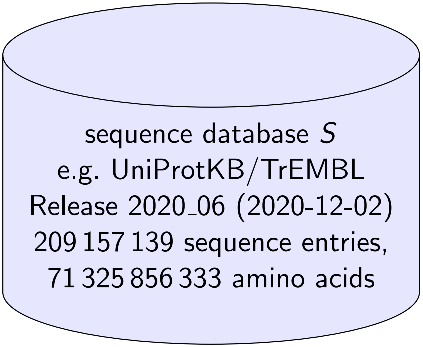
Local Similarity Searches with Fasta

- Fasta is a popular tool for comparing biological sequences.
- It was introduced in [Lipman and Pearson, 1985] and is further described in [Pearson, 1990].
- First consider the problem the Fasta-program was designed for: Let w be a *query sequence* (e.g. a novel DNA-sequence or an unknown protein).
- Let S be a set of sequences (the database), illustrated as follows:


sequence database S
e.g. UniProtKB/TrEMBL
Release 2020_06 (2020-12-02)
209 157 139 sequence entries,
71 325 856 333 amino acids

```
>query sequence w
MPMILGYWNVRGLTHPIRML
```

– The problem is to find all sequences in S, which have local similarities with w and to display these similarities in form of alignments and their positions within the sequences, see Figure 1 for an example.

Figure 1: Sample output of the program `ssearch36` (which implements the Smith-Waterman using SIMD-acceleration), when comparing a single protein sequence `mGSTM1` against a database of 13 143 protein sequences.

```
# ../bin/ssearch36 -q -w 80 ../seq/mgstm1.aa proteindatabase.fasta
1>>>mGSTM1 mouse glutathione transferase M1 - 218 aa
Library: PIR1 Annotated (rel. 66) 5121825 residues in 13143 sequences
Algorithm: Smith-Waterman (SSE2, Michael Farrar 2006) (7.2 Nov 2010)
Parameters: BL50 matrix (15:-5), open/ext: -10/-2
The best scores are:                                             s-w bits E(13143)
sp|P14942|GSTA4_RAT Glutathione S-transferase alpha-4; GST 8-8 (222) 179 49.9 6.1e-07
... (alignments deleted) ...
>>sp|P14942|GSTA4_RAT Glutathione S-transferase alpha-4; GST 8-8;            (222 aa)
Smith-Waterman score: 179; 25.6% identity (54.5% similar) in 75 aa overlap
                 10        20        30        40        50          60         70
mGSTM    MPMILGYWNVRGLTHPIRMLLEYTDSSYDEKRYTMGDAPDFDRSQWLNEKF-KLG-LDFPNLPYL-IDGSHKITQSNA
             : :.. ::  . :: ::  .   ..:         .: ...   ::. : : : : ..: . :::   .::. :
sp|P14 MEVKPKLYYFQGRGRMESIRWLLATAGVEFEE---------EFLETREQYEKLQKDGCLLFGQVPLVEIDG-MLLTQTRA
               10        20        30                 40        50        60         70
... (alignments deleted) ...
218 residues in 1 query   sequences
Total Scan time:  3.820 Total Display time:  0.130
```

Local Similarity Searches with Fasta

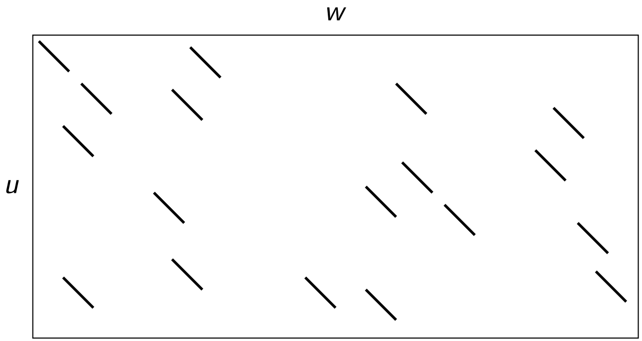
- Applying the Smith-Waterman Algorithm to w and each sequence from S is too slow.
- The idea is to quickly eliminate many sequences from S, which likely do not contain any interesting local alignments.
- The remaining (hopefully few) sequences can be processed by expensive DP-based methods like the SW-Algorithm.
- Such a filtering approach requires a similarity criterion and thresholds referring to the database sequences and the query sequence.
- The criterion must satisfy at least these three conditions:

1. One must be able to quickly determine the database sequences satisfying/not satisfying the criterion.
2. The number of sequences satisfying the criterion must be very small compared to the entire sequence database
3. The sequences not satisfying the criterion do not have high local similarities to the query sequence.

