Report for LAB2: Sequence Analysis

Bioinformatics

Ricardo Brancas 83557 | Margarida Ferreira 80832 | Felipe Gorostiaga 95383 | Benedict Schubert 95034

25th October 2019

Group I. Needleman–Wunsch Algorithm

Needleman–Wunsch is a global sequence alignment algorithm that uses dynamic programming. Consider as inputs two sequences of nucleotides, s and p, of lengths N and M, respectively. We define a table, F, of size $N+1$ by $M+1$, as exemplified in figure 1). In this table each position is filled as follows:

$$\begin{aligned}
&\mathrm{F}(i,0) = -i * d && ,\forall i \in [0,N] \\
&\mathrm{F}(0,j) = -j * d && ,\forall j \in [0,M] \\
&\mathrm{F}(i,j) = \max \begin{cases} \mathrm{F}(i-1,j) - d \\ \mathrm{F}(i,j-1) - d \\ \mathrm{F}(i-1,j-1) + \mathrm{score}(s_i, p_j) \end{cases} && ,\forall i \in [1,N].\forall j \in [1,M]
\end{aligned}$$

Where score is a function that receives two nucleotides and returns a value that corresponds to their matching.

In the case of this exercise, we have $d = 4$ and $\mathrm{score}(a,b) = \begin{cases} -2 & \text{if } a \neq b \\ 3 & \text{if } a = b \end{cases}$.

The solution is then obtained by starting at the (N,M) position of the matrix and tracing back the positions that were used to obtain that value.

In figure 1 we present the table obtained when executing the Needleman–Wunsch algorithm over the strings `GGATCC` and `GGCCG`, with the optimal tracebacks highlighted. It's easy to see that there are two optimal alignments, shown in figure 2.

Group II. Smith–Waterman Algorithm

The Smith–Waterman algorithm performs local sequence alignment, by determining similar regions between two strings of proteins or nucleic acids. Instead of looking at the

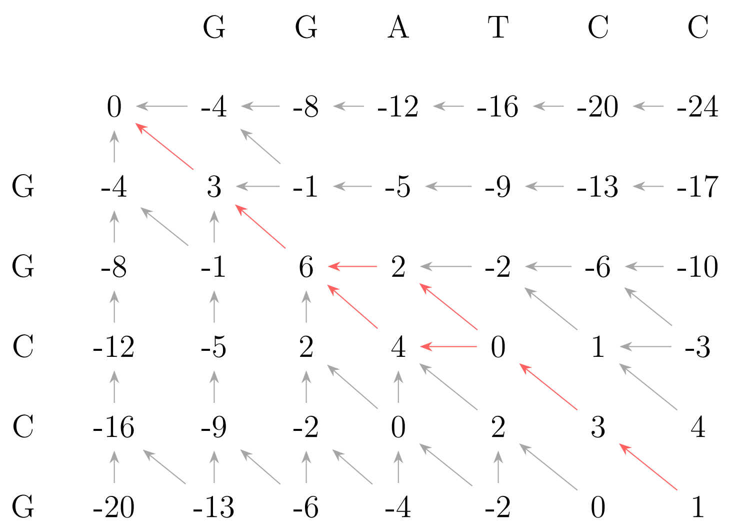


Figure 1: Result of the execution of the Needleman–Wunsch algorithm.

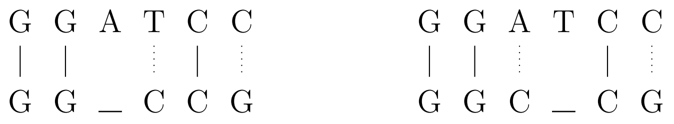


Figure 2: Optimal alignments for figure 1.

