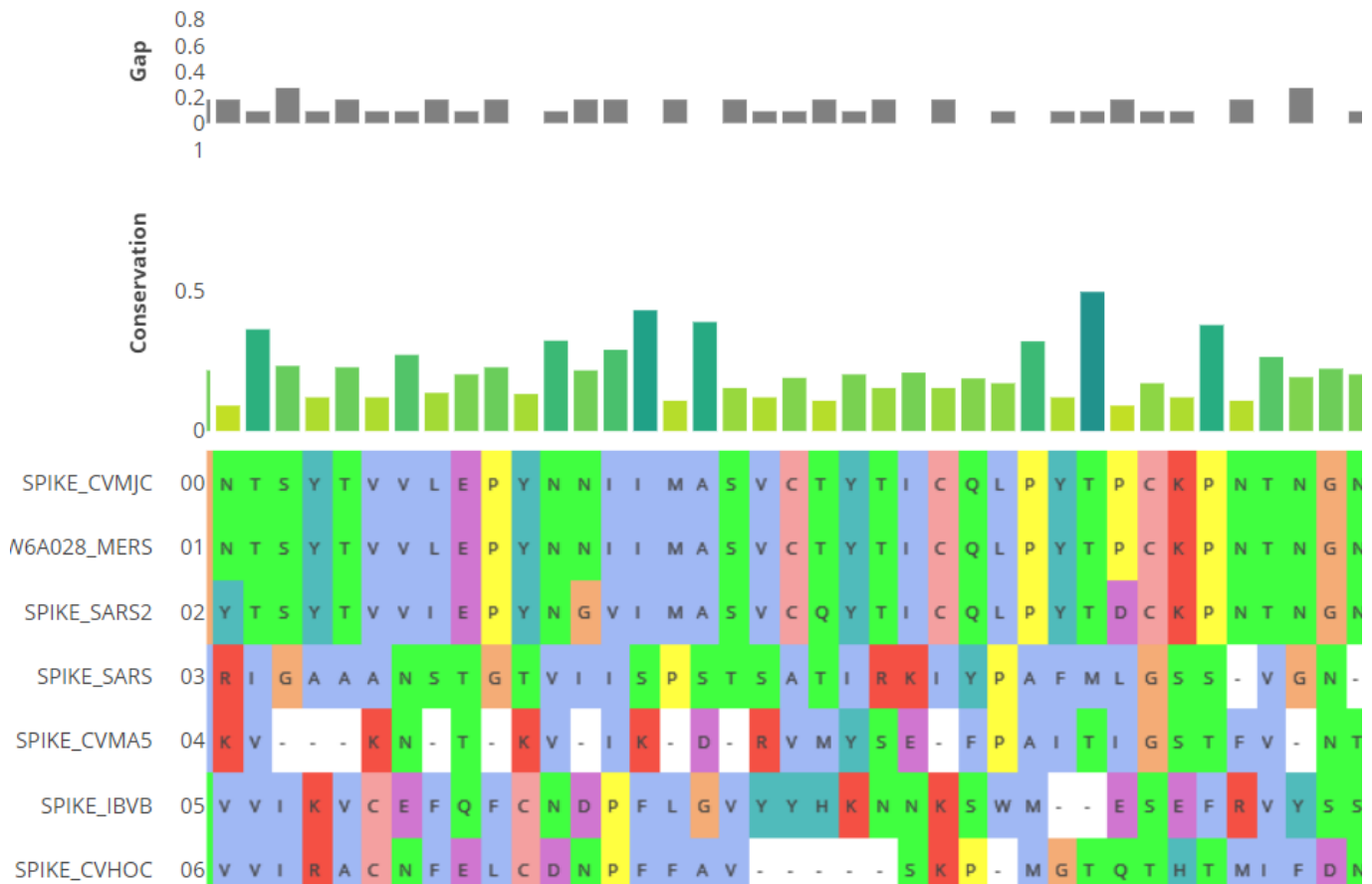


Aligning Protein Sequences

Alexandre de Carvalho Assunção



Contents:

- 1- Introduction.
- 2- The Needleman-Wunsch Algorithm.
- 3- A multiple alignment heuristic.
- 4-Implementation
- 5- Conclusion.

