# Sequence Bioinformatics: Assignment 02

Tobias Fehrenbach and Ignacio García Ribelles

October 2021

The help function of the global aligner program we wrote for this assignment can be displayed from the command line by typing:

*>python3 global_aligner_Garcia_Fehrenbach.py –help*

The program was implemented to take three command-line options –mode, –stat and the file path of the input FastA-file as input from the user. The FastA-file must contain the two sequences to be compared separated and preceded by a header. The mode can be eiter 0, 1 or 2: This will specify which algorithm is used to solve an optimal global alignment (as specified in the following tasks) with the default mode being 0. The –stat option is optional: when set, the program prints out the memory usage of the algorithm, as well as the run-time it needed.

## 1 Needleman-Wunsch basic implementation

The first mode (- -mode 0) corresponded to the original Needleman-Wunsch algorithm using quadratic space in the form of a dynamic programming (n+1)*(m+1) matrix. With n being the length of the first sequence and m that of the second. This matrix was filled recursively depending on both the letters of the two corresponding sequence positions and the scores of the preceding top, left and diagonal cells. If the two letters matched, a score of +1 was added, else a gap-score or mismatch score of the same size was subtracted (-1). Once the matrix was filled, an optimal global alignment was traced back starting from the last cell (corresponding to the optimal score) and following every step that lead to that score. Thus ending at the top left corner. The optimal score and alignment are then printed out to the terminal.

## 2 Needleman-Wunsch with linear space

The second mode (- -mode 1) corresponded to an implementation of the Hirschberg algorithm (Hirschberg 1975). This algorithm combined the divide-and-conquer and dynamic programming principles resulting in a linear space variant of the Needleman-Wunsch algorithm. The idea was to divide the DP-matrix by two on the n-side(x-sequence), rounding to the next integer and thus obtaining an ith column corresponding to $\frac{n}{2}$. All cells of that column were calculated recursively. The jth row was then selected by finding a "cut" that maximized the score of an optimal alignment path up to the ith column. This had to be computed twice for the two recursions: one starting at the top left corner up to (i,j) and another starting at (i+1,j+1) and ending on the bottom right corner of the DP-matrix. The first recursion was defined as F for forward and the second one as B for backward, even though they both used the same function, corresponding to the NW-algorithm. The perk of this modification is that you save memory by storing only linear arrays that get shorter and shorter. The similarity score is then obtained by adding the (i,j) and the last scores. Due to an error in our code implementation, the obtained similarity score and the corresponding alignment were not optimal, as they were always smaller than that of the mode 0 algorithm. They were expected to yield the same similarity score.

# 3 Needleman-Wunsch, no table

The third mode (–mode 2) was an implementation of the Needleman-Wunsch algorithm without using any form of matrix. The calculation (computeF(x,y)) of each entry was done following the given pseudo code, then we looped through each combination of each position of the two given sequences. In this loop, the computeF function was called for each combination. However we couldn't figure out a way to trace-back the optimal alignment, therefore we only get the score of the alignment. The main drawback of this implementation is the excessive run-time needed to compute every single score recursively. For this reason we needed to create an even shorter FastA-file to test the algorithm, which also took very long to compute. The three given .fasta files were too large to be taken as input and even the shortest file took longer than 1 hour to compute and was thus not computed.

# 4 Comparison

The resulting optimal score and alignment of the NW-algorithm is in our opinion the only ones that were really optimal, due to errors in the implementation. We expected all of the algorithms to output the exact same similarity score for the same three given fasta files. We obtained similarity scores of 24 for the short.fasta, 79 for the medium.fasta and 635 for the long.fasta. The corresponding optimal alignments were not expected to be exactly the same, but all of the three modes should output the same optimal score for the same sequences, given enough time.

Even though we did not implement the Hirschberg algorithm correctly, we did include the –stat command, that let us measure the amount of memory and run-time used for each of the three FastA-files: short, medium and long, using the tracemalloc package. The peak memory usage (figure 1) and the run-time (figure 2) for the three modes can be seen below in the two figures, as well as in table 1. To be noted is that the NW without table graph is empty due to the inability of this algorithm to compute sequences longer than 10 bases long.