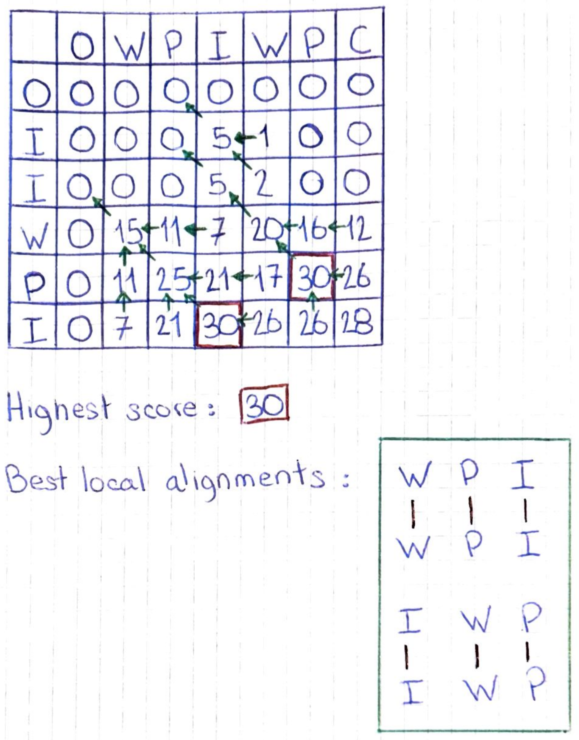
entire sequence, it compares segments of all possible lengths and optimises the similarity measure.

Like the Needleman–Wunsch algorithm, Smith–Waterman is a dynamic programming algorithm. The main difference to the Needleman–Wunsch algorithm is that negative scoring matrix cells are set to zero, which renders the (thus positively scoring) local alignments visible.

Consider as inputs two strings, s and p, of lengths N and M, respectively. Like in Needleman–Wunsch, we define a table, F, of size $N+1$ by $M+1$, filled as follows:

$$\begin{aligned}
&\mathrm{F}(i,0) = 0 && ,\forall i \in [0,N] \\
&\mathrm{F}(0,j) = 0 && ,\forall j \in [0,M] \\
&\mathrm{F}(i,j) = \max \begin{cases} 0 \\ \mathrm{F}(i-1,j) - d \\ \mathrm{F}(i,j-1) - d \\ \mathrm{F}(i-1,j-1) + \mathrm{score}(s_i,p_j) \end{cases} && ,\forall i \in [1,N].\forall j \in [1,M]
\end{aligned}$$

The solution is then obtained by finding the positions of the matrix containing the maximum value, and from those tracing back to the positions that were used to obtain that value.



	0	W	P	I	W	P	C
0	0	0	0	0	0	0	0
I	0	0	0	5	1	0	0
I	0	0	0	5	2	0	0
W	0	15	11	7	20	16	12
P	0	11	25	21	17	30	26
I	0	7	21	30	26	26	28

Highest score : $\boxed{30}$

Best local alignments :

```
W P I
| | |
W P I

I W P
| | |
I W P
```

Figure 3: Optimal alignments of strings S1: `WPIWPC` and S2: `IIWPI` computed using the Smith–Waterman algorithm.

Group III. Smith-Waterman implementation

The file `smith_waterman.py` contains Python source code that implements the Smith–Waterman algorithm. The program computes all the optimal local alignments between two strings (S1 and S2) using the BLOSUM50 scoring matrix. S1, S2 and the linear gap function are passed as command-line arguments as follows:

```
$ python3 smith_waterman.py  [S1] [S2] [lin_gap]
```

where S1 and S2 are strings of characters and linear gap is an integer.