Introduction

Protein sequencing was the birth of bioinformatics and it is still one of the main cornerstones of the field, being used to construct phylogenetic trees, predict protein structure, and to estimate functionally important regions of homologous proteins.

In this project, I will present an implementation of the Needleman-Wunsch algorithm for pairwise alignment of two proteins, an heuristic using this implementation to align multiple sequences, and a short analysis of these methods applied to a series of SARS-CoV2 glycoproteins (spike proteins).

1- The Needleman-Wunsch algorithm:

Sequence alignment is an inherently exponential problem. Greedy strategies and other heuristic methods can produce good results but the only way to get to the coveted global minimum of our search space is to exhaustively test every element of it.

Luckily, there is a technique that manages to sometimes do this exhaustive testing in polynomial time: Dynamic programming. If the optimal solution to our problem contains the optimal solution to our subproblems, and these subproblems overlap and/or are solved repeatedly, we can use dynamic programming. We break the problem into smaller problems, usually (at least in the sequence alignment case) with a bottom up approach,

encode our subproblems in matrix format, and come up with a recursive formula that combines two solved subproblems to obtain the optimal solution of their union. The trick here is that the recursion will simply involve checking certain indexes of our matrix (done in $O(1)$), as opposed to making another function call ($O(n)$ on a good day), turning something that should be exponential into a polynomial problem.

But how do we align two sequences with dynamic programming?

Given two strings $S = (s_1, s_2, \dots s_n)$ and $T = (t_1, t_2, \dots t_m)$, we start by creating two $(n+1) \times (m+1)$ matrices, F and P . F will store the optimal solutions to every subproblem, and P will keep track of the path we took to our global optimum.

In the first step, we initialize F like this:

$$\begin{aligned} F_{0,0} &\leftarrow 0 \\ F_{i,0} &\leftarrow G_{open} + i \cdot G_{widen} \\ F_{0,j} &\leftarrow G_{open} + j \cdot G_{widen} \end{aligned}$$

Where G_{open} is the gap opening penalty, and G_{widen} is the gap widening penalty. The values used for these constants were -10 and -2 respectively. These values were suggested by chatGPT as being standard in the literature when using the BLOSUM62 matrix.

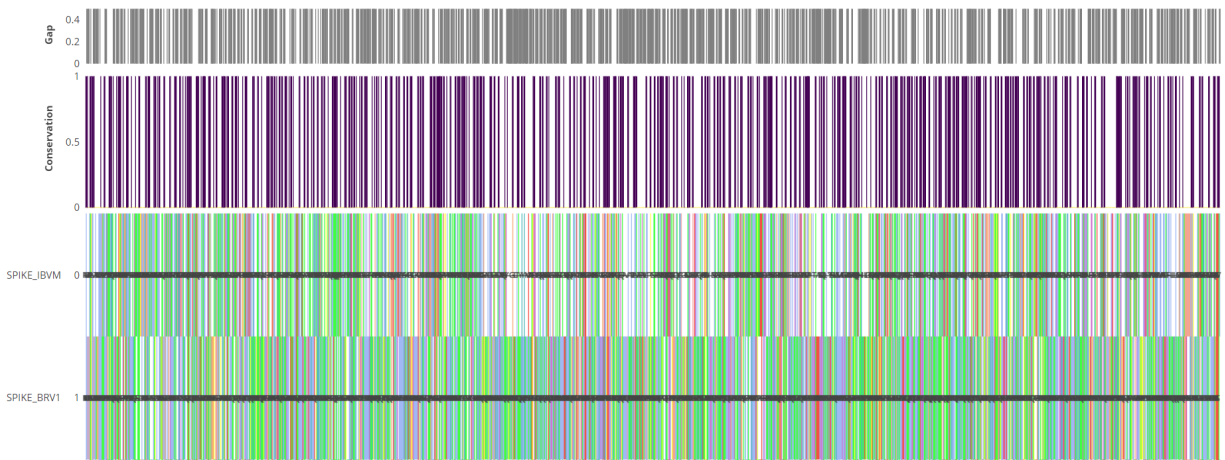
Next, we systematically fill our scores matrix F according to the following recursive formula:

$$F_{ij} \leftarrow \max \begin{cases} F_{i-1,j-1} + B_{p1,p2} \\ F_{i-1,j} + \max(F_{i,j-1} + G_{widen}, F_{i-1,j} + G_{open}) \\ F_{i,j-1} + \max(F_{i-1,j} + G_{widen}, F_{i,j-1} + G_{open}) \end{cases}$$

Where $B_{p1,p2}$ is the BLOSUM62 entry for the amino acids in the positions i and j of the strings S and T .

