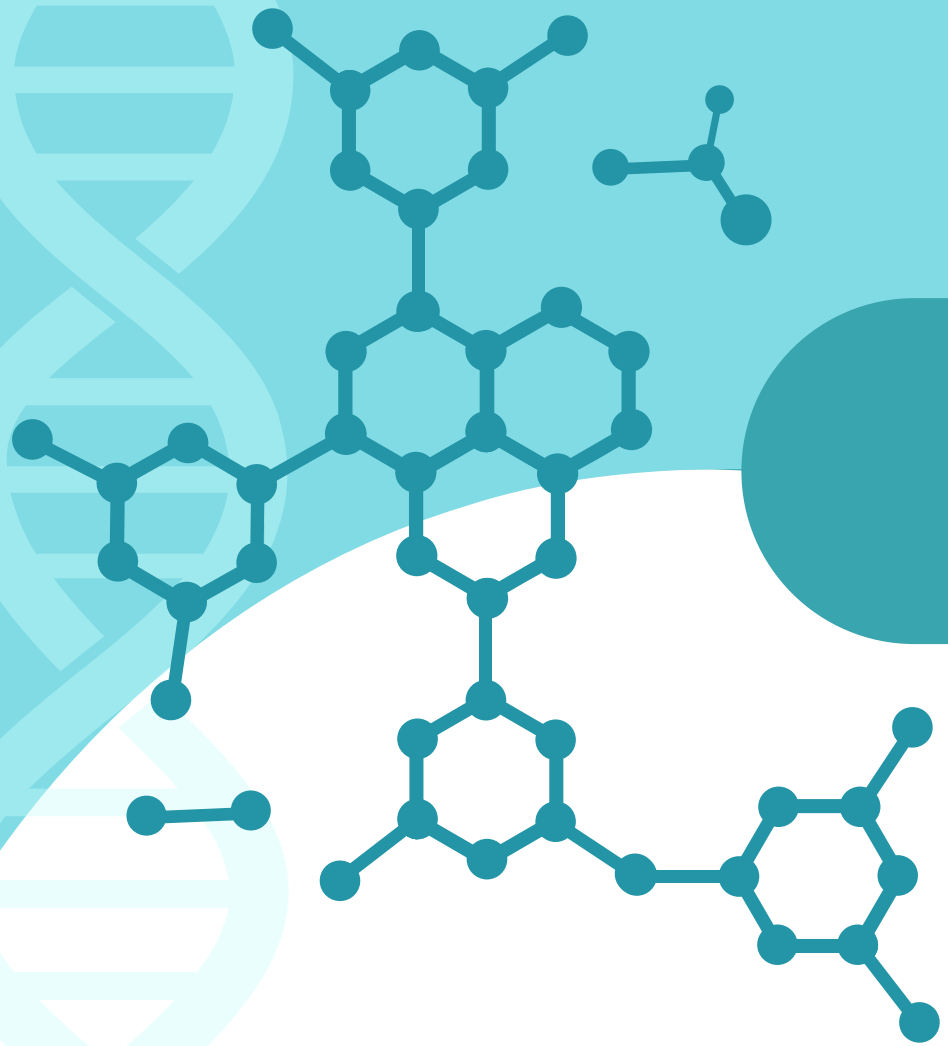
Bioinformatics LAB 2

Alignment fundamentals

Prof.ssa Elisa Ficarra
Prof.ssa Santa Di Cataldo
Eng. Marta Lovino
Eng. Alessio Mascolini

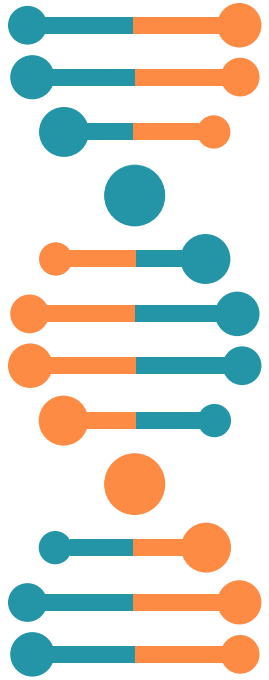
Politecnico di Torino
DAUIN
Dept. of Control and Computer Engineering



