Programming Lab I — Handout 4

Dr. Martin Vogt
martin.vogt@bit.uni-bonn.de

May 12, 2020

Deadline: May 31, 2020 **Next handout: May 19, 2020**

No handout: May 26, 2020

Sequence alignments

Theory. Sequence alignment is a widely used method for the comparison of two or more macromolecular sequences (e.g. DNA, RNA, proteins). Similar residues of the sequences are aligned, and gaps are introduced into the sequences where necessary. The quality of such a sequence alignment is assessed by means of a score. First, scores for every position of the alignment are determined and then summed up to yield the overall alignment score. The score for one position reflects the similarity of the aligned residues at this position; the more similar the residues, the higher the score. Hence, the overall alignment score measures the similarity of the aligned sequences. Very simple scoring schemes assign a positive score to a position if the aligned residues are of the same type and a constant negative score if they are distinct. This simplified approach would only be appropriate for nucleotide sequences. Substitution matrices present a more elaborate scoring approach: for every possible pair of residues, an individual score is determined. The following shows the BLOSUM62 substitution matrix.

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

The BLOSUM matrices (blocks substitution matrices) are a family of substitution matrices widely used for protein sequence alignment. The score for the alignment of two residues (amino acids) is derived from their evolutionary relatedness. The degree of relatedness is determined from a database of known related protein sequences. These related protein sequences are aligned to each other in gapless multiple sequence alignments called "blocks". Then, for each pair of amino acids, the frequency of their occurrence in the same position of a block is counted. If two amino acids are aligned frequently in the same position, they are considered to be highly similar.

BLOSUM scoring scheme. The BLOSUM score compares for each amino acid pair how often they actually co-occur in the same position of the blocks database with the statistical expectation for this frequency. The final score s_{ab} for two amino acids a and b is called the log odds ratio and is defined as follows:

$$s_{ab} = 2 \cdot \log_2 \left(\frac{p_{ab}}{e_{ab}} \right) = 2 \cdot \log_2 \left(\frac{\text{observed frequency of pair } ab}{\text{expected probability of pair } ab} \right) \tag{1}$$

Here, p_{ab} is the *observed frequency* for amino acids a and b to occur in the same position of the alignment database, whereas e_{ab} denotes the *expected frequency* for a and b to be aligned in the same position. If two amino acids occur in the same position more frequently than expected in the database of related protein sequences, the two amino acids are likely to be similar. Then, the log odds score is positive, and aligning these amino acids gives a positive contribution to the overall score of the alignment. However, if the substitution of two amino acids is observed less frequently than expected, the corresponding log odds ratio will be negative. Aligning such an amino acid pair will decrease the overall score of an alignment.

Calculation of the score. First of all, the absolute frequencies of each single amino acid (f_a) and each pair of aligned amino acids (f_{ab}) in the alignment database are counted and stored in a frequency table. Here, one does not distinguish between f_{ab} and f_{ba}. Then, the relative frequencies can be determined from that:

$$p_a = \frac{f_a}{\sum_i f_i} \tag{2}$$

where the sum is taken over all amino acids and

$$p_{ab} = \frac{f_{ab}}{\sum_{\{i,j\}} f_{ij}} = \frac{\text{frequency of pair } ab}{\text{sum of frequencies of all pairs}} \tag{3}$$

where the sum is taken over all unordered pairs of amino acids.

The next step is the computation of the expected occurrence of an aligned amino acid pair. This is computed as the statistical probability of co-occurrence of the two amino acids from their relative frequency in the database:

$$e_{ab} = p_a \cdot p_b + p_b \cdot p_a = 2 \cdot p_a \cdot p_b \text{ for two distinct amino acids} \tag{4}$$

and

$$e_{aa} = p_a \cdot p_a \text{ for an amino acid pair consisting of the same amino acid.} \tag{5}$$

From these values, the log odds score s_{ab} is computed. The score is given as an integer number. So the score s_{ab} should be **rounded to the nearest integer number**. [1]

Pairwise sequence alignment

In pairwise sequence alignment one tries to find a correspondence between the symbols of two (related) sequences taking into account that in the evolutionary process sequences may have changed by either

1. replacing amino acids with other amino acids,
2. inserting additional amino acids into a sequence,
3. deleting some amino acids from a sequence.

Because insertions and deletions can occur anywhere in the sequences and the total number of possibilities to do so increases exponentially, finding the optimal alignment between two sequences is not a trivial task. However, as you might know, the problem can be solved using a dynamic programming approach. In bioinformatics the algorithms for solving these problems are known as the Needleman-Wunsch and the Smith-Waterman algorithm. These algorithms are able to find an optimal alignment between two sequences given a scoring scheme that assigns costs (or scores) to any possible alignment between two symbols and costs/scores to inserting or deleting symbols. As we have seen above, scores assigned to certain alignments are derived from statistical considerations and form the mathematical basis for interpreting alignment scores. In order to quantify the quality of an alignment one also needs to assign a score to an insertion/deletion (indel). The most easy way to do this is to assign a fixed score to an inserted or deleted symbol known as the gap penalty w giving rise to the following formulation of the sequence alignment algorithms.