At each step, we record which path we took, store it in the P matrix, and when we fill the matrix we retrieve our path to arrive at the global optimum.

Below we can see a few results of aligned proteins:



This one is between the SPIKE_IBVM and SPIKE_BRV1 proteins, which didn't receive as good a score as the previous one. There are lots of little gaps conservation "sticks" which don't seem to favor any specific area of the protein.

It might be worth learning how to better interpret these results for future works.

2- A multiple alignment heuristic:

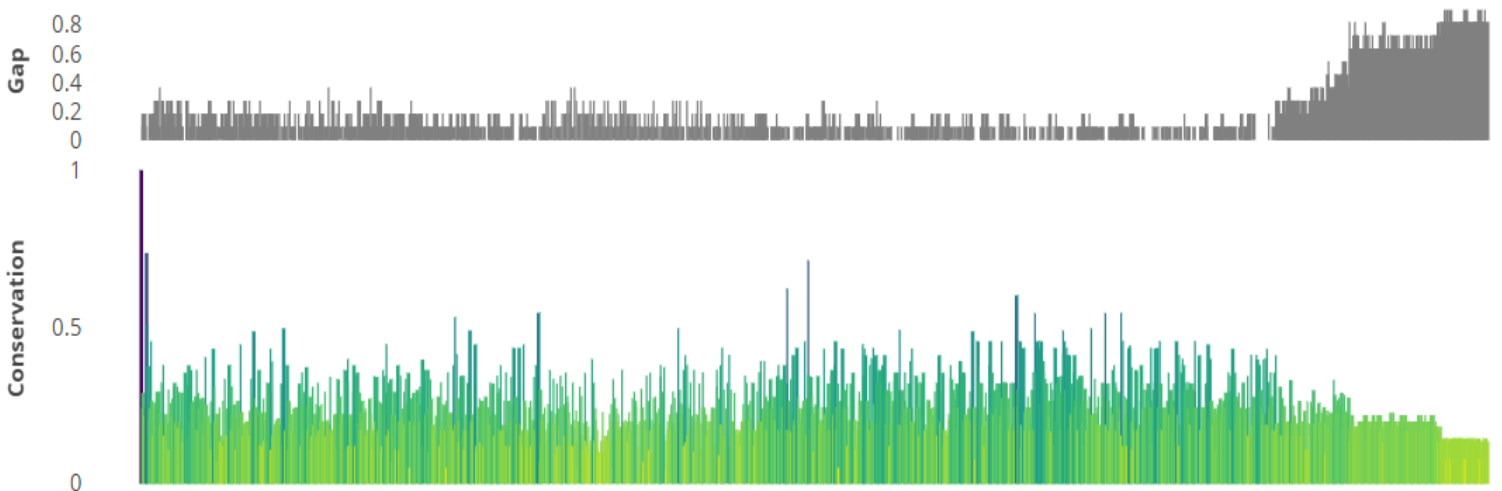
Dynamic programming works great when we want to align two sequences. But what about three? Or four? Fifty? The problem is that we can no longer encode all of the subproblems in a two-dimensional structure. And instead of the 3 checks we needed for two sequences, now we have 8 for the cube encoding three sequences, 16 for the hypercube encoding four, and on and on and now we're back to square one—dealing with an exponential problem. And this time dynamic programming can't help us, so we need an heuristic.

I used progressive alignment after ordering the proteins in a linear path. To order the proteins I created a complete graph with proteins as nodes and their pairwise Needleman-Wunsch scores as the edges' weights. After completing this step, I chose the two proteins with the highest scores, which means the heaviest edge and built this edge's adjacency list. From this list I again pick the best neighbor. These steps are repeated until I have an Eulerian circuit.

