space.

Example 3

Consider the similarity score $\sigma(\alpha \to \beta) = \begin{cases} -1 & \text{if } \alpha = \varepsilon \text{ or } \beta = \varepsilon \\ -2 & \text{if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha \neq \beta \\ +2 & \text{if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha = \beta \end{cases}$

and sequences $u =$ PQRAFADCSTVQ and $v =$ FYAFDACSLL.

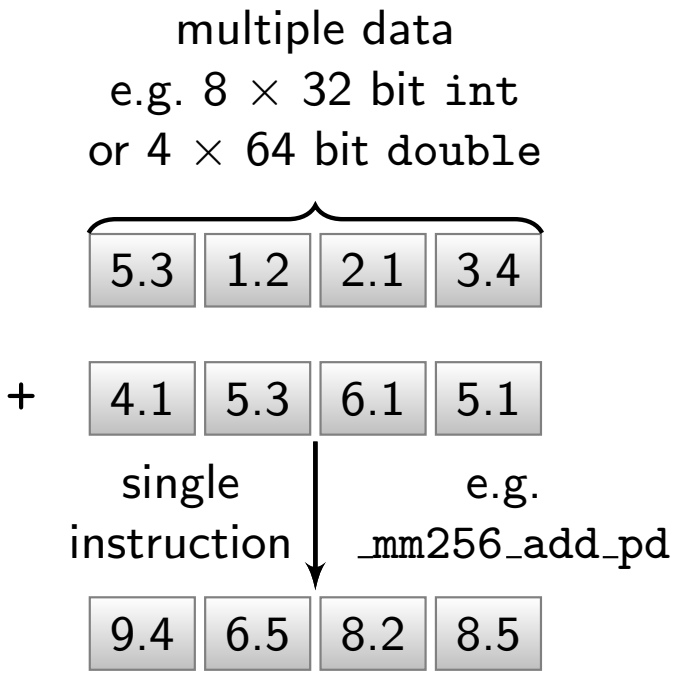
L_σ		F	Y	A	F	D	A	C	S	L	L
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0
R	0	0	**0**	0	0	0	0	0	0	0	0
A	0	0	0	**2**	1	0	2	1	0	0	0
F	0	2	1	1	**4**	**3**	2	1	0	0	0
A	0	1	0	3	3	2	**5**	4	3	2	1
D	0	0	0	2	2	5	**4**	3	2	1	0
C	0	0	0	1	1	4	3	**6**	5	4	3
S	0	0	0	0	0	3	2	5	**8**	7	6
T	0	0	0	0	0	2	1	4	7	6	5
V	0	0	0	0	0	1	0	3	6	5	4
Q	0	0	0	0	0	0	0	2	5	4	3

optimal local alignment:

```
A F - A D C S
A F D A - C S
```

Local similarity

- as the Smith-Waterman Algorithm is so important, there are many implementations available
- the most efficient implementations make use of SIMD-instructions for vectorized computations (originally developed for image processing)
- instructions performed in parallel on short vectors
- widely available standards: AVX2 on X86-CPUs (vectors have 256 bits); Neon on M1-silicon (Mac): vectors have 128 bits



SIMD instructions provide potential speedup of 8/4 for arithmetic ops on int/double-values

Local similarity

Example application

- Given: $k = 2\,550$ protein sequences; total length $n = 997\,571$
- perform all-against-all comparison: compare sequence i and j for all $0 \leq i \leq k-2$ and $i+1 \leq j \leq k-1$

$\Rightarrow$ $\frac{k \cdot (k-1)}{2} = 3\,186\,250$ pairwise sequence comparisons (using SW-alg.)

- total size of all matrices to be computed $\approx 5 \cdot 10^{11}$
- computation time of software developed in the GI-group (and based on SSW[a]): 23 005 ms using 4 threads of an Intel i5 (3.2 GHz)
- $\frac{5 \cdot 10^{11}}{23005} = 21\,734\,405$ matrix entries per ms
- $\frac{3186250}{23005} \approx 138$ matrices per ms

[a] Zhao et. al. 2013