The program outputs the table produced by the algorithm, all the optimal local alignments and their total score. With the input S1 = `WPIWPC` , S2 = `IIWPI` and linear gap penalty = 4, the program produces the following output.

```
$ python3 smith_waterman.py  WPIWPC IIWPI 4
  Smith—Waterman Algorithms — find all optimal local alignments
  S1: WPIWPC
  S2: IIWPI
  Gap penalty: 4

           W    P    I    W    P    C
      0    0    0    0    0    0    0
  I   0    0    0    5    1    0    0
  I   0    0    0    5    2    0    0
  W   0   15   11    7   20   16   12
  P   0   11   25   21   17   30   26
  I   0    7   21   30   26   26   28

  Score of optimal alignment(s): 30
  Optimal alignment(s):
  S1: IWP
      |||
  S2: IWP
  S1: WPI
      |||
  S2: WPI
```

The algorithm was run with protein sequences with size varying from 10 to 1000 amino-acids, with step 10. 18 sequences were run of each size, amounting to a total of 1800 runs. The resulting runtimes are presented in the graph of figure 4. A quadratic regression made on the obtained data allowed us to conclude that the time complexity of the Smith-Waterman algorithm is in fact quadratic in the length of the strings.

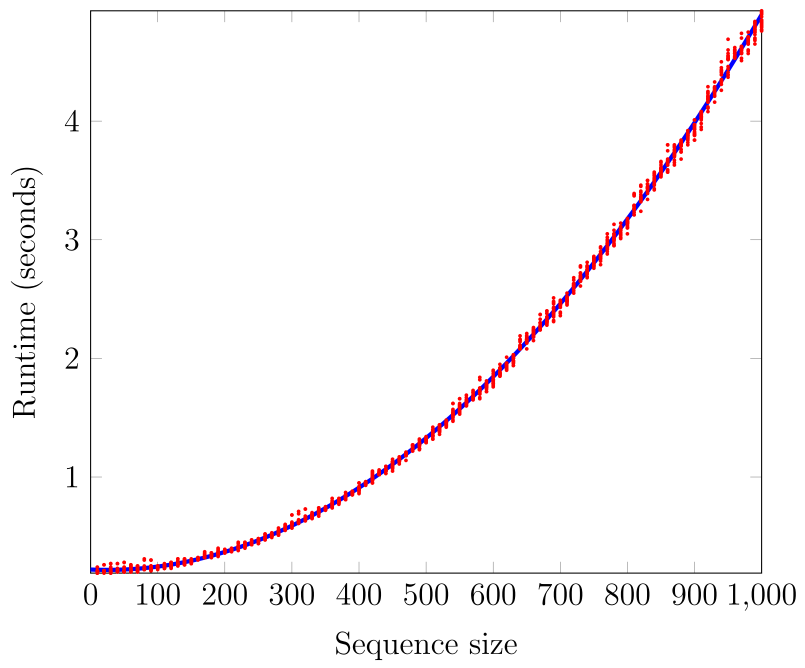


Figure 4: Runtime of 1800 pairs of protein sequences. In red, the runtime of individal instances, in blue, a quadractic regression $0.22 - 2.43 \cdot 10^{-4} \cdot x + 4.92 \cdot 10^{-6} \cdot x^2$. The R^2 value for this regression is 0.999, which means that 99.9% of the variability of the data around is explained by the regression.

Group IV. Multiple Sequence Alignment

In Multiple Sequence Alignment (MSA) we try to align more than two sequences, revealing subtle similarities not observable with pairwise alignment. The results can be used to infer sequence homology (does a new sequence belong to a specific family?) and conduct phylogenetic analysis in order to assess the sequences' shared evolutionary origins.

There are multiple methods to compute MSAs. Exact MSA methods are guaranteed to find optimal solutions, and use dynamic programming in a similar way as the methods presented above. Instead of a 2D matrix like described above, to align k strings, a k-D matrix is required. To align 3 strings, a 3D matrix is filled as follows:

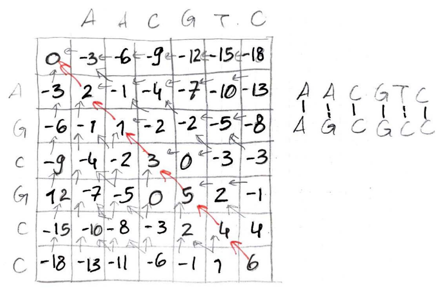
$$
\begin{aligned}
&\mathrm{F}(i,0) = 0 && ,\forall i \in [0,N] \\
&\mathrm{F}(0,j) = 0 && ,\forall j \in [0,M] \\
&\mathrm{F}(i,j) = \max \begin{cases}
\mathrm{F}(i-1,j-1,k-1) + \mathrm{score}(v_i, w_j, u_k) \\
\mathrm{F}(i-1,j-1,k) + \mathrm{score}(v_i, w_j, -) \\
\mathrm{F}(i-1,j,k-1) + \mathrm{score}(v_i, -, u_k) \\
\mathrm{F}(i,j-1,k-1) + \mathrm{score}(-, w_j, u_k) \\
\mathrm{F}(i-1,j,k) + \mathrm{score}(v_i, -, -) \\
\mathrm{F}(i,j-1,k) + \mathrm{score}(-, w_j, -) \\
\mathrm{F}(i,j,k-1) + \mathrm{score}(-, -, u_k)
\end{cases} && ,\forall i \in [1,N].\forall j \in [1,M]
\end{aligned}
$$

This method's complexity is exponential in the number of strings being aligned $O(2^k n^k)$.

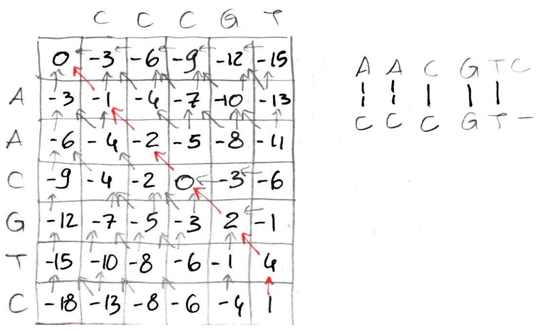
In some cases, progressive methods may also be used to optimally align more than 2 strings. Instead of trying to align all of them at once, we try to align all the pairs. If all optimal pairwise alignments are consistent, it is then possible to construct an optimal multiple alignment. This method, although not complete (does not always find a solution), is less complex: $\frac{k(k-1)}{2}$ pairwise alignments of strings of length n solved with Needleman-Wunsch amount to a total complexity of $O(n^2k^2)$.

We used this algorithm to compute the optimal alignment of the strings S1: `AACGTC`, S2: `AGCGCC`, S3: `CCCGT` and S4: `ACAT`.

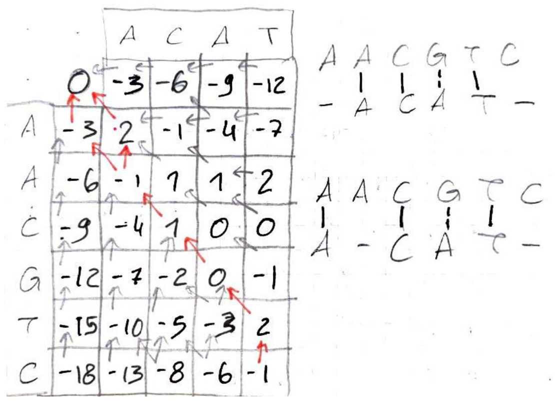
First, all the optimal pairwise alignments were computed using the Needleman–Wunsch algorithm (figure 5). Resulting from the alignments we can construct the distance matrix (figure 8). After extracting the closest sequences based on the matrix we can build the guide tree (figure 7). Then, since it is possible to choose a set of these alignments such that they are consistent, we can trivially construct an optimal multiple alignment (figure 6).