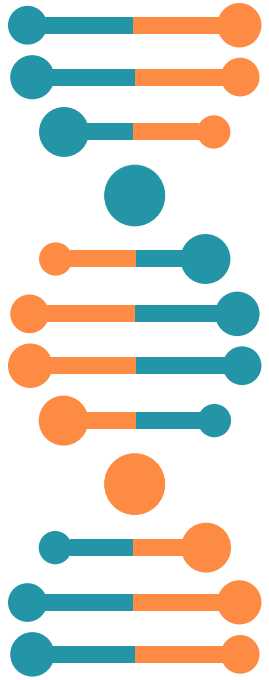
LAB 2 - Objectives

Objectives

- Understand global and local alignment
- Use popular aligner aligner



Global alignment



Definitions:

$s(a,b)$	the substitution score for aligning letters a and b
g	the gap score for aligning any letter to a null
X_i	the partial sequence consisting of the first i letters of $X \equiv x_1 x_2 \dots x_m$
Y_j	the partial sequence consisting of the first j letters of $Y \equiv y_1 y_2 \dots y_n$
$SIM(i,j)$	the similarity of X_i and Y_j

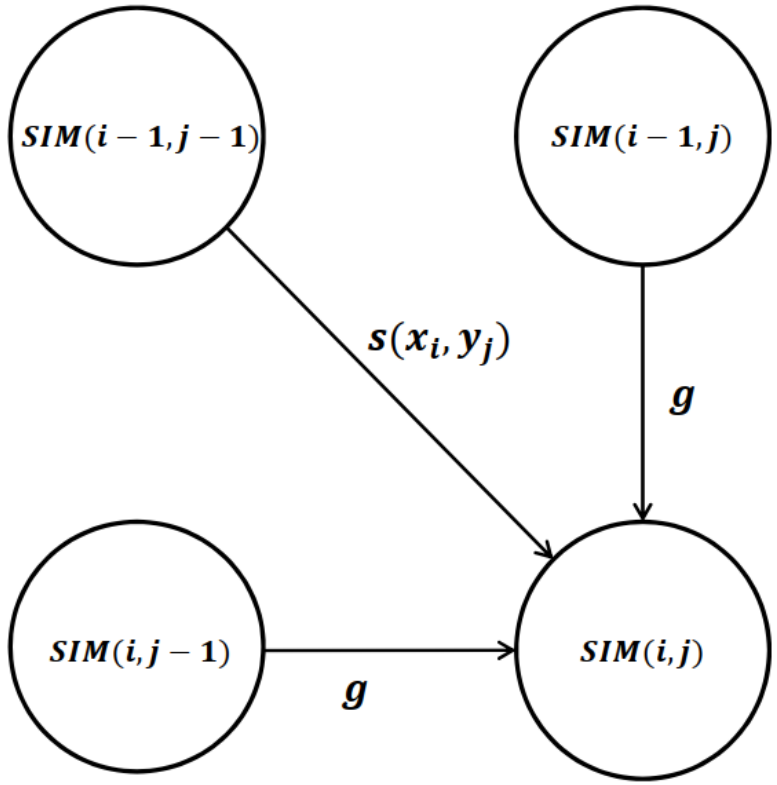
Consider the *last column* of an optimal alignment of X_i and Y_j. This column either aligns x_i to y_j, or x_i to a null, or y_j to a null. Because we do not allow "crossing", there are no other possibilities. This observation yields the following recurrence:

$$\boldsymbol{SIM(i,j) = max}\begin{cases} \boldsymbol{SIM(i-1,j-1) + s(x_i, y_j)} & x_i \text{ and } y_j \text{ aligned} \\ \boldsymbol{SIM(i-1,\ j) + g} & x_i \text{ aligned with a null} \\ \boldsymbol{SIM(i,\ j-1) + g} & y_j \text{ aligned with a null} \end{cases}$$

