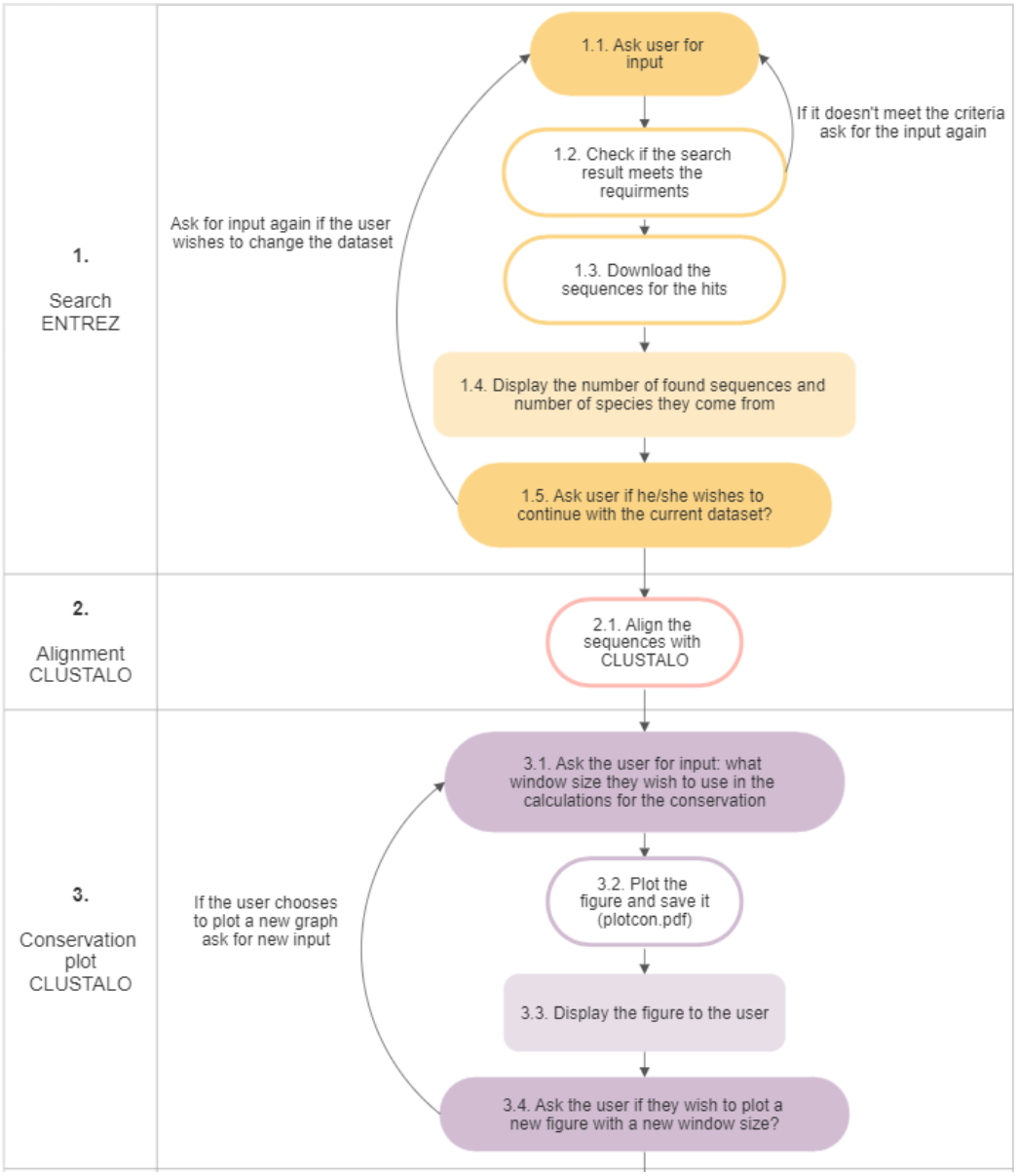
ICA II Script

Coded by: BB226659

Link to my public GitHub repository: https://github.com/B226659-2022/ICA2
Decryption key: BB226659P123

Ordinary user manual:

We prepared a simple flowchart to highlight the steps that the program will take the user through. It shows how different parts of the code work together and what to expect when the program is first to run.



