Laboratory of Bioinformatics I

International Bologna Master in Bioinformatics

University of Bologna

Academic Year 2009/2010

Development of bio-sequence comparison tools in Python

Pietro Di Lena

dilena@cs.unibo.it

Overview

- This part of the course has to be intended as a tutorial for the development of Python code for:
  - dot-plot matrix computation
  - pair-wise alignment of protein sequences
    - optimal global alignment: Needleman-Wunsch algorithm (NW)
    - optimal local alignment: Smith-Waterman algorithm (SW)
    - statistical significance of the alignments
    - affine gap penalty variants of NW and SW
  - multiple-alignment of protein sequences
    - multiple-alignment with respect to a reference sequence
    - progressive methods:
      - UPGMA (Unweighted Pair Group Method with Arithmetic Mean)
      - WPGMA (Weighted Pair Group Method with Arithmetic Mean)
  - extraction of statistical information from multiple-alignments:
    - sequence profiles
    - substitution matrices

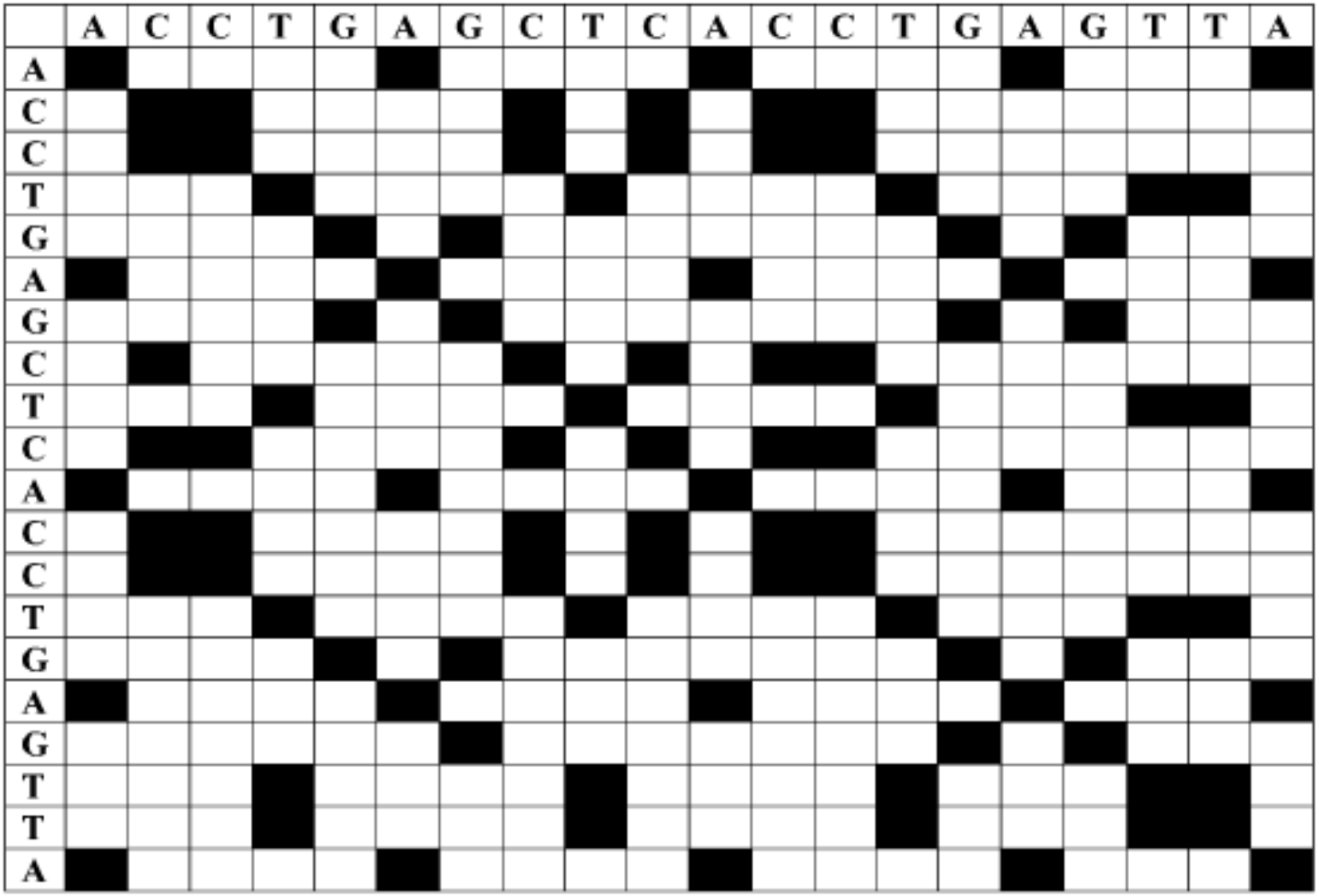
Dot-plot matrix

- Graphical method for the comparison of two biological sequences
- It allows the *visual* detection of regions of similarity between the two sequences
- It does not give an alignment between two sequences
- Given two sequences $A=a_1a_2\ldots a_n$ and $B=b_1\ldots b_m$, the k-blocks dot-plot matrix of A and B is defined by

$$M[i,j]=\sum_{l=0}^{k}\delta(a_{i+l},b_{j+l})\ 1\le i\le n-k, 1\le j\le m-k$$

where $\delta(a,b)$ is a similarity score between symbols a and b.

Example: k=1, identity similarity score



Exercises

1. Write a Python program

   dotplot.py <seq1> <seq2> <k>

   that computes the k-block dot-plot matrix of sequences seq1, seq2 (in FASTA format), using the identity similarity score:

$$\delta(a,b) = \begin{cases} 0 & if\ a \neq b \\ 1 & if\ a = b \end{cases}$$

2. Modify the program above so that it sets to 0 all the entries in the dot-plot matrix that have a score below some given threshold *t:*

   dotplot.py <seq1> <seq2> <k> <t>

3. Modify the program above so that it can eventually read the similarity scores from a 21x21 symmetric matrix (20 amino acids + 1 unknown symbol) stored in a file:

   dotplot.py <seq1> <seq2> <k> <t> [sim matrix]

Pairwise Alignment

- A *pairwise alignment* is an injective partial mapping

$$f : \{1,\ldots n\} \rightarrow \{1,\ldots m\}$$

between two sequences $A=a_1a_2\ldots a_n$ and $B=b_1\ldots b_m$

- An *optimal* pairwise alignment is the mapping that maximizes some well defined objective function:

$$\sum_{i=1, f(i)\neq\varnothing}^{n} \delta(a_i, b_{f(i)}) + \sum_{k\in G} \sigma(k)$$

- The objective function defines the costs for *match/mismatch* between symbols, i.e. the *similarity score* $\delta(a,b)$, and *insertions/deletions* of symbols, i.e. the *gap-penalty* score $\sigma(k)$, where $k \in G$

$$G = \{k \mid Exists\ a\ maximal\ gap\ of\ length\ k\ in\ f\}$$

Example

- Under the assumption that

$$\delta(a,b) = \begin{cases} 1 & if \ a = b \\ 0 & if \ a \neq b \end{cases} \quad and \quad \sigma(k) = -k$$

the alignment

```
FTFTALLLAFAV
F__TALMLA_AV
```

has score equal to 8-3=5 (the symbol _ denotes a gap)

- Is this an optimal alignment with respect to the defined scoring function and gap-penalty score?
- In general, given two sequences of lengths m and n, the number of possible alignments between the to sequences is

$$f(n,m) = \sum_{k=0}^{n} \binom{m+k}{n} \binom{n}{k} = \sum_{k=0}^{n} \frac{(m+k)!}{n!(m+k-n)!} \frac{n!}{k!(n-k)!} = \sum_{k=0}^{n} \frac{(m+k)!}{(m+k-n)!k!(n-k!)}$$

$$f(12,9) = 14.218.905$$

Optimal global alignment: Needleman-Wunsch algorithm

- Dynamic programming algorithm for computing the *optimal* *global* alignment between two sequences with respect to some similarity score and gap-penalty functions
- Here *global* means that the objective function is maximized with respect to the overall length of the two sequences
- The cost of the NW algorithm is on the order of O(nm), where n and m are the lengths of the two sequences

Implementation of the NW algorithm with linear gap-penalty function

- Assumption: $\sigma(k)=-dk$, for some constant d
- To find the optimal global alignment between two sequences of lengths n and m, two $(n+1)\text{x}(m+1)$ matrices, say M and P, are allocated and updated with the following recursive formulas