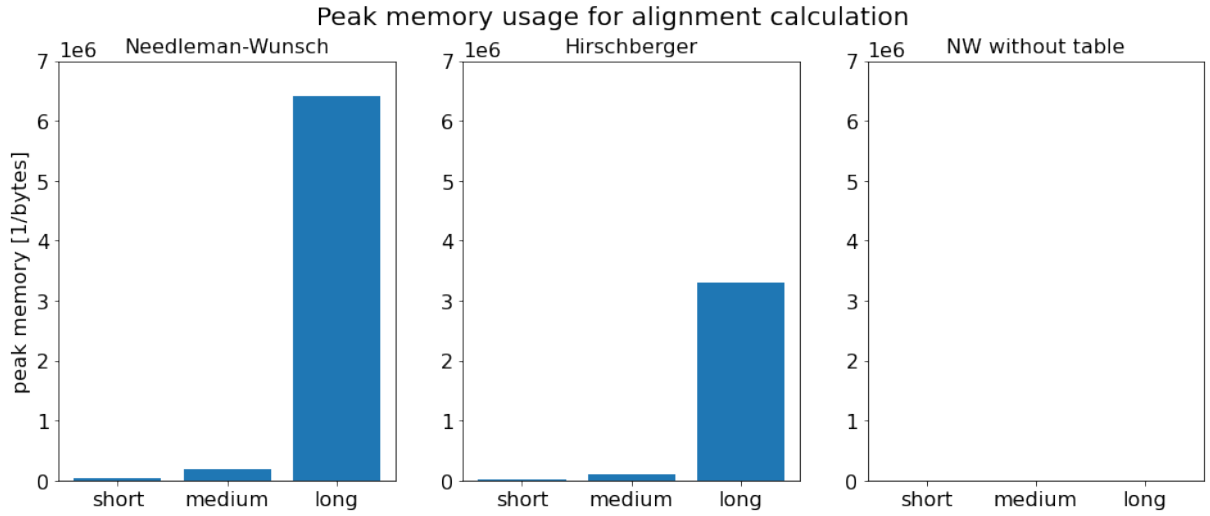


Figure 1: Plot of the peak memory usage for the alignment calculation depending on the sequence length for each algorithm.

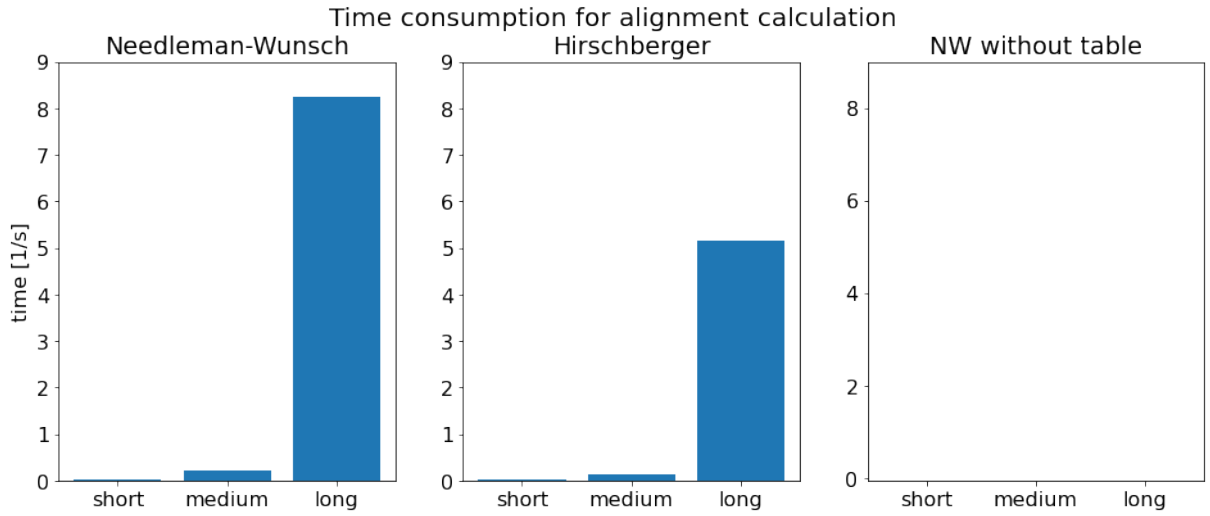


Figure 2: Plot of the time needed for the alignment calculation depending on the sequence length for each algorithm.

Table 1: Peak memory usage and Run-time for all three modes (mode 0: Needleman Wunsch, mode 1: Hirschberg algorithm and mode 2: Global alignment without table) and three files (short: ∼50, medium: ∼150 and long: ∼900).

| | peak memory usage in kbytes | | | run-time in s | | |
|---|---|---|---|---|---|---|
| | short | medium | long | short | medium | long |
| Needleman Wunsch | 63.7 | 135.6 | 6409.8 | 0.028 | 0.210 | 8.235 |
| Hirschberg | 26.4 | 113.2 | 3307.5 | 0.024 | 0.128 | 5.154 |
| Global alignment without table | < | < | < | >> | >> | >> |

As can be seen in the two graphs and the table, the peak memory and run-time increased consistently as the sequence length increased from short to long across all the three modes. Also to be noted is the fact that the Hirschberg algorithm took consistently less run-time and also yielded a smaller peak memory. The reduced peak memory was expected as the aim of this modification was to reduce the space from quadratic to linear. This reduction was also expected for the global alignment without dynamic programming, as its space is further reduced, due to the lack of a DP-matrix having to be stored. Nonetheless, the opposite was true for the run-time which was clearly much larger than the other two algorithms, to the point that none of the attempts to compute a similarity score for the three files was successful.

# 5 References

D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, Communications of the ACM, Volume 18, Issue 6, June 1975, pp 341–343, DOI: 10.1145/360825.360861