Single alignment: FASTA

17 march 2017

FASTA

FASTA is a DNA and protein sequence alignment software package first described (as FASTP) by David J. Lipman and William R. Pearson in 1985.[1]

FASTA is pronounced "fast A", and stands for "FAST-All", because it works with any alphabet, and it is an extension of "FAST-P" (protein) and "FAST-N" (nucleotide) alignment.

The current FASTA package contains programs for protein:protein, DNA:DNA, protein:translated DNA (with frameshifts), and ordered or unordered peptide searches.

Recent versions of the FASTA package include special translated search algorithms that correctly handle frameshift errors (which six-frame-translated searches do not handle very well) when comparing nucleotide to protein sequence data.

[1] Lipman, DJ; Pearson, WR (1985). "Rapid and sensitive protein similarity searches". Science. 227 (4693): 1435–41.

FASTA

In addition to rapid heuristic search methods, the FASTA package provides SSEARCH, an implementation of the optimal Smith-Waterman algorithm.

A major focus of the package is the calculation of accurate similarity statistics, so that biologists can judge whether an alignment is likely to have occurred by chance, or whether it can be used to infer homology.

The FASTA program in its classic version, is a heuristic program able to search the **global sequence similarity**. Two variants thereof, and LFASTA PLFASTA, are able to search for **local sequence similarity**.

The FASTA package is available from fasta.bioch.virginia.edu.

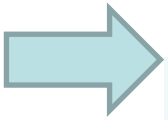
FASTA

FASTA is a four steps algorithm.

First Step: OFFSET Definition

Initially, a table is created and that contains all the positions for each type of amino acid (or nucleotide) within each of the sequences present in the database. For example, if it exists in the database a sequence like the one below, it is created the table at the right side:

Position	1	2	3	4	5	6	7
Sequence A	F	L	W	R	T	W	S

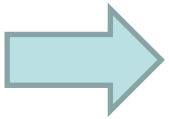


F	1
L	2
W	3, 6
R	4
T	5
S	7

FASTA

First Step: OFFSET Definition

Position	1	2	3	4	5	6
Sequence B	S	W	R	T	W	T



S	1
W	2, 5
R	3
T	4, 6

The positional table can be constructed taking into account the position of the amino acids taken individually (ktup = 1) (as is the case above) or as taken in pairs (ktup = 2). Using this second mode will result in a speeding up of the process at the expense of the accuracy of the final data.
However, the approximation is still valid because it can be assumed that the homology between two sequences are meaningful only if it can be considered pairs of amino acids and not individual amino acids. In the case of nucleotide sequences ktup is 4 or 6.

FASTA

First Step: OFFSET Definition

At this point you have to run the mathematical difference of positional values of the amino acids of the same type in the sequence that is being compared (sequence query) and the sequences in the database. This difference is also called OFFSET.

In the example below, the amino acids common to the two sequences are S, W (present in two positions), R and T.

The comparison of the positional values of these amino acids in the two sequences gives rise to these values of OFFSET:

F	**1**
L	**2**
W	**3, 6**
R	**4**
T	**5**
S	**7**

S	**1**
W	**2, 5**
R	**3**
T	**4, 6**

AA	DELTA	OFFSET
S	7 - 1	6
W	3 – 2	1
W	3 – 5	-2
W	6 – 2	4
W	6 – 5	1
R	4 – 3	1
T	5 – 4	1
T	5 – 6	-1

FASTA

First Step: OFFSET Definition

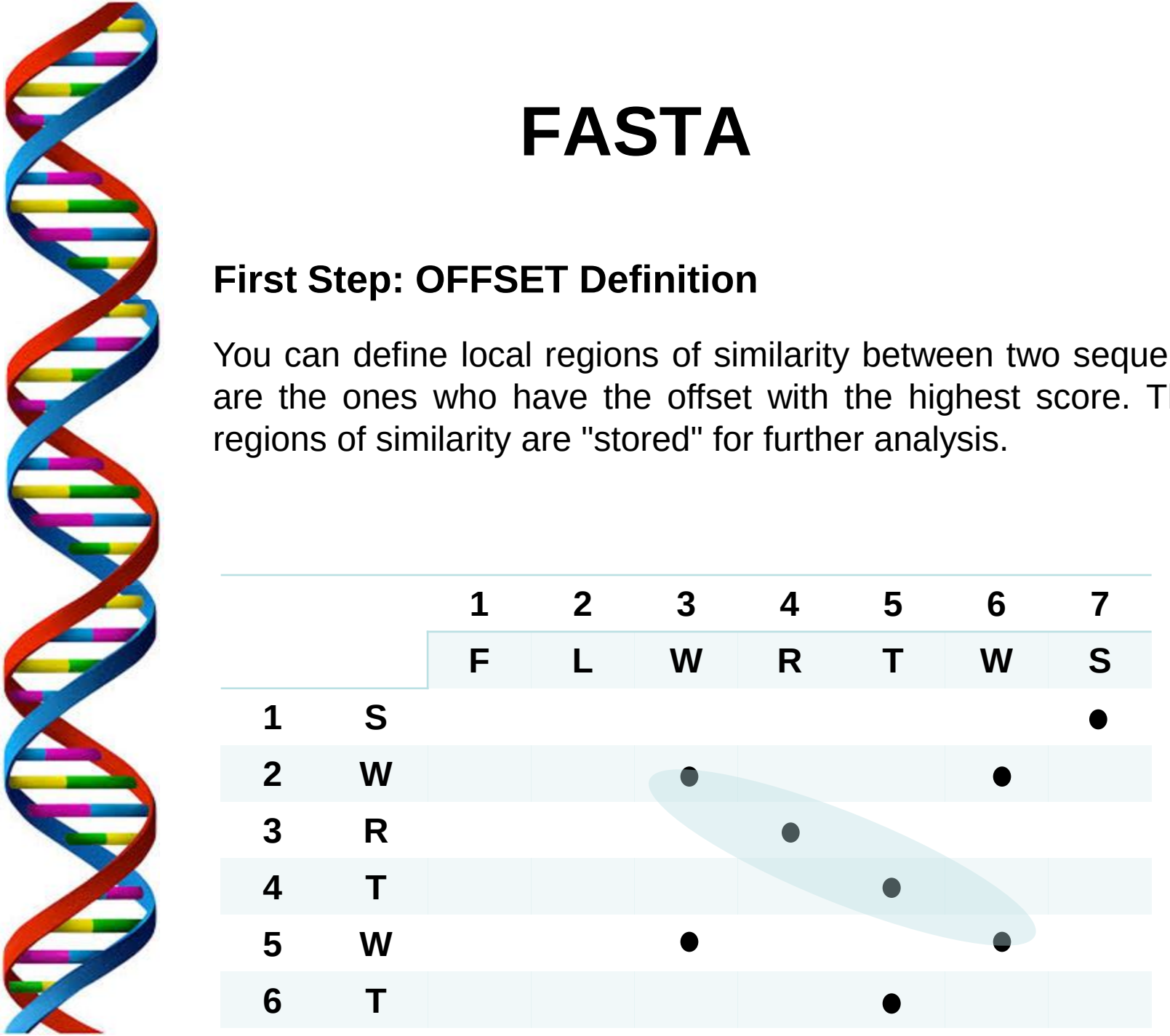
In the example, the best offset is the one with the value 1 which allows alignment of 4 amino acids. The other offsets allow the alignment of a single amino acid or no amino acid. The score in an offset is increased for each identity and reduced for each misalignment (mismatch). The latter is allowed, while insertions / deletions that are forbidden.

		1	2	3	4	5	6	7
		F	L	W	R	T	W	S
1	S							●
2	W			●			●	
3	R				●			
4	T					●		
5	W			●			●	
6	T					●		

FASTA

First Step: OFFSET Definition

You can define local regions of similarity between two sequences. They are the ones who have the offset with the highest score. The 10 best regions of similarity are "stored" for further analysis.



		1	2	3	4	5	6	7
		F	L	W	R	T	W	S
1	S							●
2	W			●			●	
3	R				●			
4	T					●		
5	W			●			●	
6	T					●		

FASTA

Second Step: Evaluation of Amino Acids Substitution

This step evaluates any **replacements** that occurred between amino acids in the **10 best regions of similarity** selected in the first phase, using for this purpose the **empirical matrices** (scoring matrix).