Finding hot spots

- Fasta provides such a filtering approach which is described here.
- There is no formally well-defined model of what Fasta computes, but a heuristic stepwise method defined next.
- Consider an arbitrary but fixed $u \in S$.
- Let q be a fixed constant.
- One chooses $q = 6$ for DNA and $q = 2$ for Proteins.
- The idea is to count for each diagonal the number of common q-grams in u and w.
- In the context of Fasta these are called "hot-spots", see Figure 2 for an illustration.
- The number of hot spots on each diagonal gives a score, according to the definition following the figure.

Figure 2: Hot spots between the query sequence w and the database sequence u, as considered by the Fasta-Algorithm. Each Hot spot represents a q-gram occurring in u and w.



Finding hot spots

Definition 1

Let $m = |u|$ and $n = |w|$. For d, $-m \leq d \leq n$ let

$$hotsp(u, w, d) = |\{(i, j) \mid \underbrace{\overbrace{1 \leq i \leq m - q + 1}^{\text{startpos in } u}, \overbrace{1 \leq j \leq n - q + 1}^{\text{startpos in } w}, \overbrace{j - i = d}^{\text{diag}}, u[i \ldots i + q - 1] = w[j \ldots j + q - 1]}_{\text{matching } q\text{-grams}}\}|$$

So $hotsp(u, w, d)$ is the number of matching q-grams in u and w whose start position pair (i, j) is on diagonal d. We are interested in the maximum $hotsp$-value for all diagonals.

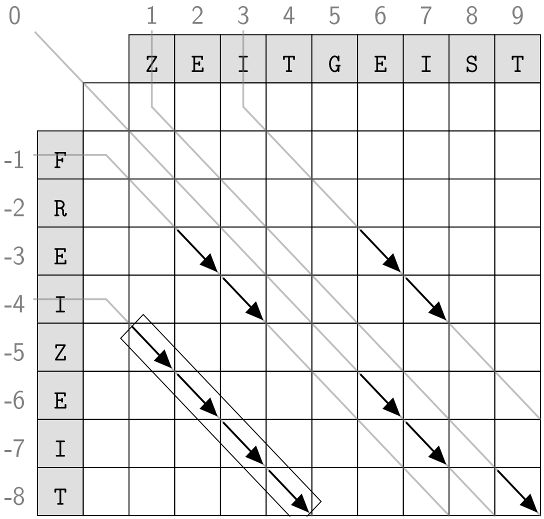
Definition 2

The Fasta score is defined by
$score_{\text{fasta}}(u, w) = \max\{hotsp(u, w, d) \mid -m \leq d \leq n\}$. $\square$

Example 1

Let $q = 2$, $u =$ `FREIZEIT` and $w =$ `ZEITGEIST`. In the table on the right, matching characters are represented by diagonal arcs. Diagonals and their numbers are shown in grey. We have

$$\begin{aligned} hotsp(u, w, -4) &= 3, \\ hotsp(u, w, -1) &= 1, \\ hotsp(u, w, 0) &= 1, \\ hotsp(u, w, 3) &= 1 \text{ and} \\ hotsp(u, w, d) &= 0 \text{ for} \\ d &\notin \{-4, -1, 0, 3\} \end{aligned}$$

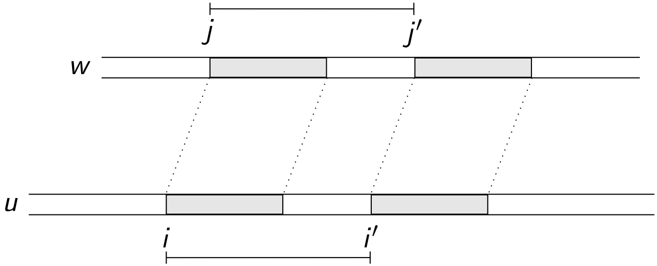


Finding hot spots

- Note that only the q-grams on the same diagonal are counted.
- That is, if

$$u[i \dots i+q-1] = w[j \dots j+q-1] \text{ and}$$
$$u[i' \dots i'+q-1] = w[j' \dots j'+q-1]$$

are on the same diagonal d, we have $j - i = d = j' - i'$ which implies $j - i = j' - i'$ and therefore $i' - i = j' - j$, i.e. the start positions of the matching q-grams have the same distance in both u and w, see the following illustration:



Finding hot spots

- The fact that the order of the q-grams is relevant is the main difference to the q-gram distance model, where the order of the q-grams is not important.
- A crucial step of the Fasta-Algorithm is to first preprocess the query sequence w.
- This preprocessing step (which is independent of the database sequences) gathers information which allows us to efficiently determine the matching q-grams and thus the *hotsp*-values.
- Preprocessing makes sense, as we have to process w many times, namely for each database sequence.
- So the additional effort of the preprocessing likely pays off.
- Algorithm 1 provides details on how $score_{\mathsf{fasta}}(u, w)$ is computed.