Initialization: for 1≤ i ≤ n, for 1≤ j ≤ m

$$M[0,0] = 0, M[0,j] = -dj, M[i,0] = -di$$

$$P[0,0] = DIAG, P[0,j] = LEFT, P[i,0] = UP$$

Iteration: for 1≤ i ≤ n, for 1≤ j ≤ m

$$M[i,j] = \max\begin{cases} M[i-1,j-1] + \delta(a_i,b_j) & case\ 1 \\ M[i-1,j] - d & case\ 2 \\ M[i,j-1] - d & case\ 3 \end{cases}$$

$$P[i,j] = \begin{cases} DIAG & if\ case\ 1 \\ UP & if\ case\ 2 \\ LEFT & if\ case\ 3 \end{cases}$$

Result: the optimal score can be found in the entry $M[n,m]$. The corresponding alignment can be recovered by tracing back from $P[n,m]$ up to $P[0,0]$.

Optimal local alignment: Smith-Waterman algorithm

- Dynamic programming algorithm for computing the *optimal local* alignment between two sequences with respect to some similarity score and gap-penalty functions.
- Here *local* means that the objective function is maximized with respect to arbitrary-length segments of the two sequences
- Necessary conditions to obtain a *proper* local alignment: the similarity score and the gap-penalty functions must eventually allow negative scoring
- The cost of the SW algorithm is on the order of O(nm), where n and m are the lengths of the two sequences

Implementation of the SW algorithm with linear gap-penalty function

- Assumption: $\sigma(k)=-dk$, for some constant d
- As in NW, we need two $(n+1)\text{x}(m+1)$ matrices

Initialization: for $1 \le i \le n$, for $1 \le j \le m$

$$M[0,j] = M[i,0] = 0$$

$$P[0,j] = P[i,0] = STOP$$

Iteration: for $1 \le i \le n$, for $1 \le j \le m$

$$M[i,j] = \max \begin{cases} 0 & case\ 0 \\ M[i-1,j-1] + \delta(a_i, b_j) & case\ 1 \\ M[i-1,j] - d & case\ 2 \\ M[i,j-1] - d & case\ 3 \end{cases}$$

$$P[i,j] = \begin{cases} STOP & if\ case\ 0 \\ DIAG & if\ case\ 1 \\ UP & if\ case\ 2 \\ LEFT & if\ case\ 3 \end{cases}$$

Result: the optimal local score can be found in the entry i,j such that $M[i,j]$ is maximal. The corresponding alignment can be recovered by tracing back from $P[i,j]$ up to the first entry STOP

Exercises

1. Implement in Python the NW/SW algorithms for optimal global/local pairwise alignment, given in input two FASTA sequences, the gap-penalty score and the similarity matrix:

   align.pl <seq1> <seq2> <gap penalty> <sim. matrix>

   The program must return the score of the alignment and the alignment itself. Example:

```
linux-term$ nw.py seq1 seq2 1 BLOSUM62
Score = 34
FTFTALLLAFAV
__FTALMLA_AV
```