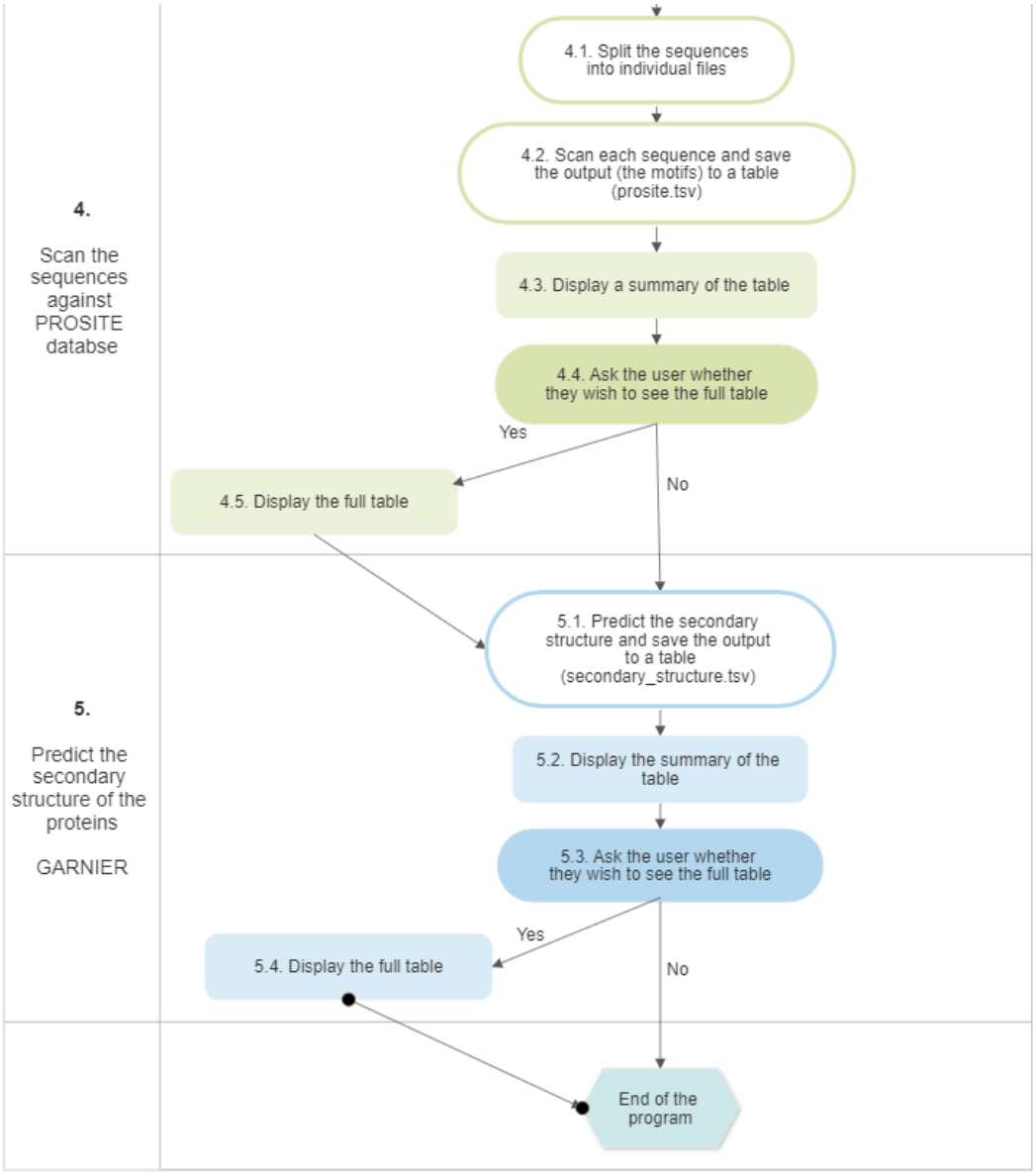


Figure 1. Flowchart describing the process by which the code runs. It is written in simplified terms and divided into 5 parts (each with a separate colour assigned).

- The inputs are represented with coloured-in, round figures.
- The processing steps are represented by outlined by colour, but empty figures
- The outputs are represented by square figures but filled in with translucent

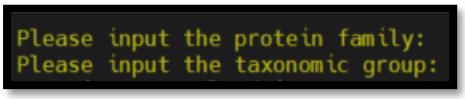
Colour codes

When you first run the script, you will be greeted with lots of text colours and each one of them means a different thing.