Several king of matrix can be used, including the most famous and popular ones that are the PAM and BLOSUM.

The most widely used of which it is the PAM 250.

Such matrices define a score for each possible substitution of amino acids, based on the frequency of mutations that occurred during the time.

FASTA

Second Step: Evaluation of Amino Acids Substitution

To each of the *10 best regions* of similarity it is then assigned a score based on the selected score matrix and *for each of them it is identified those residues that contribute to define the maximum score*.

The defined sub-region is called the initial region (initial region) and its score is said initial score or **INIT1** (in FASTP it is called INITN).

This score is used as a parameter to define the **similarity** between two sequences (similarity score).

To calculate the initial score you can use a ktup = 1 or 2.

FASTA

Second Step: Evaluation of Amino Acids Substitution

The method that uses ktup = 2 is much faster (about 5 times) than the one that uses ktup = 1, but will also increase the level of imprecision. The accuracy remains still acceptable for long proteins and nucleotide sequences, instead, this accuracy is unacceptable in the case of oligonucleotides or oligopeptide.

- The oligonucleotides are short (oligo) nucleotide sequences (RNA or DNA), typically with 20 or fewer base pairs.
- An oligopeptide is a molecule formed by the condensation of certain amino acids in a number which, by convention, varies from 10 to 50. The amino acids that form an oligopeptide are linked together through a series of peptide bonds.

The **initial region** *with the highest score* is used to create a ranking of the collation sequences found in the database in order to define which of them are most similar to the input sequence.

FASTA

Third Step: Join several initial regions

This is the new step introduced by **FASTA** with respect to FASTP/FASTN.

FASTA carries out an assessment about possible ways to connect (join) different starting regions. The constraints for the creation of the link are:

-*Exclusion* of potential areas of *overlapping* between regions
-*Score* must be *higher* than a "threshold value"
-Introduction of a *penalty* score (-16) for each gap (the gap penalty).

Given the location of the initial regions and their respective scores, FASTA assigns a score penalty at intermediate regions without similarity.

FASTA

Third Step: Join several initial regions

An algorithm assesses whether the introduced penalty lowers the score below a certain "threshold-value."

If it does not, FASTA calculates the optimal alignment of initial regions. Such alignment is defined by *the join of the earliest regions (a sub-set) showing highest score.*

Consequently the initial score **INIT1** is also recalculated being redefined on the basis of created joins, and that is called **INITN**.

The **rank** of the comparison sequences in the database is also recalculated.

This step increases the sensitivity of the method at the expense of specificity. The reduction of specificity is somewhat compensated by the inclusion in the join of only the initial regions that show a score higher than a threshold value (OPTCUT). The OPTCUT is about one standard deviation above the mean score expected for non-correlated sequences in the database.

FASTA

Fourth Step : Deletions and insertions evaluation:

Sequences showing greater similarity are aligned to the input sequence using a methods that is based on a modified version of Needleman and Wunsch' algorithm.

Needleman and Wunsch' algorithm is based on dynamic programming. It takes into account possible deletions or insertions of amino acids (or nucleotides).

- The variant proposed by Lipman and Pearson does not take into account the sequence regions that are not similar. This allows for an optimized score (OPT).

FASTA

Fourth Step : Deletions and insertions evaluation:

The final comparison is made among

- all possible alignments for the input sequence that are found during the second phase and
- the similar sequences extracted from the database that fall within a range of 32 amino acids, centred around the initial region that has the highest score.

In this phase, it is used a gap penalty of -12 for the first residue missing and -4 for each additional residue.