Algorithm 1 Heuristic to Order Nodes in a Complete Graph(E, N)

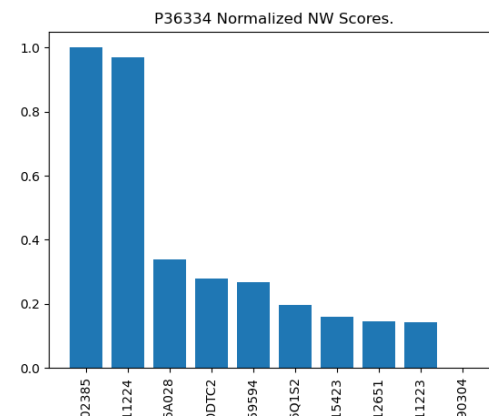
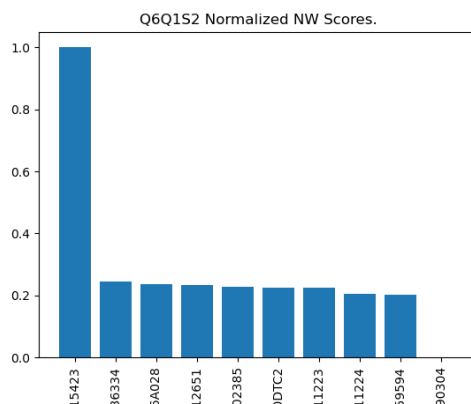
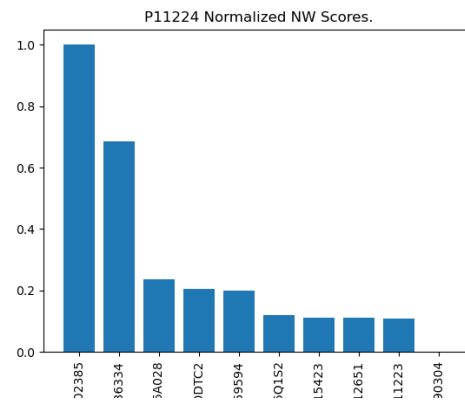
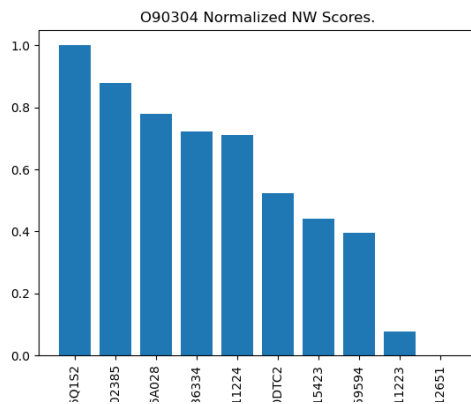
```
Path  $\leftarrow \emptyset$   
NodesInPath  $\leftarrow \emptyset$   
Adj  $\leftarrow E$   
while NodesInPath  $\neq$  Nodes do  
    M  $\leftarrow$  Edge with highest NWScore  
    if M.Nodes() not Visited then  
        M.Nodes()  $\leftarrow$  Visited  
    end if  
    if M.Nodes() was already Visited then  
        Remove M from Adj  
        continue  
    end if  
    Path  $\leftarrow$  Path  $\cup$  M  
    NodesInPath  $\leftarrow$  NodesInPath  $\cup$  M.Node1  $\cup$  M.Node2  
    Adj  $\leftarrow$  M.AdjacencyList()  
end while
```

As the observant reader will have guessed by the image boldly displayed in this report's cover, the heuristic's results ended on the south side of stellar. Below, we can see the full alignment of ten spike proteins from different SARS-CoV 2 strains.



Outside of the very first amino acids, we don't see any areas of high conservation and others of high mutation, as we usually see in proteins. The heuristic was unfortunately not suited to aid us in studying these proteins. And it does make sense now that I am writing this and explaining the algorithm. First, the heuristic assumes evolution happens linearly which is factually wrong. Barring outside forces, diversity tends to increase in every direction. In other words, the heuristic was looking for a cycle when it should've been looking for a very branched tree.