- White – is just the background text to let you know what the programme is doing
- Yellow – indicates that the programme is expecting some sort of input
- Blue – text in that colour will contain some advice/hint on either what the programme is doing, or what the user is meant to do
- Green – text in that colour will contain information about the outputs.

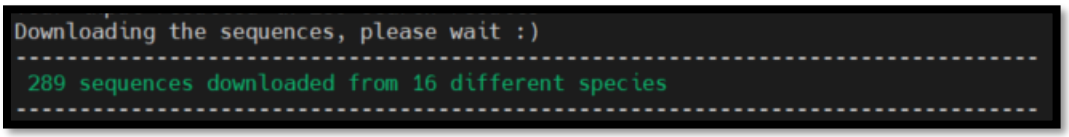
The workflow of the program

1. The program begins by asking the user for input
    - Protein family (the default is **pyruvate dehydrogenase**)
    - Taxonomic group (the default is **ascomycete fungi**)

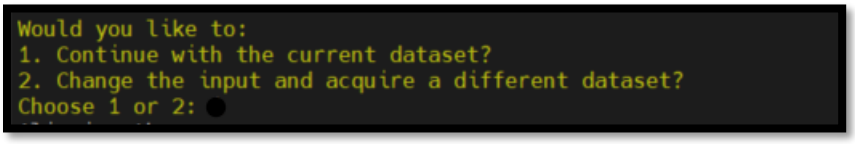


        - In this manual, we will use the inputs above as an example
        - There are error traps in place
            - If the search is empty or fails, the user will be asked for input again
            - If the search returns 0 or too many hits (1000), the script will ask for input again

    - The program will then display the result of the search
        - The number of sequences and the number of unique species will be printed. If all the sequences are from one species, the user might not find this programme scientifically useful.



        - The user will get a chance to choose if they wish to continue with that dataset or get a new one



2. The code will automatically start aligning the downloaded sequences
    - There was an alternative option coded in, but left out (the code can create a distance matrix)
        - This option can be easily employed back in, but we decided to leave it out, due to a long processing time and not an essential need for it.
        - To do that, line no.166 should be replaced with line 177.

3. The next step is going to plot the level of conservation between the sequences from our set
   - Before moving on to the next step of our script, the user will be met with a piece of short information about what the code will attempt to do next.
     - The user can press Enter to continue.
     - This will occur each time the user has finished with one of the major steps of the programme. It will allow the users to use the programme without the manual and still understand what is happening at each step.

   - The user will now have to specify the window size they want the programme to use, in order to calculate the average similarity. (Larger window size means a smoother graph)

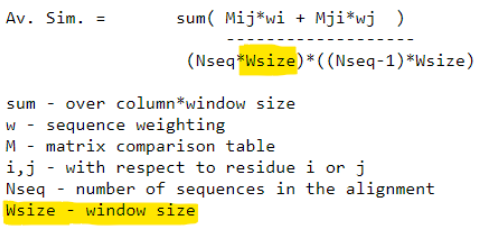
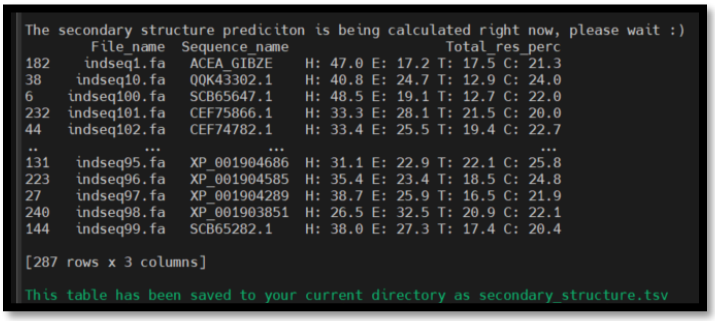


Figure 2. Equation and the legend explaining how the script calculates the average similarity. This might bring some insight into where and how the window size is used.
This figure was taken from the official website of Emboss: https://www.bioinformatics.nl/cgi-bin/emboss/help/plotcon

   - After specifying the window size, the script will save the plot as a "plotcon.pdf" file in the current directory of the user
     - The graph will also be automatically displayed.
       - The user will have to close the pop-up window first and then press enter to continue (in that order).
     - There will also be an option to change the window size and get a new conservation plot