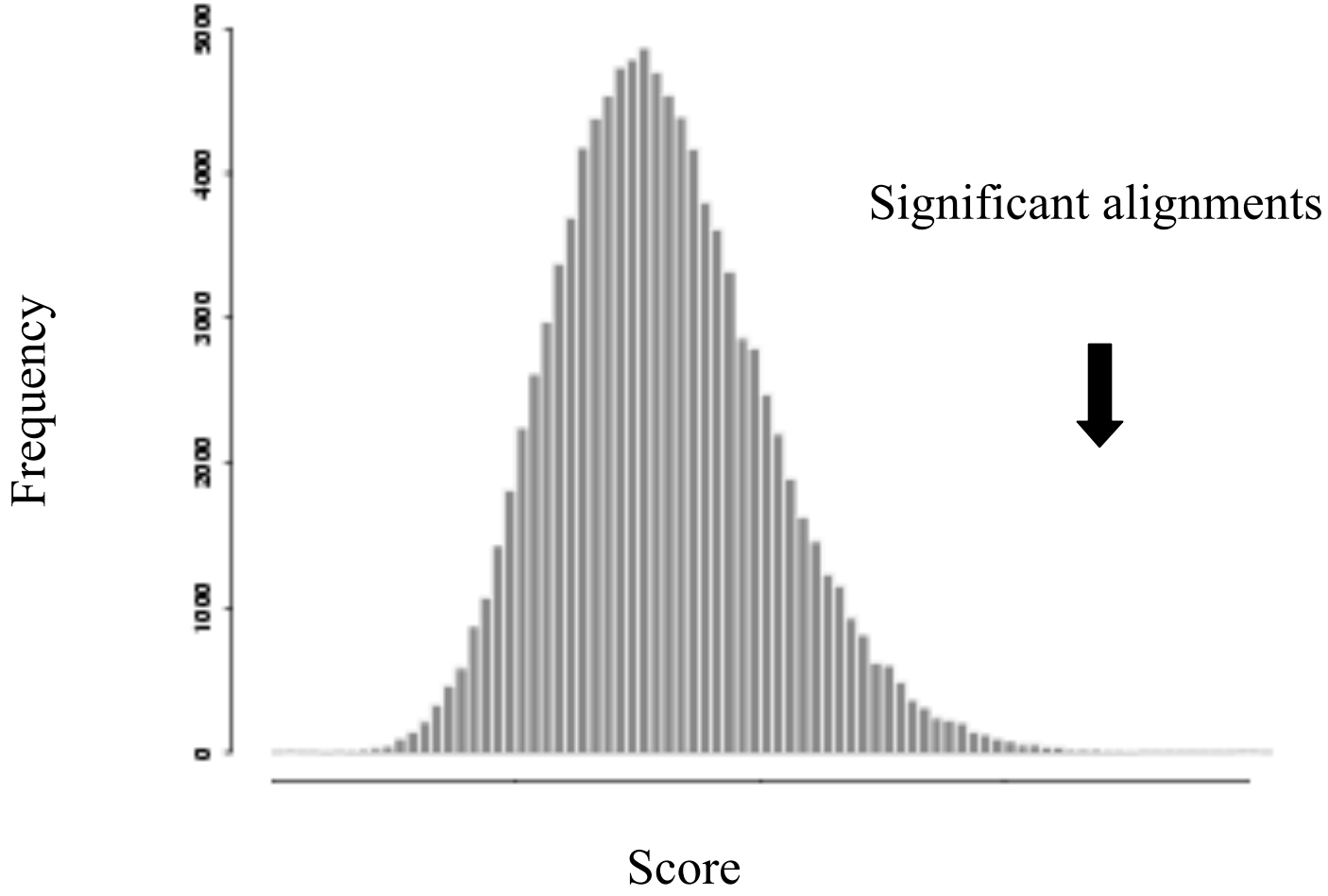
2. Enhance the output of the programs by providing additional information, such as length of the alignment and num. of gaps, identities, positive matchings (i.e. sim score > 0). Example:

```
linux-term$ nw.py seq1 seq2 1 BLOSUM62
Score = 34, Length = 12
Id = 8/12 (67%), Pos = 9/12 (75%), Gaps = 3/12 (25%)
FTFTALLLAFAV
  FTAL+LA AV
__FTALMLA_AV
```

Statistical significance of alignments

- How can we detect if the optimal score of an alignment is a reliable similarity measure between two sequences?
- Typical problem: assume that you want to search in a database of sequences the one that is most *similar* (w.r.t an optimal alignment) to some target sequence A. It is always reliable to conclude that the most similar sequence to A, among all sequences in the database, is the one that has the highest alignment score with A?
- The statistical significance of an alignment tells us how much the alignment scores are reliable.

Example: hypothetical score distributions between shuffled sequences



Z-score

- Assume that the score on an alignment between two sequences A,B is *s*. The Z-score of the alignment can be computed as

$$Z = \frac{s - Mean(S)}{StandardDeviation(S)}$$

  where S in a vector of scores obtained by iteratively shuffling A and by computing the score of the alignment between the shuffled A and B
- Note: the Z-score values depend on the length of the sequences and on the number of shuffling performed
- General assumptions (NOT always true):
  - $Z<3$ means not significant
  - $3<Z<10$ means probably significant
  - $Z>10$ means significant

Exercises

1. Write a procedure in python for shuffling strings.
2. Include the Z-score calculation in the NW and SW implementations. Choose a fixed number of shuffling (>100) to be performed (i.e. you do not have to pass the number of shuffling to the procedures).