Algorithm 1 (Computing $score_{\text{fasta}}(u, w)$)

1. Encode each q-gram as an integer c, $0 \leq c \leq r^q - 1$, where $r = |\mathcal{A}|$. The details of this encoding are described in the section on the q-gram distance.
2. The query sequence w is preprocessed into a table h_w such that for each c, $0 \leq c \leq r^q - 1$ we have

   $$h_w(c) = \{i \mid \underbrace{1 \leq i \leq |w| - q + 1}_{\text{start pos of } q\text{-gram in } w}, c = \underbrace{\overline{w[i \ldots i + q - 1]}}_{\text{code of } q\text{-gram}}\}$$

   That is, $h_w(c)$ holds the positions in w where the q-grams with integer code c occurs.
3. In the final phase, the database is processed as follows:

Algorithm 1 (Computing $score_{\mathrm{fasta}}(u, w)$)

```
1:  $n \leftarrow |w|$
2:  for all $u \in S$ do
3:      $m \leftarrow |u|$
4:      for $d \leftarrow -m$ upto $n$ do
5:          $hotsp(u, w, d) \leftarrow 0$
6:      end for
7:      for $j \leftarrow 1$ upto $m - q + 1$ do
8:          $c \leftarrow u[j \ldots j + q - 1]$
9:          for all $i \in h_w(c)$ do
10:             $hotsp(u, w, j - i) \leftarrow hotsp(u, w, j - i) + 1$
11:         end for
12:     end for
13:     $score_{\mathrm{fasta}}(u, w) \leftarrow \max\{hotsp(u, w, d) \mid -m \leq d \leq n\}$
14: end for
```

Finding hot spots

- The preprocessing of w into table h_w can be done by scanning w twice: In the first scan, for each c, $0 \leq c \leq r^q - 1$, the size of $h_w(c)$ is determined.
- This is the same as computing the q-gram profile of w and takes $O(n + r^q)$ time, as we have seen in the section on the q-gram distance.
- Then one determines for each c, $0 \leq c \leq r^q - 1$, the partial sums $P(c) = \sum_{c' < c} |h_w(c')|$ in $O(r^q)$ time.
- Let H be an array of size $n - q + 1$.
- In a second scan over w one inserts in H the positions in w where a q-gram starts as follows:

```
1: for $i \leftarrow 1$ upto $n - q + 1$ do
2:     $c \leftarrow \overline{w[i \ldots i + q - 1]}$                    $\triangleright$ $O(1)$ time
3:     $H[P(c)] \leftarrow i$
4:     $P(c) \leftarrow P(c) + 1$
5: end for
```

- Thus the preprocessing takes $O(n + r^q)$ time.
- H contains the start positions of all q-grams in w lexicographically ordered by the q-grams (i.e. their integer codes).
- For any c, $0 \leq c \leq r^q - 1$, the subarray $H[\ell \dots P(c) - 1]$ stores the elements in $h_w(c)$ where $\ell =$ if $c = 0$ then 0 else $P(c-1)$.
- So all elements in $h_w(c)$ can be enumerated in $O(|h_w(c)|)$ time.

Example 2

Let $\mathcal{A} = \{\mathtt{a}, \mathtt{c}\}$, $q = 2$ and $w = \mathtt{aaaccacacacaaca}$. So $n = |w| = 15$. In the first step, a scan over w delivers the q-gram profile for w, see the first three columns in the following table.

q-gram	code c	$\lvert h_w(c)\rvert$	$P(c)$
aa	0	3	0
ac	1	5	3
ca	2	5	8
cc	3	1	13

Example 2

From this the partial sums are computed, see column 4. The array H to insert the start positions of the q-grams into is of length

$$\begin{aligned} n-q+1 = 15-2+1 = 14 = 13+1 &= P(3)+|h_w(3)| \\ &= P(r^q-1)+|h_w(r^q-1)| \end{aligned}$$

We again scan w from left to right. The following illustration shows how table P (left side, indexed by q-grams instead of integer codes) and table H (right side) are updated while w is scanned from left to right.