4. Finding any associated motifs from the PROSITE database in our sequences.
   - There is no input required in this step as the script will automatically produce, save (prosite.tsv) and display the summary of the table
   - If the table is too long the user has the option to open it in a text editor, which will allow scrolling through the table
     - The table contains information on which sequences matched with which motifs from the PROSITE database
     - Template sequence names, locations in the sequences, the score, and the names of the motif)
     - If the user wants to return to the script, they will have to press a specific combination on their keyboard (Shift+Z+Z)

5. The script also allows predicting the secondary structure of the proteins in our database.
    - The structure design of this step is the same as in the previous step. There is no need for any input. The table will be saved (secondary_structure.tsv) and its summary displayed.
    - There will be the same option to view the table in full screen in order to scroll through and the same keyboard combination to return to the script.
    - The output table contains the file names, sequence names and four different percentage values. An example of the output is shown below
        - H – represents the helical structures
        - E – represents the beta-sheets
        - T – represents the hydrogen-bonded turns
        - C – represents other coils



Competent Python3 code writer section

The flowchart presented in the “ordinary user” part of the manual might still be useful, but we will now discuss various techniques that were used in this programme and how they link together.

- The code starts with defining a few functions:
    - input_function():
        - This function prompts the user for input and saves it into 2 different variables (p_family and t_group)
        - These variables are then used for creating a command (saved and returned as a variable ‘cmd’) which will later be sent to the operating system, through an os module.
        - The first error trap I created does not allow the user to leave the input empty as the script will keep on requesting it until there is some text in it.
    - search_function()
        - this function will use the cmd variable as input and send an esearch enquiry, which will in turn be stored as another variable search_o
        - we then use this search output to extract the number of hits (through a cElementTree package and stored it as a variable - count) and check if that number is acceptable later
    - step1_search()
        - both of the previous functions are employed in this major function used in the step 1. It connects the input, search and error trapping any failures associated with the search or download.

        - It also extracts the number of different species by iterating through the downloaded file containing all the sequences and extracting the species names and appending them to a list (species_name) and displaying the number of unique sequences to the user
        - The user then has the ability to change the input.
        - Because during that first step the user has many opportunities to change the input and there are plenty of opportunities for error, I decided to split the code into these three functions. This allows to error trap mistakes much earlier and returns the necessary step in order to fix these errors.
        - The chronological start of the code begins at line no. 150
- The next step is running clustalo alignment
    - Because I have already set a hard limit for the maximum number of sequences the user can download (to avoid downloading thousands of unnecessary sequences), there was no need for limiting the number of alignments.
    - I used –verbose to print out the progress of the process, as this step takes a long while, even though it has 32 threads assigned to it.
    - There is a fully runnable command on line 177, which can replace line 166 and create an additional percentage ID matrix.
- Creating the conservation plot
    - I've decided to create a function which will output the plot because I wanted to give the user the opportunity of using different window sizes and change them if they were not satisfied with the previous one.
    - This demanded quite a bit of error traps, as the input had to be a positive integer, therefore any wrong input demanded to run the function anew.
    - This step used the aligned_seqs fasta file created in a previous part
    - I have used GhostScript software through the os module again, to display the graph because all the other applications took too long to load. Pdf also seemed like an optimal format to save the graph as. Unfortunately, GhostScript printed a lot of unwanted warnings and other text, therefore I decided to silence all the shell errors.
- To scan our sequences against the known motifs I had to split the multiseq fasta file which contained all the downloaded sequences into separate files.
    - There were plenty of different methods to do it (commented in the code), but I decided to use pure python to achieve that, by looping through the lines of the file and saving individual sequences into a newly created directory 'Individual'.
    - I had included some try:/except: error traps for users running the script for a second time, so no errors would come up. (e.g., if the directory already exists etc.)
    - By then looping through all the files in the new directory I inserted their names into a patmatmotifs command, which was then sent through the os module to a shell.
    - I used an excel format to output all the desired information and joined all small tables into one big one and saved it in a variable. The variable was then converted into a tab-separated file.
    - The summary of the table was then printed out, with an opportunity for the user to open the full file in a vim editor (read-only mode)

- The final step was done using Garnier, an emboss script, which predicts the secondary structure based on a protein sequence. This was also done in a function (although unnecessary)
    - Unfortunately, there was no format to nicely output exactly what I wanted, therefore I had to extract substrings from a large summary file and save each substring into a variable.
    - The variables were then used to append a panda's data frame.
    - That data frame was then sorted based on the first column (file_name), printed and returned by the function
    - The table was then exported as a tab-separated file and the user was allowed to view the table in the full screen as in the previous step.