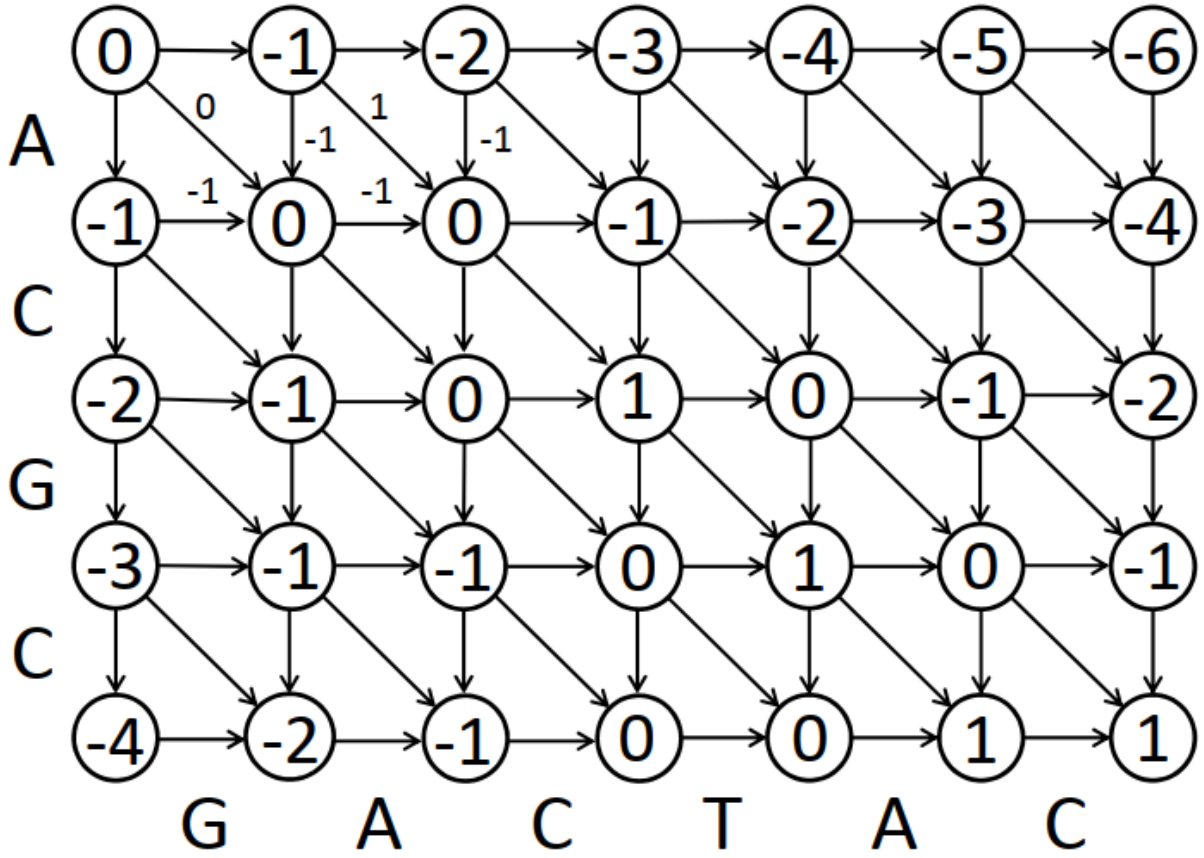
In brief, we can solve for $SIM(m,n)$ by solving smaller versions of the problem first.

Global alignment

One may associate a partial similarity with each node of a path graph. If the values of $SIM(i-1, j-1)$, $SIM(i-1, j)$ and $SIM(i, j-1)$ are known, the value of $SIM(i, j)$ may be calculated.