[1] **Further reading.** The original publication of the BLOSUM matrix method by Henikoff & Henikoff might be helpful for understanding the theory about substitution matrices. It is deposited as `blosum_paper.pdf` in the course folder. Another good introduction is presented in the Nature Biotechnology Primer on BLOSUM matrices, which you will find at the same location as `blosum_primer.pdf`.

Needleman-Wunsch Global Alignment Algorithm:

1. *Given:*
   (a) Two (DNA or amino-acid) sequences (x_i), (y_j) of length n and m.
   (b) A score matrix $s(x, y)$ giving the score value for matching symbols x and y.
   (c) A negative gap penalty score w for introducing gaps into the alignment.
2. *Construction of the Alignment Matrix M:*
   (a) *Initialize:* $M(0,0) = 0$, the first row $M(0, j) = jw$ for $j = 1 \dots m$, and the first column $M(i, 0) = iw$ for $i = 1 \dots n$
   (b) *Fill Matrix:*

$$M(i,j) = \max \begin{cases} M(i-1, j-1) + s(x_i, y_j), \\ M(i-1, j) + w, \\ M(i, j-1) + w \end{cases} \quad \text{for } i = 1 \dots n, j = 1 \dots m.$$

   (c) Keep pointer to indices yielding the maximum value in a traceback matrix $T(i, j)$ indicating whether we matched two symbols or introduced a gap in one of the sequences.
3. *Traceback Alignment:* Build alignment backwards starting at (n, m) by following the pointers in $T(i, j)$ and
   (a) aligning x_i and y_j if $T(i, j)$ points to $(i-1, j-1)$ (move along the diagonal),
   (b) aligning (y_j) to a gap if $T(i, j)$ points to $(i, j-1)$ (move 'left'),
   (c) aligning (x_i) to a gap if $T(i, j)$ points to $(i-1, j)$ (move 'up').

   Continue with $(i, j) \leftarrow T(i, j)$, i.e. the indices $T(i, j)$ pointed to.

   The alignment can be ambiguous. For your implementation you should prefer (mis-)matches over gaps.

   The traceback matrix keeps track, which of the three possible values to the left, up, or along the diagonal of the current cell (i, j) of the matrix yielded the maximum according to 2.(b). Instead of explicitly keeping track of the direction in a matrix as suggested here it is also possible to reconstruct the direction by repeating the calculations of 2.(b) during traceback and determining which calculation yielded the value of the current cell and move accordingly. This avoids the necessity of an explicit traceback matrix and trades off memory consumption for a loss in speed.

Local Sequence Alignment

The Needleman-Wunsch algorithm tries to find an optimal alignment between two sequences by trying to align all symbols of the two sequences to each other. However, frequently either one sequence will only match well to a small portion of the other sequence, or only a part of each sequence is related to each other and matches well. The global alignment algorithm does not take this into account. However the algorithm can be easily adapted to account for this situation. This yields the well known Smith-Waterman algorithm for local sequence alignment:

Modifications for Smith-Waterman Local Alignment Algorithm:

1. Mismatches must be scored negatively (so that a random alignment will have a negative expected score. Otherwise long alignments of unrelated sequences will yield bigger scores).
2. While filling the matrix all negative values are replaced with 0, that is

$$M(i,j) = \max \begin{cases} M(i-1, j-1) + s(x_i, y_j), \\ M(i-1, j) + w, \\ M(i, j-1) + w, \\ 0. \end{cases}$$

3. Traceback does not start at (n,m) but at the highest value of $M(i,j)$ and stops when an entry of the matrix with value 0 is encountered.

Affine gap penalties

A constant gap penalty is not optimal for sequence alignment. In evolution it is generally 'hard' to insert symbol(s) at any given position of a sequence, however it is not much harder to insert multiple symbols during a mutation than a single symbol. The same goes for removed symbols. In this light, the penalty for a gap w should actually depend on the length of the gap l making it a function $w(l)$.
Can the recursive equation be modified?
Note that for a gap of length 1, in the dynamic programming matrix you need to look 1 entry to the left: $M(i-1,j)+w(1)$. For longer gap penalties you need to also look further to the left: $M(i-l,j)+w(l)$ depending on how many symbols in the first sequence are to be assigned to a gap. Again, the highest score wins. Taking the original global sequence alignment as a template we only need to modify the initialization and the recursive entry for the determination of $M(i,j)$.