Affine gap penalty

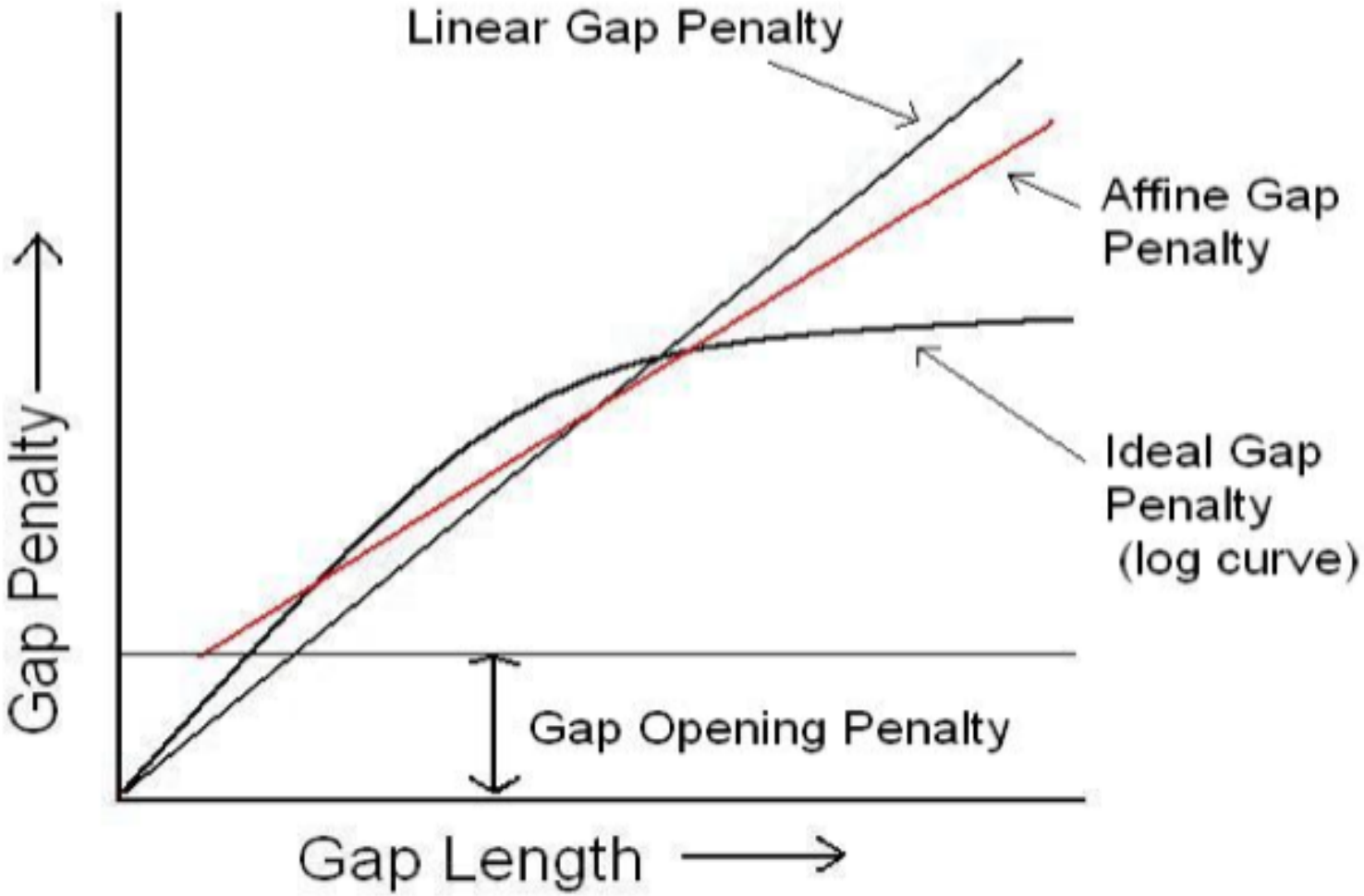
- A more accurate way (from the biological point of view) to account for gap penalties
- Example: assume we are using the identity similarity score and the linear gap penalty with d=1. The two alignments

```
TLLLLLLAF     TLLLLLLAF
_L_L_L_AF     ____LLLAF
```