A result that seems to corroborate this fact is the plot of each protein align to all of the others. Below I will show some of these plots, with y axis normalized so they are all between 0 and 1.



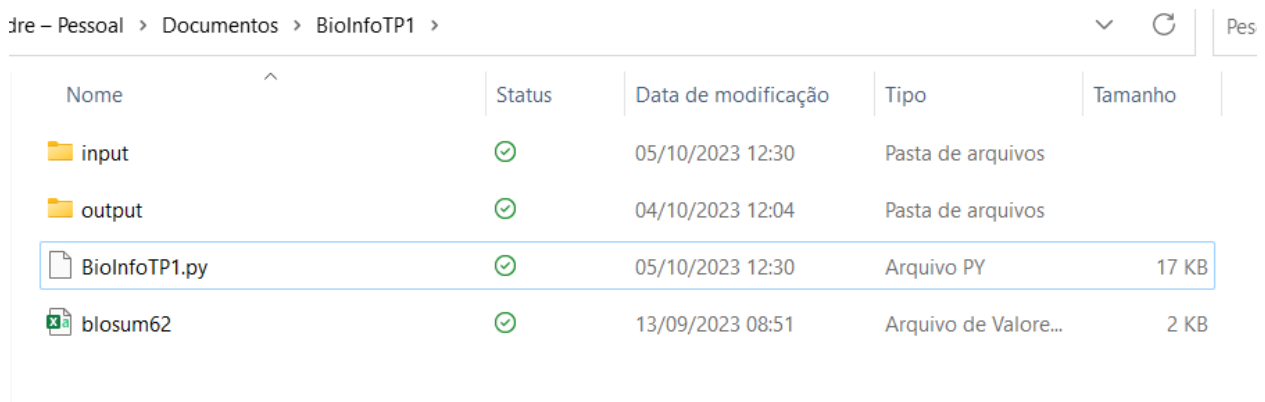
As we can see, some proteins are more similar than all the others, suggesting that the phylogenetic tree should have a high variance when it comes to the degree of each node.

Another problem that is easy to see is the spike in gap number at the end of the graph. It's not good but unfortunately it does make sense. The sequences were aligned pairwise, and the resulting aligned sequences are bigger than the individual sequences. After all, its size is at least the maximum length between the two. At each alignment, the sequence would become bigger and more gaps would end up being inserted.

3- Implementation:

The algorithm (and the multiple alignment heuristic discussed in the next session) were implemented in python3, in the Spyder IDE. In this section I'll review some details of the code, and, more importantly, how to run it.

To start, make sure the directory in which the BioInfoTP1.py file looks like this:



Pessoal > Documentos > BioInfoTP1						▼	↻	Pes
Nome	^	Status	Data de modificação	Tipo	Tamanho			
input		✓	05/10/2023 12:30	Pasta de arquivos				
output		✓	04/10/2023 12:04	Pasta de arquivos				
BioInfoTP1.py		✓	05/10/2023 12:30	Arquivo PY	17 KB			
blosum62		✓	13/09/2023 08:51	Arquivo de Valore...	2 KB			

An input folder containing a fasta file and the blosum62.csv are essential to run the code.