Needleman-Wunsch Global Alignment Algorithm with arbitrary gap penalty:

1. *Given:*
   (a) Two (DNA or amino-acid) sequences (x_i), (y_j) of length n and m.
   (b) A score matrix $s(x,y)$ giving the score value for matching symbols x and y.
   (c) A gap penalty function $w(l)$ for introducing gaps of length l into the alignment.
2. *Construction of the Alignment Matrix M:*
   (a) *Initialize:* $M(0,0)=0$, $M(0,j)=w(l)$ for $j=1\ldots m$, $M(i,0)=w(l)$ for $i=1\ldots n$
   (b) *Fill Matrix:*
$$M(i,j)=\max\begin{cases} M(i-1,j-1)+s(x_i,y_i), \\ \max\left\{M(i-l,j)+w(l)|1\leq l\leq i\right\}, \\ \max\left\{M(i,j-l)+w(l)|1\leq l\leq j\right\}, \end{cases} \quad \text{for } i=1\ldots n, j=1\ldots m.$$
   (c) Keep pointer to indices yielding the maximum value in a traceback matrix $T(i,j)$ indicating whether we matched two symbols or introduced a gap of a given length in one of the sequences.
3. *Traceback Alignment:* Build alignment backwards starting at (n,m) by following the pointers in $T(i,j)$ and
   (a) aligning x_i and y_j if $T(i,j)$ points to $(i-1,j-1)$,
   (b) aligning $(y_{j-l+1}\ldots y_j)$ to l gaps if $T(i,j)$ points to $(i,j-l)$,
   (c) aligning $(x_{i-l+1}\ldots x_i)$ to l gaps if $T(i,j)$ points to $(i-l,j)$.

   Continue with $(i,j)\leftarrow T(i,j)$, i.e. the indices $T(i,j)$ pointed to. The alignment can be ambiguous. For your implementation you should prefer (mis-)matches over gaps.

Note, that the major change over the algorithm with constant gap penalty occurs in the updating of $M(i,j)$. Here, it is not sufficient to just look to the next entry to the left/above but instead one needs to consider all the entries to the left and above $M(i,j)$ to determine $M(i,j)$. This will in fact increase the running time from $O(n^2)$ to $O(n^3)$ for sequences of length n.

The gap penalty function is frequently a simple affine function: $w(l)=d+e(l-1)$ consisting of a gap open penalty d and a gap extension penalty e where the absolute value of the extension penalty e is much smaller than that of the opening penalty d. In this case it is possible to reduce the computational complexity again to $O(n^2)$, by keeping track whether a gap is extended or opened at each point. This requires an additional matrix for keeping track of these things. More details can be found, for instance, in R. Durbin, S. Eddy, A. Krogh, G. Mitchison. *Biological sequence analysis.* Cambridge University Press, 1998.

Programming exercises

Ex.1 *(0 pts) Manually determining scoring matrices*

To test your understanding, consider this small example of a "blocks alignment database" consisting of 5 sequences of length 12 each. From these to numbers one can determinate the denominators in eqs. (2) and (3). What is $\sum_i f_i$ and $\sum_{\{i,j\}} f_{ij}$?

```
TSVKTYAKFVTH
TSVKTYAKFSTH
TSVKTYAKFVTH
LSVKKYPKYVVQ
SSVKKYPKYSVL
```

Count the frequencies f_a for all amino acids in the alignment and f_{ab} for all amino acid pairs occurring in the same column of the alignment. (For the pairs, do not consider the order of the amino acids: do not distinguish between `VS` and `SV`, for example). From these values, calculate the relative frequencies p_a for each occurring amino acid and p_{ab} for each occurring amino acid pair. Finally, calculate the expected probability and the score for each amino acid pair. Fill your results into the given tables.

	T	L	S	V	K	Y	A	P	F	H	Q
f_a											
p_a											

	TT	TL	TS	LS	SS	VV	KK	TK	YY	AA
f_{ab}										
p_{ab}										
e_{ab}										
s_{ab}										

	AP	PP	FF	FY	VS	TV	HH	HQ	HL	QL
f_{ab}										
p_{ab}										
e_{ab}										
s_{ab}										

Ex.2 *(7 pts) Calculating scoring matrices*

Write a program that computes a scoring matrix score from a given block alignment database. You should be able to call your problem like `python blosum.py alignment.dat blosum_matrix.out` from the command line. The input file `alignment.dat` (found in the course folder) contains a small alignment "database". It contains a number of aligned sequences of equal length with no gaps with one sequence per line.Your program should compute the substitution matrix from this alignment. The output file `blosum_matrix.out` should contain an output of the matrix like in the example of the BLOSUM matrix given above. (The original BLOSUM62 matrix is provided as `blosum62.txt` in the group folder. Are the scores of the matrix you have calculated for `alignment.dat` similar?)