FASTA

Fourth Step : Deletions and insertions evaluation:

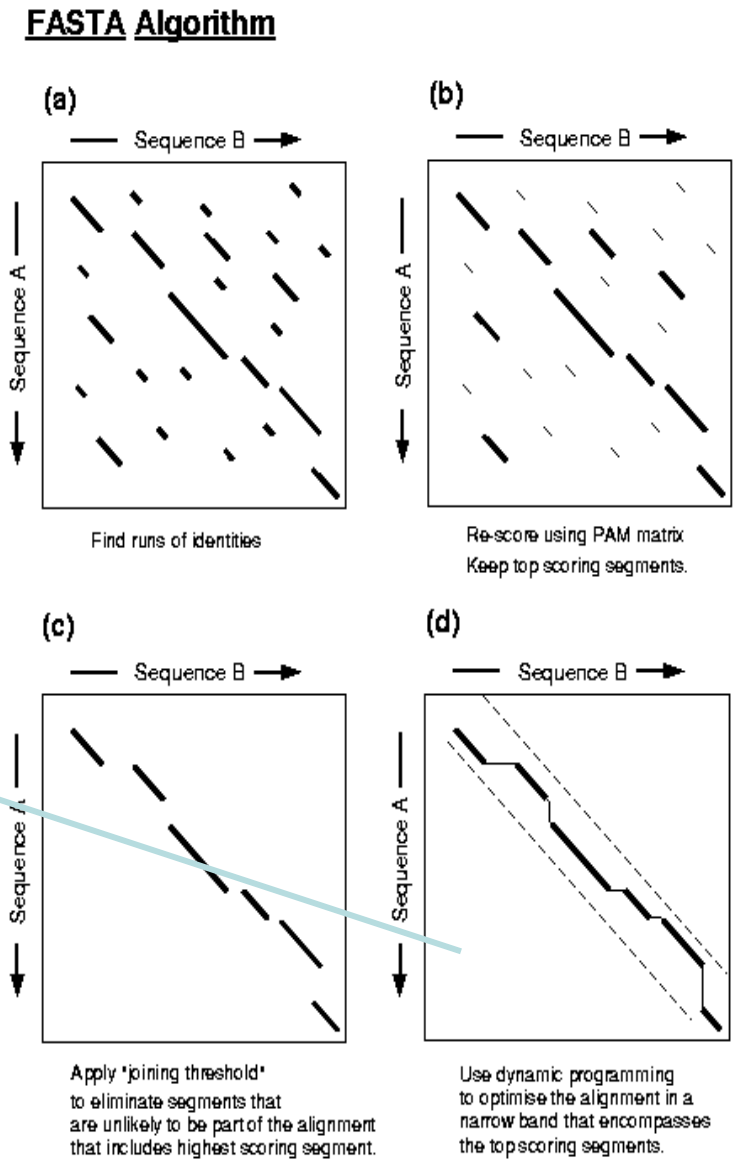
Therefore FASTA defines three different scores:

INIT1: it is the first score calculated, the one obtained by the second phase after the usage of the score matrix
INITN: that is calculated after the introduction of the join, in the third phase
OPT: optimized score, after evaluation of the insertions and deletions, in the fourth phase

FASTA

Summarizing FASTA

The final score is **Opt**.



Dynamic Programming

Divide et Impera vs Dynamic Programming

- The basic steps of the algorithms based on the technique Divide and Conquer to solve a given algorithmic problem:
  1. Divide the problem into sub-problems of smaller size
  2. Solve (recursively) sub-problems of smaller size
  3. Combine the solutions of the sub-problems into a solutionfor the original problem
- Typically the sub-problems obtained from step 1. can be different, therefore, each of them was individually resolved from the recursive step Call 2.
- In many situations, the sub-problems obtained in step 1 may be similar, or even the same. In this case, the algorithm based on D & I solves the same problem several times, doing unnecessary work!

Divide et Impera vs Dynamic Programming

- In such situations (ie when the subproblems of a given algorithmic problem tend to be "similar", or even "Equal"), it is useful to employ the technique of Dynamic Programming
- This technique is essentially similar to D & I, having the characteristic of solving each sub-problem only once.
- Hence, algorithms based on Dynamic Programming are more efficient.
- Basic idea behind Dynamic Programming:
  - It calculates the solution to distinct sub-problems only once
  - It stores these solutions in a table, in such a way that they can be used in the following, if necessary.

Example: Fibonacci numbers

- The sequence F0, F1, F2, F3, . . . of Fibonacci numbers is defined by the following recurring equation:
  - F0 = F1 = 1
  - Fn = Fn−1 + Fn−2 For n>2

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, . . .