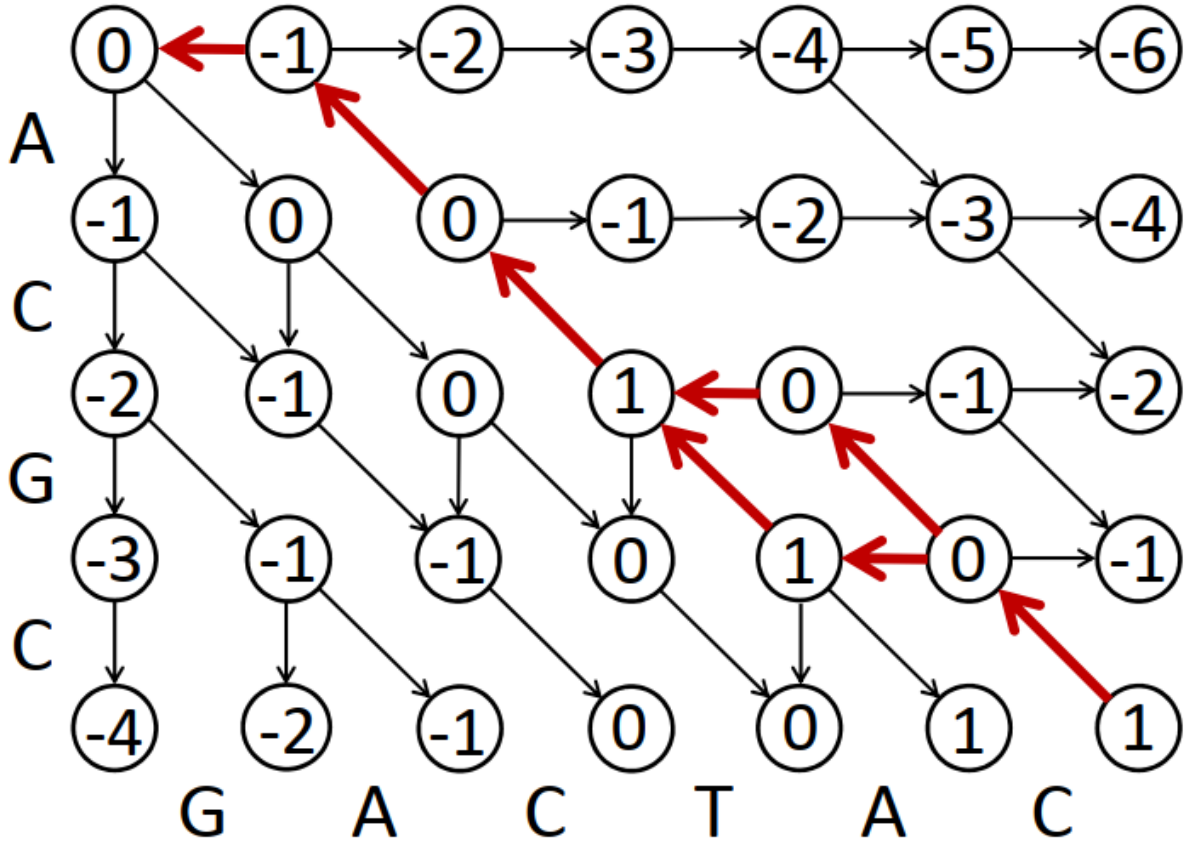


Global alignment



Scores: Match +1 Mismatch 0 Gap -1

Global alignment



Optimal alignments: **–ACG–C / GACTAC** and **–AC–GC / GACTAC**

Global alignment

		G	A	C	T	A	C
A							
C							
G							
C							

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1						
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1						
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0					
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0					
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0				
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2		
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	
C	-2						
G	-3						
C	-4						

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	-4
C	-2	-1	0	1	0	-1	-2
G	-3	-1	-1	0	1	0	-1
C	-4	-2	-1	0	0	1	1

Scores:

Match	+1
Mismatch	0
Gap	-1

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	-4
C	-2	-1	0	1	0	-1	-2
G	-3	-1	-1	0	1	0	-1
C	-4	-2	-1	0	0	1	1

Scores:

Match	+1
Mismatch	0
Gap	-1

C

C

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	-4
C	-2	-1	0	1	0	-1	-2
G	-3	-1	-1	0	1	0	-1
C	-4	-2	-1	0	0	1	1

Scores:

Match	+1
Mismatch	0
Gap	-1

GC

AC

–C

AC

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	-4
C	-2	-1	0	1	0	-1	-2
G	-3	-1	-1	0	1	0	-1
C	-4	-2	-1	0	0	1	1

Scores:

Match	+1
Mismatch	0
Gap	-1

-GC
TAC

G-C
TAC

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	-4
C	-2	-1	0	1	0	-1	-2
G	-3	-1	-1	0	1	0	-1
C	-4	-2	-1	0	0	1	1

Scores:

Match	+1
Mismatch	0
Gap	-1

C-GC
CTAC

CG-C
CTAC

Global alignment

		G	A	C	T	A	C
	0	-1	-2	-3	-4	-5	-6
A	-1	0	0	-1	-2	-3	-4
C	-2	-1	0	1	0	-1	-2
G	-3	-1	-1	0	1	0	-1
C	-4	-2	-1	0	0	1	1

Scores:

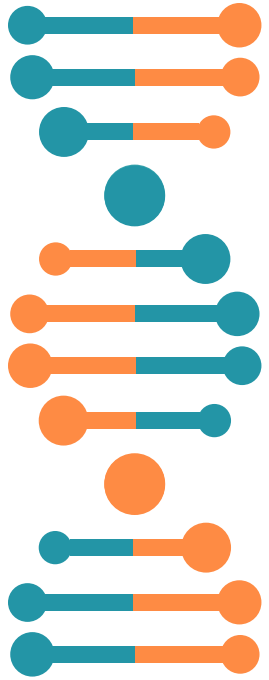
Match	+1
Mismatch	0
Gap	-1

-AC-GC
GACTAC

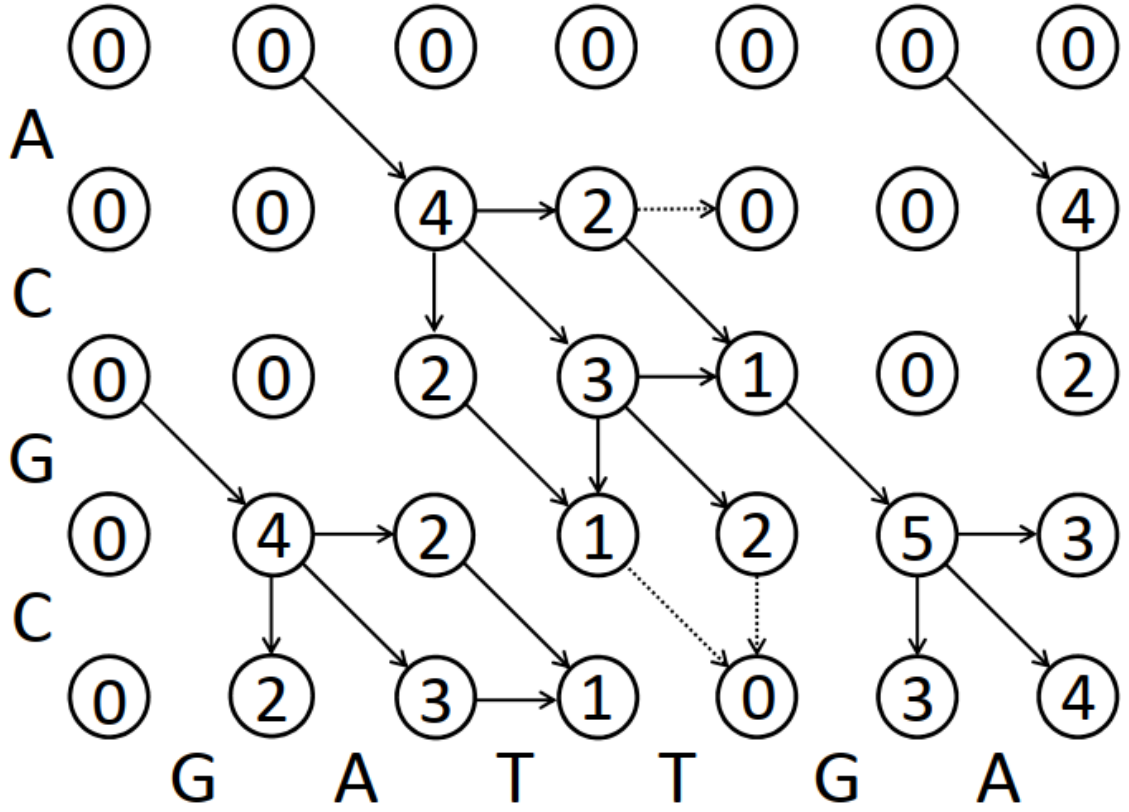
-ACG-C
GACTAC

Global alignment

```
Similarity(X,Y):
    For i = 0,...,m:  SIM[i,0] = i*g
    For j = 1,...,n:   SIM[0,j] = j*g
    For i = 1,...,m:
        For j = 1,...,n:
            SIM[i,j] = max(
                SIM[i-1,j-1] + s(X[i],Y[j]),
                SIM[i-1,j]+g,
                SIM[i,j-1]+g
            )
        EndFor
    EndFor
Return SIM[m,n]
```