  have exactly the same score (=1). Which alignment is intuitively better from a biological point of view?
- Insertions/deletions in protein sequences occur as *block* events with a number of residues inserted/deleted at a time. This phenomenon can be easily modeled by using two different gap penalty values:
  - o: gap opening penalty (high value)
  - e: gap extension penalty (much lower than o)
- The affine gap penalty finction score is formally defined as

$$\sigma(k) = -o - e(k-1)$$

Comparison of gap penalty functions



Implementation of the NW algorithm with affine gap penalty 1/3

- Assumption: σ(k)=-o-e(k-1), for some constants $o>>e$
- In order to maintain the computational cost of the alignment procedure on the order of O(mn), we need three different $(n+1)$x$(m+1)$ matrices, say M, A,B, for explicitly distinguish between three different kind of alignments:
  - $M[i,j]$ is the optimal score for an alignment between $a_1 \ldots a_i$ and $b_1 \ldots b_j$ that ends with no gaps
  - $A[i,j]$ is the optimal score for an alignment between $a_1 \ldots a_i$ and $b_1 \ldots b_j$ that ends with a gap aligned with a_i
  - $B[i,j]$ is the optimal score for an alignment between $a_1 \ldots a_i$ and $b_1 \ldots b_j$ that ends with a gap aligned with b_j

Initialization: for $1 \le i \le n$, for $1 \le j \le m$

$$M[0,0] = 0,\ M[0,j] = -\infty,\ M[i,0] = -\infty$$

$$A[0,0] = 0,\ A[0,j] = -\infty,\ A[i,0] = -o - e(i-1)$$

$$B[0,0] = 0,\ B[0,j] = -o - e(j-1),\ B[i,0] = -\infty$$

Note that $M[0,j]$ and $M[i,0]$ do not correspond to alignments that ends with no gaps. The same holds for $A[0,j]$ and $B[i,0]$.

Implementation of the NW algorithm with affine gap penalty 2/3

Iteration: for $1 \le i \le n$, for $1 \le j \le m$

$$M[i,j] = \max \begin{cases} M[i-1,j-1] + \delta(a_i, b_j) \\ A[i-1,j-1] + \delta(a_i, b_j) \\ B[-1i,j-1] + \delta(a_i, b_j) \end{cases}$$

$$A[i,j] = \max \begin{cases} M[i-1,j] - o & \textit{opening gap} \\ A[i-1,j] - e & \textit{extending gap} \\ B[i-1,j] - o & \textit{opening gap} \end{cases}$$

$$B[i,j] = \max \begin{cases} M[i,j-1] - o & \textit{opening gap} \\ A[i,j-1] - o & \textit{opening gap} \\ B[i-1,j] - e & \textit{extending gap} \end{cases}$$

Implementation of the NW algorithm with affine gap penalty 3/3

Result: the optimal alignment score is

$$\max\{M[n,m],A[n,m],B[n,m]\}$$

The traceback must be started from the matrix (among M,A,B) that has the highest value in position n,m. The path matrices related to M,A,B are updated as in the linear gap implementations with the following differences:

- the path matrix related to M can contain only DIAG values. A LEFT value corresponds to a jump into the A's path matrix and an UP value corresponds to a jump into the B's path matrix,
- the path matrix related to A can contain only LEFT values. A DIAG value corresponds to a jump into the M's path matrix and an UP value corresponds to a jump into the B's path matrix,
- the path matrix related to B can contain only UP values. A DIAG value corresponds to a jump into the M's path matrix and an UP value corresponds to a jump into the B's path matrix.

Exercises (optional)

- Implement the affine gap variants of the NW and SW algorithms for pairwise alignment of sequences

Multiple Sequence Alignment (MSA)

- Sequence alignment of three or more bio-sequences
- The sequences to be aligned are generally assumed to have the same ancestor and thus a common evolutionary relationship
- MSA can be used to extract statistical information about protein sequences, such as:
  - sequence conservation
  - sequence profile
  - residue similarity matrices
- An optimal MSA is the multiple alignment that maximizes the sum of the scores of each pairwise comparison of sequences in the MSA
- Differently from the pairwise alignment problem, the MSA in NP-hard, i.e. no optimal algorithms exist for finding the optimal MSA in reasonable time