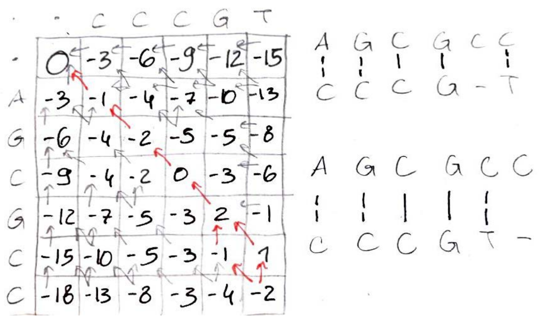
(a) Alignment of strings S1 and S2.



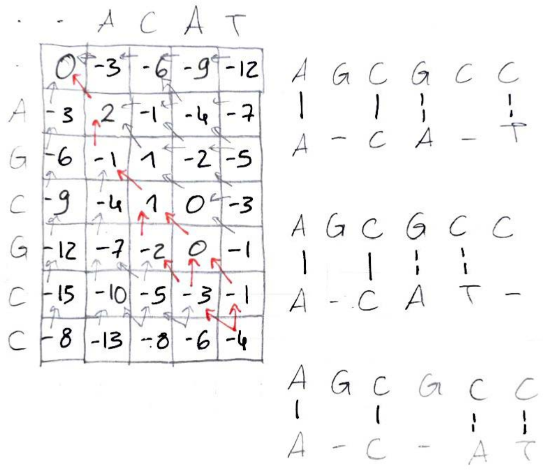
(b) Alignment of strings S1 and S3.



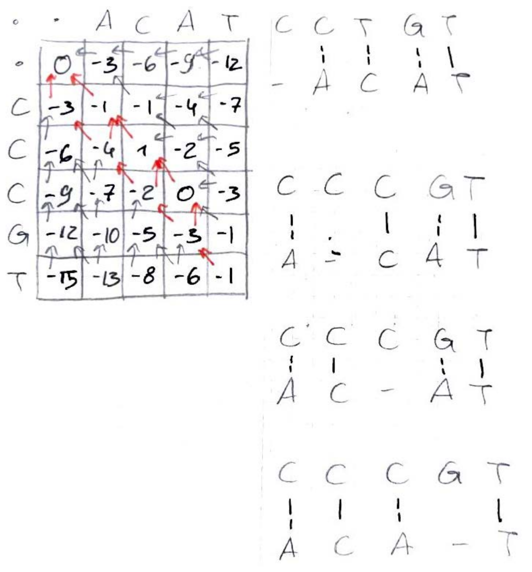
(c) Alignment of strings S1 and S4.



(d) Alignment of strings S2 and S3.



(e) Alignment of strings S2 and S4.



(f) Alignment of strings S3 and S4.

Figure 5: Pairwise alignments for exercise 4.

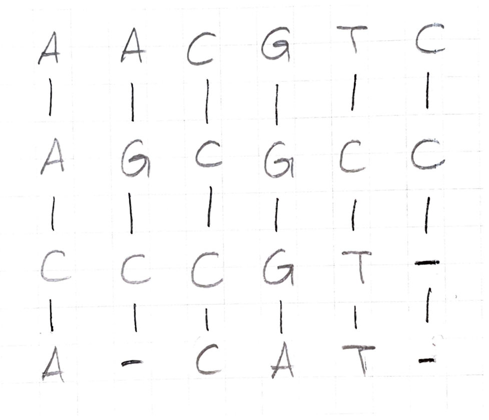


Figure 6: Optimal multiple alignment of strings S1, S2, S3 and S4.

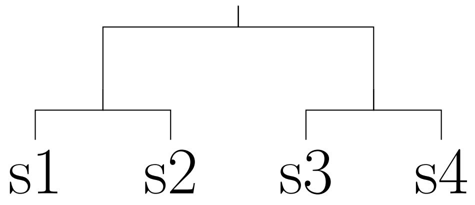


Figure 7: Guide Tree for S1, S2, S3 and S4.

	s1	s2	s3	s4
s1		6	1	-1
s2			-2	-4
s3				-1

Figure 8: Distance matrix for S1, S2, S3 and S4.