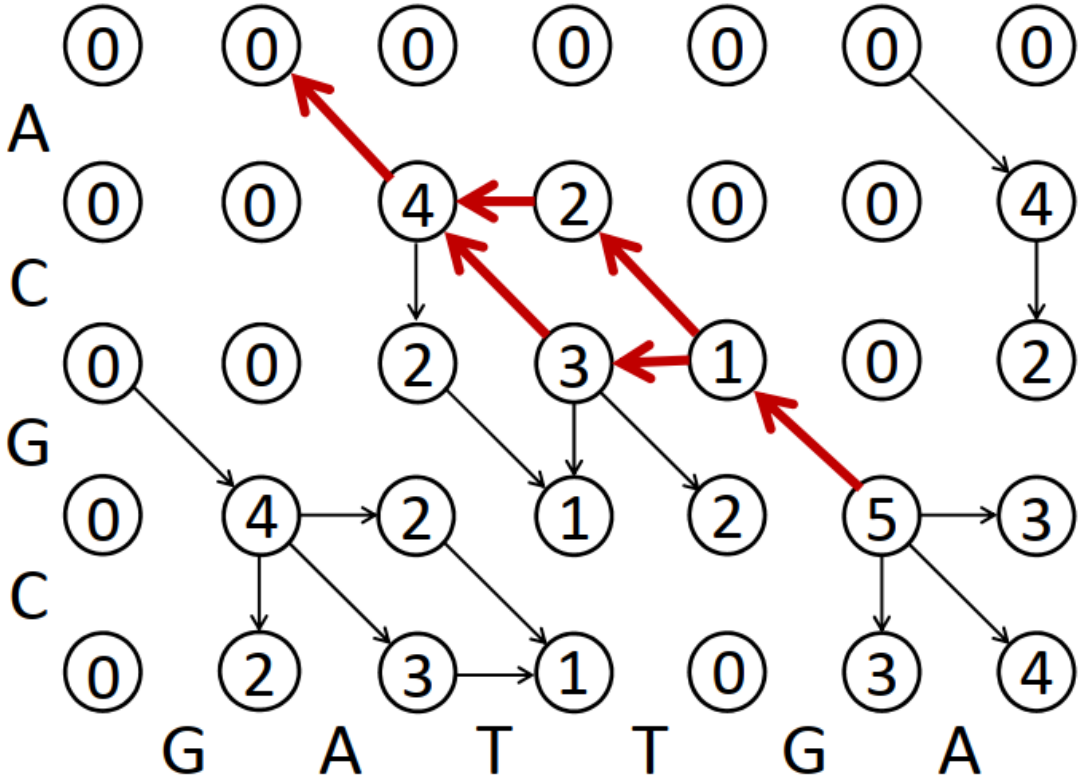


Local alignment



Scores: Match +4 Mismatch -1 Gap -2

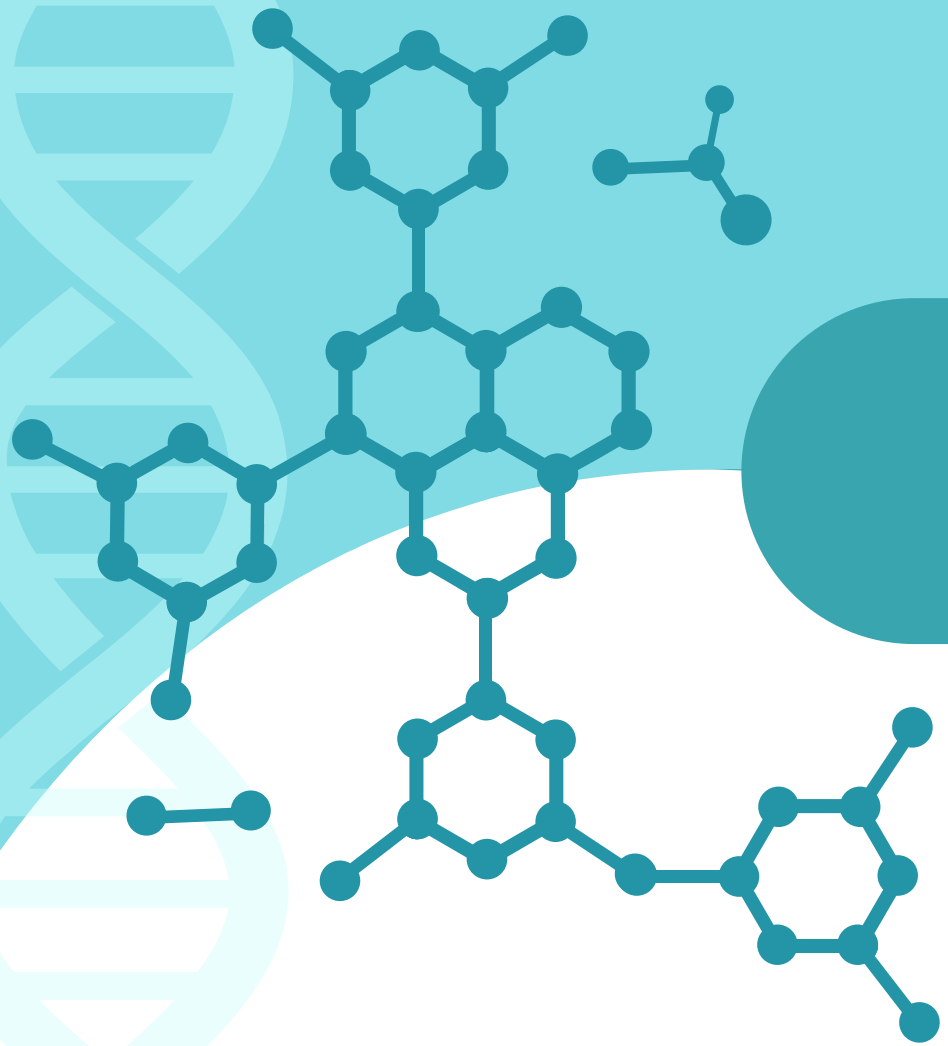
Local alignment



Scores: Match +4 Mismatch -1 Gap -2

Local alignment

```
Local_Similarity(X,Y):
   S=0
   For i = 0,...,m:  SIM[i,0] = i*g
   For j = 1,...,n:   SIM[0,j] = j*g
   For i = 1,...,m:
      For j = 1,...,n:
         SIM[i,j] = max(
            0,
            SIM[i-1,j-1] + s(X[i],Y[j]),
            SIM[i-1,j]+g,
            SIM[i,j-1]+g
         )
         S=max(S,SIM[i,j])
      EndFor
   EndFor
Return S
```



LAB 2 - Assignments

Assignment 1: Global alignment

Under the assumption that both input sequences a and b stem from the same origin, a global alignment tries to identify matching parts and the changes needed to transfer one sequence into the other. The changes are scored and an optimal set of changes is identified, which defines an alignment. The dynamic programming approach tabularizes optimal subsolutions in matrix D, where an entry $D_{i,j}$ represents the best score for aligning the prefixes $a_{1..i}$ with $b_{1..j}$.

To better clarify how global alignment works, take a look here: http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Needleman-Wunsch

Then, write a Python program that given two sequences (passed as first and second argument from command line) and match, mismatch and gap costs (passed as third, fourth, fifth argument from command line):