Multiple alignment with respect to a reference sequence

- The simplest example of multiple sequence alignment (efficiently computable)
- Can be used to compute the *profile* of a target sequence (i.e. a table of the frequency of occurrence of each residue in each position of the target sequence)
- Given a reference sequence P and a set of (related) sequences D, the MSA is obtained by computing the parwise alignment (usually with a local alignment algorithm) of P with every P' in D and by writing each aligned sequence in row (maintaining the correspondence between aligned residues) with P on top. Example:

```
seq1 FTFTALLLAFAV (ref)
seq2 FTALMLAAV
seq3 PTFKALIIAF
```

```
local al. 1-2  FTALLLAFAV
               FTALMLA_AV
local al. 1-3  TFTALLLAF
               TFKALIIAF
```

```
FTFTALLLAFAV
__FTALMLA_AV
_TFKALIIAF__
```

Progressive multiple alignments

- The most widely used heuristic for computing MSA
- Given a set of (related) sequences, the final MSA is built by iteratively combining pairwise alignments and/or multiple alignments, beginning with the most similar sequences and progressing to the most distantly related
- The ordering of the alignments to be performed is defined by a guide-tree (dendogram), usually computed by some clustering algorithm from a (dis)similarity matrix of scores between each pair of sequences in the set
- In order to implement a multiple alignment with the progressive approach we need:
  - a hierarchical clustering algorithm for computing a dendogram from a set of (dis)similarity scores (e.g. UPGMA or WPGMA)
  - a *uniform* measure of (dis)similarity between pairs of protein sequences in a given set
  - an alignment procedure for aligning two MSA (usually a global alignment procedure)

UPGMA (Unweighted Pair Grouping Method with Arithmetic Mean)

UPGMA(D,C)

Input: A nxn dissimilarity matrix D and a cluster set C=$\{C_1,\ldots,C_n\}$

Main body:

if |C|=1 **return** C

else

1. **Selection**: select $1 \le i<j \le n$, such that $D[i,j]=\max(D)$
2. **Update**:
   1. Matrix D:
      - $\forall k \in \{1,\ldots,n\}, i \neq k \neq j$

$$D[k,i] = \frac{|C_i|}{|C_i \cup C_j|} D[k,i] + \frac{|C_j|}{|C_i \cup C_j|} D[k,j]$$

      - remove column and row j in D
   2. Cluster C:
      - C_i=$C_i \cup C_j$
      - remove cluster C_j from C
3. **Recursive call**: return UPGMA(D,C)

WPGMA (Weighted Pair Grouping Method with Arithmetic Mean)

- Same underlying algorithm of UPGMA
- The only difference is the updating function for the (dis)similarity matrix:

$$\forall k \in \{1,\ldots,n\}, i \neq k \neq j$$

$$D[k,i] = \frac{1}{2}\left(D[k,i] + D[k,j]\right)$$

A simple way to compute a progressive multiple alignment

- We need a scoring measure for computing the (dis)similarity matrix. For example: Z-score, % of identities/positives between the aligned sequences
- We need an algorithm for aligning two MSA. Idea: modify the NW/SW by just changing the similarity score function and the gap penalty value d. The function $\delta(a_i,b_j)$ that scores the similarity between the ith and the jth residue must now be changed in a function $\Delta(A_i,B_j)$ that scores the similarity between the ith column of the MSA A and the jth column of the MSA B. The gap penalty d must be changed in the cost of aligning a column in A/B with a column of gaps in B/A. Assume that A,B contain n and m sequences, respectively and that the ith and jth columns of A and B are $A_i=a_1\ldots a_n$ and $B_i=b_1\ldots b_m$, respectively. Then, a possible scoring function is:

$$\Delta(A_i,B_j)=\sum_{k=1}^{n}\sum_{l=1}^{m}\omega(a_k,b_l)$$

$$\omega(a,b)=\begin{cases}-d & \text{if either } a \text{ or } b \text{ is a gap}\\ 0 & \text{if both } a \text{ and } b \text{ are gaps}\\ \delta(a,b) & \text{otherwise}\end{cases}$$

Alignment of multiple alignments: NW modification

- Assumption: constant gap penalty
- To find the optimal global alignment between two MSA of lengths n and m, two $(n+1)\text{x}(m+1)$ matrices, say M and P, are allocated and updated with the following recursive formulas