Example: Fibonacci numbers

- D&I algorithm based on the Fibonacci numbers' definition will be

$\textsc{Fib}(n)$

if $n \leq 1$ **then return** 1

else return $\textsc{Fib}(n-1)$+$\textsc{Fib}(n-2)$

The complexity T(n) of Fib(n) is

T(0)=T(1)=1

T(n)=T(n-1)+T(n-2)+1 , n>=2

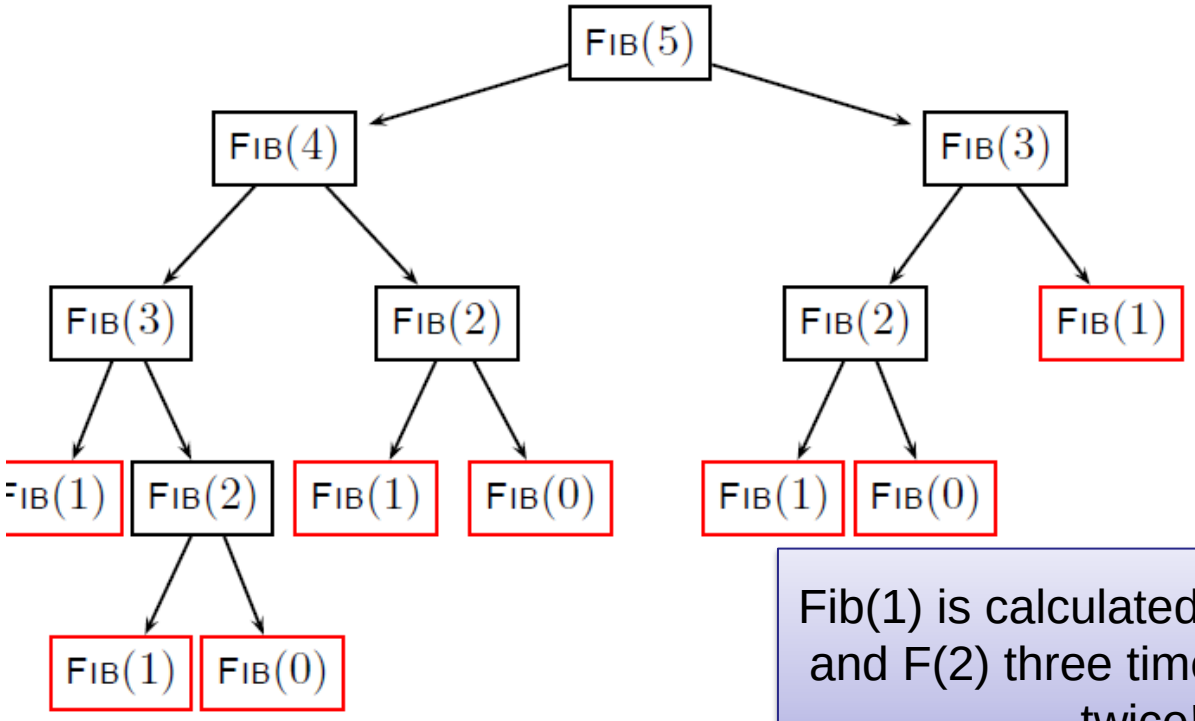
By defining T'(n)=T(n)+1, we obtain

T'(0)=T'(1)=2

T'(n)=T'(n-1)+T'(n-2), n>=2

Example: Fibonacci numbers

Let us run Fib(5):



Fib(1) is calculated 5 times, F(0) and F(2) three times and Fib(3) twice!

Example: Fibonacci numbers

Dynamic Programming

MEMFIB(n)

if $n \leq 1$ **then return** 1

else if $F[n]$ non é definitito

$F[n] \leftarrow$ MEMFIB$(n-1)$ + MEMFIB$(n-2)$

return $F[n]$

Algorithm 1: use an array of n+1 elements, $O(n^2)$ time complexity and O(n) space complexity

ITERFIB(n)

$F[0] \leftarrow 1, F[1] \leftarrow 1$

for $i \leftarrow 2$ **to** n **do**

$F[i] \leftarrow F[i-1] + F[i-2]$

return $F[n]$

Algorithm 2: use an array of 2 elements, $O(n^2)$ time complexity