Your program will need to.

(a) read in the alignment data in an appropriate data structure,

(b) determine the log-odds scores for each possible alignment of amino acids,

(c) produce a (nicely formatted) output of the resulting scoring matrix.

Note: You can use your code on the example from the previous exercise and compare the values of f_{ab}, p_{ab}, e_{ab} and s_{ab} to check whether your program works as expected.

Note: The "blocks alignment database" given isn't very large and it might happen that alignments between some pairs of amino acids do not occur at all, i.e., they will have a count of 0. You will run into numerical difficulties if you try to take the logarithm as this would yield a score of $-\infty$. To avoid this, set the score to a large negative number (e.g. -99). However it is even better to start counting all the frequencies from 1 and not from 0 thereby pretending that there is one additional alignment of each amino acid pair by default. This or a similar strategy is frequently used when calculating frequencies/probabilities from a limited amount of data. The additional counts are also known as (Laplace) pseudocounts.

Ex.3 *(13 pts) Global/local alignment*

(a) *(5 pts)* Implement the Needleman-Wunsch algorithm: Write a program called `needle.py` that takes 4 parameters and should be called like `python needle.py -8 blosum.txt sequence1.fasta sequence2.fasta`, where `-8` is the gap penalty w, `blosum.txt` is a text file containing the scoring matrix and `sequence1.fasta` and `sequence2.fasta` are two amino acid sequences in Fasta format.

The program should output the score of the alignment and the aligned sequences, i.e. the sequences plus the gaps (represented by the symbol `-`).

- Note that you will need to write a function that is able to read in a scoring matrix from a text file and store it in a suitable data structure.
- Test your program using the scoring matrix `blosum62.dat` and a gap penalty of -5 on the sequences
`THRQATWQPPLERMANGRQVE` and
`RAYMQNDLVKVRYYACHT`

(b) *(2 pts) Formatted output.* Modifiy the output of the alignment so that sequence 1 appears in the first line and sequence 2 in the third line while the second line contains symbols reflecting the similarity between aligned symbols: `|` for identical residues, `:` for conservative substitutions (positive score in scoring matrix) and spaces otherwise. E.g.:

```
ELVISISALIVE
  |: ||    :
PAVL-ISDEA-D
```

Break the output into multiple lines of about 80 characters so that alignment of long sequences is displayed properly.

(c) *(2 pts) Testing.* Perform all pairwise sequence alignments between sequences `RNAS1_horse.fasta`, `RNAS1_minke-whale.fasta`, and `RNAS1_red-kangaroo.fasta` using `blosum50.dat` and a gap penalty of -8. What can you conclude about the pairwise relationships?

(d) *(2 pt) Smith-Waterman.* Implement the Smith-Waterman algorithm by modifying the code from `needle.py`. In addition to the alignment score you should output the sequence similarity and the sequence identity of the two aligned sequences in percent. Sequence identity is the percentage of matching residues relative to the length of the aligned sequences including gaps, sequence similarity is the percentage of matched similar residues (i.e., those with a positive value in the scoring matrix) relative to the length of the aligned sequences including gaps.

(e) *(2 pt) Testing.* Run a global and a local sequence alignment on the sequences `halodurans.fasta` and `lentus.fasta` using `blosum62.txt` and a gap penalty of -8. Which is more appropriate?

Ex.4 *(Optional: 6 pts) Affine gaps*

(a) *(2 pts)* Modify the global sequence alignment implementation to allow affine gap penalties $w(l) = d+e(l-1)$ according to the algorithm described above. I.e., in each step calculate the match/mismatch score and the scores for introducing gaps of varying length and choose the best score.[2] Note that your program needs to accept an additional parameter for the gap open and extension penalty, e.g. `python needle.py -8 -2 blosum.txt sequence1.fasta sequence2.fasta` for $d = -8$ and $e = -2$.

(b) *(2 pts) Testing.* Perform the sequence alignment for sequences `GLB7A_CHITH.fasta` and `GLBE_CHITH.fasta` using `blosum62.txt` a gap open penalty of -8 and an gap extension penalty of -2. Compare this to the version using a constant gap penalty. To check your solution: In the affine case you should get a gap of length 6 that corresponds to the word `ALIGNE` in the other sequence.

The relevant files can be found in the folder `handout-04`.

[2]Note, that for sequences of length n this implementation has a running time of $O(n^3)$ compared to $O(n^2)$ for the linear gap penalty. For affine penalties a faster $O(n^2)$ algorithm exists, but it is more complex and requires three scoring matrices. (R.Durbin, S.Eddy, A.Krogh, G.Mitchison. Bioogical sequence analysis. *Cambridge University Press* 1998)