Finding hot spots

aa	ac	ca	cc
0	3	8	13

0	1	2	3	4	5	6	7	8	9	10	11	12	13

Finding hot spots

$\boxed{\texttt{a a}}$ a c c a c a c a c a a c a

aa	ac	ca	cc
1	3	8	13

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1													

Finding hot spots

a $\boxed{\texttt{a a}}$ c c a c a c a c a a c a

aa	ac	ca	cc
2	3	8	13

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2												

Finding hot spots

a a $\boxed{\text{a c}}$ c a c a c a c a a c a

aa	ac	ca	cc
2	4	8	13

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3										

Finding hot spots

a a a $\boxed{\texttt{c c}}$ a c a c a c a a c a

aa	ac	ca	cc
2	4	8	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3										4

Finding hot spots

`a a a c` $\boxed{\texttt{c a}}$ `c a c a c a a c a`

aa	ac	ca	cc
2	4	9	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3					5					4

Finding hot spots

$\texttt{a a a c c}\ \boxed{\texttt{a c}}\ \texttt{a c a c a a c a}$

aa	ac	ca	cc
2	5	9	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3	6				5					4

Finding hot spots

a a a c c a $\boxed{\texttt{c a}}$ c a c a a c a

aa	ac	ca	cc
2	5	10	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3	6				5	7				4

Finding hot spots

a a a c c a c $\boxed{\texttt{a c}}$ a c a a c a

aa	ac	ca	cc
2	6	10	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3	6	8			5	7				4

Finding hot spots

a a a c c a c a $\boxed{\texttt{c a}}$ c a a c a

aa	ac	ca	cc
2	6	11	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3	6	8			5	7	9			4

Finding hot spots

`a a a c c a c a c` $\boxed{\texttt{a c}}$ `a a c a`

aa	ac	ca	cc
2	7	11	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3	6	8	10		5	7	9			4

Finding hot spots

a a a c c a c a c a $\boxed{\texttt{c a}}$ a c a

aa	ac	ca	cc
2	7	12	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2		3	6	8	10		5	7	9	11		4

Finding hot spots

a a a c c a c a c a c $\boxed{\texttt{a a}}$ c a

aa	ac	ca	cc
3	7	12	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	12	3	6	8	10		5	7	9	11		4

Finding hot spots

a a a c c a c a c a c a $\boxed{\texttt{a c}}$ a

aa	ac	ca	cc
3	8	12	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	12	3	6	8	10	13	5	7	9	11		4

Finding hot spots

`a a a c c a c a c a c a a` $\boxed{\texttt{c a}}$

aa	ac	ca	cc
3	8	13	14

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	12	3	6	8	10	13	5	7	9	11	14	4

From the final values in table P we can, for each integer code c, deduce the range in H where the elements in $h_w(c)$ are stored:

q-gram	code c	$h_w(c)$	subarray of H representing $h_w(c)$
aa	0	$\{1, 2, 12\}$	$H[0, \dots, 2]$
ac	1	$\{3, 6, 8, 10, 13\}$	$H[3, \dots, 7]$
ca	2	$\{5, 7, 9, 11, 14\}$	$H[8, \dots, 12]$
cc	3	$\{13\}$	$H[13, \dots, 13]$

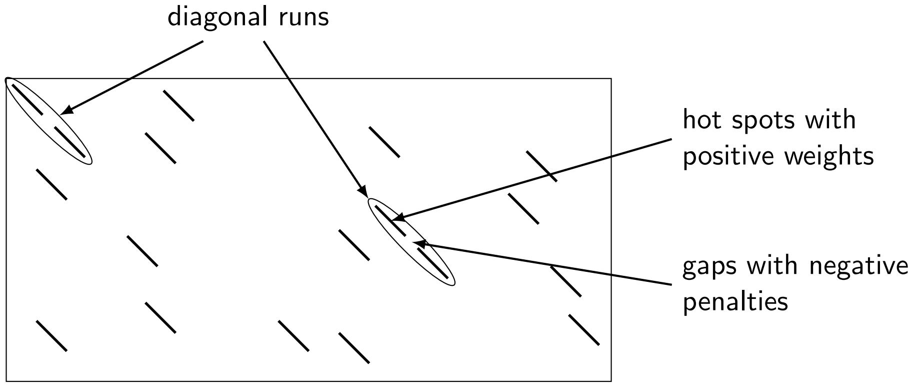
Finding hot spots

- The total number of start positions in $h_w(c)$ enumerated in line 9 of Algorithm 1, is the same as the number of common q-grams in u and w, which is $\sum_{d=-m}^{n} hotsp(u, w, d)$.
- Thus the running time of Algorithm 1 is clearly $O(r^q + m + n + \sum_{d=-m}^{n} hotsp(u, w, d))$ for one database sequence u.
- That is, the more common q-grams sequences w and u contain, the more time the algorithm requires.
- So the algorithm does not waste time on database sequences, which are filtered out, as they have a too small fasta-score.