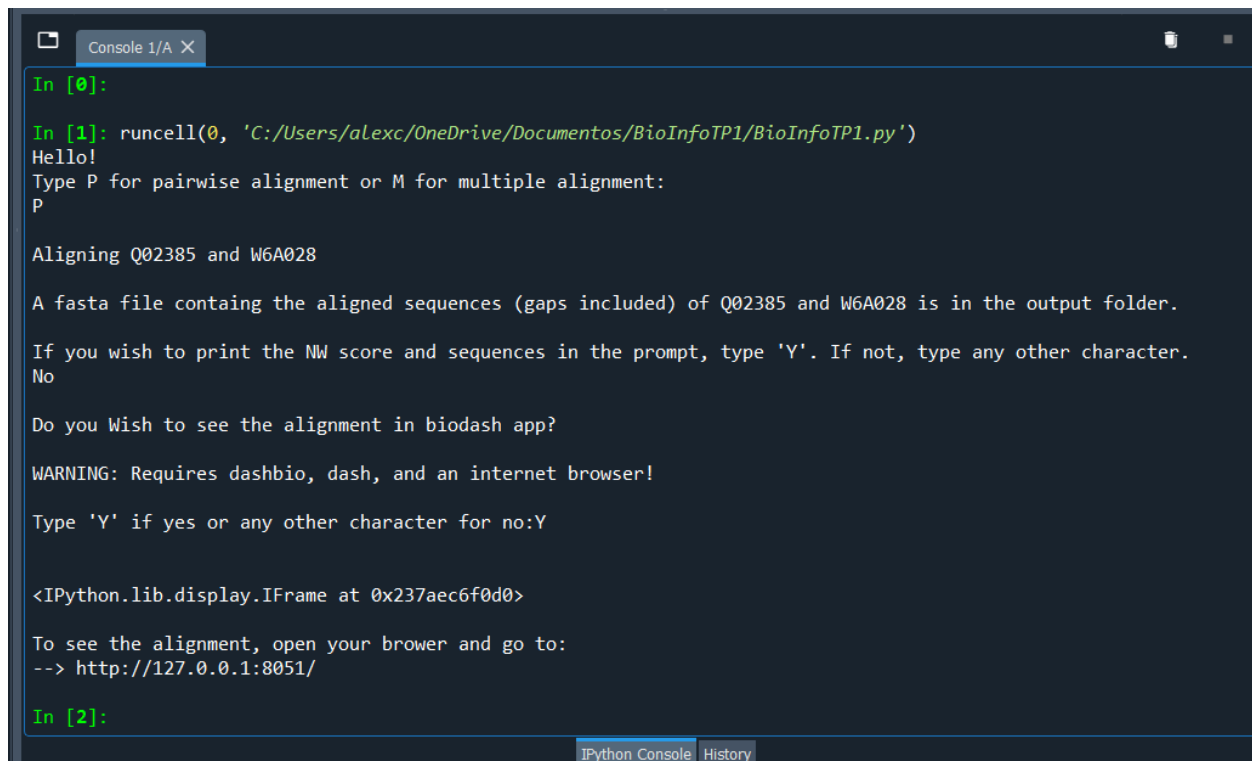
As you run the BioInfoTP1.py, you will be asked whether you wish to run the simple pairwise alignment or the multiple alignment heuristic. The fasta file you wish to sequence

should be in the input folder. If it has multiple proteins and you ask to run the standard NeedleMan-Wunsch algorithm, then the two first sequences of that file will be aligned. By default, the given example *secuencias_spike.fasta* is there.

A fasta file will be generated in the output folder containing the align proteins. And you will also be asked whether you wish to print this alignment on the python prompt, which I only recommend for very small peptides.

Lastly, you will be asked whether you wish to open the biodash sequence alignment visualizer, with which many of the pictures in the report used. If you wish to do so, simply type the address printed on the python prompt on your favorite internet browser.

A print screen of these prompts follows below:



```
Console 1/A X
In [0]:
In [1]: runcell(0, 'C:/Users/aLexc/OneDrive/Documents/BioInfoTP1/BioInfoTP1.py')
Hello!
Type P for pairwise alignment or M for multiple alignment:
P
Aligning Q02385 and W6A028
A fasta file containg the aligned sequences (gaps included) of Q02385 and W6A028 is in the output folder.
If you wish to print the NW score and sequences in the prompt, type 'Y'. If not, type any other character.
No
Do you Wish to see the alignment in biodash app?
WARNING: Requires dashbio, dash, and an internet browser!
Type 'Y' if yes or any other character for no:Y
<IPython.lib.display.IFrame at 0x237aec6f0d0>
To see the alignment, open your brower and go to:
--> http://127.0.0.1:8051/
In [2]:
IPython Console History
```

Conclusion

This first foray into bioinformatics was an enjoyable and interesting experience, despite highlighting my utter complete of knowledge as to how to properly analyse my data and come to meaningful conclusions. I hope to improve my analytical skills to present something more meaningful on the next time. Thank you for reading.