1. Compute matrix D and output it on the terminal, along with the final alignment score
2. Output the final alignment (if two sequences have more than one alignment with the same score, provide one of them e.g. check website for ‘AACCG’ and ‘AACG’)
3. Check your alignment on Freiburg website

Assignment 1 : Global alignment

Usage should be something like this:

```
python global_alignment.py AACCG AACG 1 -1 -2
```

Output:

```
Global alignment score: 2.0
[[  0.  -2.  -4.  -6.  -8.]
 [ -2.   1.  -1.  -3.  -5.]
 [ -4.  -1.   2.   0.  -2.]
 [ -6.  -3.   0.   3.   1.]
 [ -8.  -5.  -2.   1.   2.]
 [-10.  -7.  -4.  -1.   2.]]


Final alignment:
AACCG
||| |
AAC-G
```

Assignment 2: Local alignment

A local alignment approach tries to identify the most similar subsequences that maximize the scoring of their matching parts and the changes needed to transfer one subsequence into the other. The dynamic programming approach tabularizes optimal subsolutions in matrix S, where an entry $S_{i,j}$ represents the maximal similarity score for any local alignment of the (sub)prefixes $a_{x..i}$ with $b_{y..j,}$ where x,y>0 are so far unknown and have to be identified via traceback.

To better clarify how local alignment works, take a look here: http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Smith-Waterman

Then, write a Python program that given two sequences (passed as first and second argument from command line) and match, mismatch and gap cost (passed as third, fourth, fifth argument from command line).