Combining hot spots to diagonal runs

- If the Fasta-score for some database sequence w is smaller than some minimum threshold, then it is discarded.
- Otherwise, w is processed further by looking for diagonal runs in the Edit-distance-matrix (without computing the matrix, of course).
- Diagonal runs are hot spots appearing on the same diagonal, with small gaps in between, see Figure 3 for an illustration.
- To score diagonal runs, one assigns positive weights to the hot spots and negative penalties to gaps.
- Note that not necessarily all hot spots on the same diagonal are put into a single diagonal run.
- Instead, a diagonal can contain more than one diagonal run.

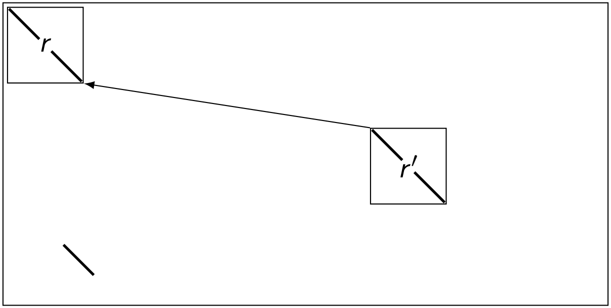
Figure 3: Diagonal runs in the fasta algorithm



Constructing a directed graph from diagonal runs

- In the next step, a directed graph is constructed.
- The nodes of the graph are the diagonal runs from the previous step with corresponding weights assigned.
- Let us denote a diagonal run r by the upper left corner $(\ell_1(r), \ell_2(r))$ and the lower right corner $(h_1(r), h_2(r))$.
- The nodes for diagonal runs r and r' are connected if $h_1(r) < \ell_1(r')$ and $h_2(r) < \ell_2(r')$, see Figure 4 for an illustration.
- The edges get a negative weight.
- The graph is obviously acyclic and therefore we can efficiently compute a path of maximal total weight.
- In the lecture 'Genome Informatics' we will have a closer look at how these paths are efficiently computed.

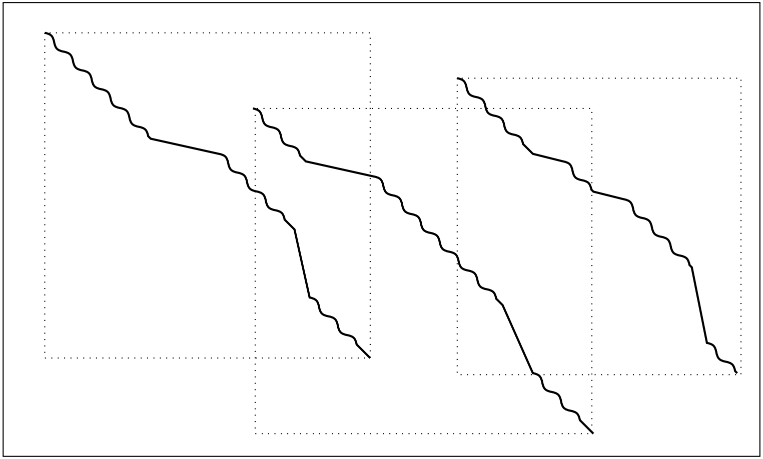
Figure 4: Diagonal runs (in the boxes) are connected by edges with negative weight.



Constructing a directed graph from diagonal runs

- Suppose that all paths of maximal weight are computed.
- From each path we pick the first and the last node.
- The upper left corner of the first node and the lower right corner of the last node define a pair of substrings of w and u.
- These are aligned using standard global alignment algorithms, see Figure 5 for illustration.
- If the score of the optimal global alignment achieves some minimum threshold, then it is reported as a local alignment of the sequence pair in which it appears.

Figure 5: Optimal paths consisting of diagonal runs define a pair of substrings of w and u.



Lipman, D. and Pearson, W. (1985).
Rapid and Sensitive Protein Similarity Search.
Science, 227:1435–1441.

Pearson, W. (1990).
Rapid and Sensitive Sequence Comparison with FASTP and FASTA.
In Doolittle, R., editor, *Methods in Enzymology*, volume 183, pages 63–98. Academic Press, San Diego, CA.