Initialization: for 1≤ i ≤ n, for 1≤ j ≤ m

$$M[0,0] = 0, M[0,j] = M[0,j-1] + \Delta(g_A, B_j), M[i,0] = M[i-1,0] + \Delta(A_i, g_B)$$

$$P[0,0] = DIAG, P[0,j] = LEFT, P[i,0] = UP$$

Iteration: for 1≤ i ≤ n, for 1≤ j ≤ m

$$M[i,j] = \max\begin{cases} M[i-1,j-1] + \Delta(A_i, B_j) & case\ 1 \\ M[i-1,j] + \Delta(A_i, g_B) & case\ 2 \\ M[i,j-1] + \Delta(g_A, B_j) & case\ 3 \end{cases}$$

$$P[i,j] = \begin{cases} DIAG & if\ case\ 1 \\ UP & if\ case\ 2 \\ LEFT & if\ case\ 3 \end{cases}$$

The two vectors $\text{g}_\text{A}, \text{g}_\text{B}$ have length equal to the number of sequences in the MSA A and B, respectively, and contain only gaps.

Exercises

1. Implement in Python a procedure that, taken in input a reference sequence S and a set of sequences, computes the MSA with respect to the reference S, by using the SW algorithm.
2. Implement in Python a procedure that, taken in input a set of sequences, computes a progressive MSA, by using:
   - UPGMA/WPGMA clustering algorithms
   - Z-score for setting up the (dis)similarity matrix
   - Modified NW algorithm for computing the alignment of two MSA

Protein sequence profile calculation

- A compact representation of the evolutionary information related to a protein sequence
- The sequence profile of a protein of length n is formally defined as a $20\text{x}n$ matrix P, where each row of the matrix is associated to one of the 20 amino acid symbols. Usually, one more row is added to account for gaps
- Each element $P[i,j]$ represents the normalized frequency of the respective ith symbol (residue or gap) at the jth position in the protein sequence
- A sequence profile for a protein A can be easily computed from a MSA M, built by taking A as reference protein:

$$P_{ij} = \frac{number\ of\ \ symbols\ of\ \ type\ i\ in\ column\ \ j\ of\ \ M}{number\ of\ \ rows\ of\ \ M}$$

Example: profile

```
AAABABAABB
_AB_AABAB_
BABBA_AAB_
___BB_BAB_
```

	A	A	A	B	A	B	A	A	B	B
A	25	75	25	0	75	25	50	100	0	0
B	25	0	50	75	25	25	50	0	100	25
_	50	25	25	25	0	50	0	0	0	75

Substitution matrix calculation

- Describes how likely one symbol in a sequence changes into another symbol
- In protein sequence comparison, residue substitution matrices (e.g. BLOSUM, PAM) are used to score the alignment of evolutionary divergent sequences
- In a substitution matrix S, the probability of transformation of a symbol is generally expressed in terms of *log-odd* scores

$$S[i,j] = \frac{1}{\lambda}\log\left(\frac{p_{ij}}{p_i p_j}\right)$$

where p_i ,p_j are the probabilities of occurrence of symbols of type i,j, respectively, and p_{ij}=p_{ji} is the probability that a symbol of type i changes into a symbol of type j. The parameter λ is a *scaling factor,* which can be used to convert the matrix values into integers

- The probabilities p_i, p_j, p_{ij} can be extracted from MSA

Example: substitution matrix

```
AAABABAABB
_AB_AABAB_
BABBA_AAB_
___BB_BAB_
```

$$p_A = \frac{15}{30} = 0.5 \quad p_B = \frac{15}{30} = 0.5 \quad p_A p_A = p_B p_B = p_A p_B = \frac{1}{4} = 0.25$$

$$p_{AA} = \frac{13}{35} = 0.37 \quad p_{AB} = p_{BA} = \frac{11}{35} = 0.315 \quad p_{BB} = \frac{11}{35} = 0.315$$

	A	B
A	0.57	0.33
B	0.33	0.33

$\lambda = 3/100 = 0.03$ →

	A	B
A	19	11
B	11	11

Exercises

1. Implement in Python a program to compute the sequence profile from a given MSA:

   profile.py <msa>

2. Implement in Python a program to compute a substitution matrix with respect to a set of given symbols, from a set of given MSA:

   substmat.py <msa list> <symbols>

   In order to simplify the code, you can assume that <msa list> is a list of names of the files containing the MSA