1. Compute matrix D and output it on the terminal, along with the final alignment score (**remember that the minimum value you can have in the matrix is 0!!**)
2. Output the final alignment (if two sequences have more than one alignment with the same score, provide one of them)
3. Check your alignment on Freiburg website
4. **Local alignment has a lot of concepts similar to global alignment, so you can reuse a lot of code you have previously written for global alignment!**

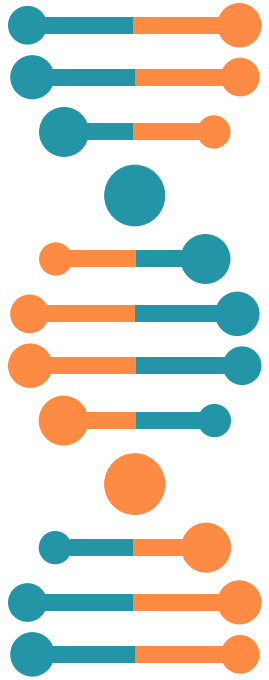
Assignment 2: Local alignment

Usage should be something like this:

```
python local_alignment.py AATCG AACG 1 -1 -2
```

Output:

```
Local alignment score: 2.0
[[0. 0. 0. 0. 0.]
 [0. 1. 1. 0. 0.]
 [0. 1. 2. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 2.]]
Final alignment:
AA
||
AA
```



Assignment 3: Run BWA

With the following steps you will install the two tools on your laptop. If you encounter problems in the installation process, feel free to contact us to solve the problem.

Open your terminal (for MacOS and Linux users) or Ubuntu 18.04 LTS app (for Windows) and type:

```
conda activate Bioinfo_labs (or source activate Bioinfo_labs)

conda install -c bioconda bwa
```

Download reference files (chr10 and chr18) and mate_1.fq and mate_2.fq

In this lab we will use **real human data**. However, the human reference genome is 3 GB long and using it in its entirety requires a lot of computational efforts. So, for convenience, we will use only two chromosomes (chr10 and chr18), but the same reasoning can be extended on all DNA.

Download the two files from here:

ftp://ftp.ensembl.org/pub/release-92/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.chromosome.18.fa.gz

ftp://ftp.ensembl.org/pub/release-92/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna.chromosome.10.fa.gz

Assignment 3: Run BWA

Move the two files to your working folder and unzip them:

```
gzip -d *.fa.gz
```

Then, concatenate the two fasta files into one file named `reference_chr10_chr18.fa`

```
cat *.chromosome.*.fa > reference_chr10_chr18.fa
```

mate_1.fq and mate_2.fq are fastq files coming from the RNA-seq sequencing of human brain, and specifically of cerebral cortex. **The original experiment has produced more than 12 GB of data**!! Therefore, to allow you to experience a real aligner on real human data, the files have been truncated. Please, download them from the course website.

Assignment 3: Run BWA

1. Create a bwa index for the reference sequence (chr10 and chr18). This process requires some minutes to be performed. Remember that indexing must be performed just once, when you have a new sample from the same species to process you can start directly from step 2.

```
mkdir path_to_folder_where_BWA_index_will_be_created
```

```
bwa index -p path_to_folder_where_BWA_index_will_be_created
path_to_reference_sequence/reference_chr10_chr18.fa
```

2. Perform the alignment of mate_1.fq and mate_2.fq

```
bwa mem path_to_folder_where_BWA_index_has_been_created
path_to_reads_file/mate_1.fq path_to_reads_file/mate_2.fq >
path_to_place_results/results.sam
```

Assignment 4: Bash scripting

Now, create a bash scripts that allow a user to run BWA taking all the arguments necessary to run the script from the command line. In particular, make sure that the user is informed at each step about what is happening (use the echo command - e.g. echo "BWA aligner is currently running. This step could require time, be patient!").

Example:

```
sh run_bwa.sh genomeref mate_1 ate_2 outputdata
```

```
Output:
```

```
BWA initializing...
```

```
BWA aligner is currently running. This step could require time, be patient!
```

```
Alignment is finished! You can now check your results.
```

LAB2 – Take home message

- Global alignment is used to match strings
- Local alignment is used to define the optimal subset alignment
- Test BWA
- Implement automatic scripts in bash

Questions?

Remember:
no question is
stupid