Project I: Alignments and Substitution Matrices

Laya Karimi chahrogh

October 19, 2020

The main idea is to do various types of alignment of nucleotide sequences and proteins using score matrices. All code is done in Python.

The main tasks are the following:

- Global and local alignment of nucleotide sequences using a given matrix (BLAST or TTM) or a user defined matrix as a score matrix
- Global and local alignments of protein sequences using PAM, BLOSUM or user defined score matrices
- Alignment of protein or nucleotide sequences, depending on the type.
- Alignment of multiple sequences using CLUSTALW algorithm.
- Semiglobal alignment of nucleotide sequences → this is provided in semi-global.py which works well but I did'nt have time to apply and use it in my code because my codes were gone and I had to do many things again
- Calculating Pam_n for various diveregence → this is provided in pam-n.py which works well but I did'nt have time to apply and use it in my code because my codes were gone and I had to do many things again
- Affine alignment → this is provided in affine.py which works well but I did'nt have time to apply and use it in my code because my codes were gone and I had to do manythings again.

Wikipedia definitions that might give the general idea:

In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences of nucleotide or amino acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns.

Global alignment: The Needleman–Wunsch algorithm is an algorithm used in bioinformatics to align protein or nucleotide sequences. It was one of the first applications of dynamic programming to compare biological sequences.

Local alignment: The Smith–Waterman algorithm performs local sequence alignment; that is, for determining similar regions between two strings of nucleic acid sequences or protein sequences. Instead of looking at the entire sequence, the Smith–Waterman algorithm compares segments of all possible lengths and optimizes the similarity measure.

ClustalW is used for aligning multiple nucleotide or protein sequences in an efficient manner. It uses progressive alignment methods, which align the most similar sequences first and work their way down to the least similar sequences until a global alignment is created. ClustalW is

a matrix-based algorithm. ClustalW uses progressive alignment methods as stated above. In these, the sequences with the best alignment score are aligned first, then progressively more distant groups of sequences are aligned. This heuristic approach is necessary due to the time and memory demand of finding the global optimal solution. The first step to the algorithm is computing a rough distance matrix between each pair of sequences, also known as pairwise sequence alignment. The next step is a neighbor-joining method that uses midpoint rooting to create an overall guide tree. The process it uses to do this is shown in the detailed diagram for the method to the right. The guide tree is then used as a rough template to generate a global alignment. Time complexity is $O(N^2)$.

The project consist of various Python3 and .txt files: .py files:

1. **main** : invokes screen() function from screens, which initializes various options.
2. **screens** : gives user a menu containing various options numbered 1-5, where first four represent 4 main tasks of this program
3. **utilities** : contains input checkers, choosers and file readers
4. **first** : contains global and local alignment functions for protein and nucleotides
5. **second** : Firstly, the sequences are recognized as protein or nucleotide. Secondly, if a nucleotide sequence is given, it will be translated into amino-acid. Finally, sequence alignment will be realized.
6. **third** : contains CLUSTALW algorithm for multiple sequence alignment 7. **pam-n** : calculating pam-n matrix
8. **affine** : contains affine alignment functions for protein and nucleotides
9. **semi-global** : contains semi-global alignment of protein and nucleotides sequences

.txt files:

1. blast : contains matrix for nucleotides alignment
2. ttm : contains matrix for nucleotides alignment
3. pam250 : contains matrix for proteins alignment
4. blosum45 : contains matrix for proteins alignment

PAM matrices are a common family of score matrices. PAM stands for Percent Accepted Mutations, where "accepted" means that the mutation has been adopted by the sequence in question. Thus, using the PAM 250 scoring matrix means that about 250 mutations per 100 amino acids may have happened, while with PAM 10 only 10 mutations per 100 amino acids are assumed, so that only very similar sequences will reach useful alignment scores.PAM matrices contain positive and negative values: if the alignment score is greater than zero, the sequences are considered to be related.

In bioinformatics, the BLOSUM (BLOcks SUbstitution Matrix) matrix is a substitution matrix used for sequence alignment of proteins. BLOSUM matrices are used to score alignments between evolutionarily divergent protein sequences. They are based on local alignments.Several sets of BLOSUM matrices exist using different alignment databases, named with numbers. BLOSUM matrices with high numbers are designed for comparing closely related sequences, while those with low numbers are designed for comparing distant related sequences. For example, BLOSUM80 is used for less divergent alignments, and BLOSUM45 is used for more divergent alignments. The matrices were created by merging (clustering) all sequences that were more similar than a given percentage into one single sequence and then comparing those sequences (that were all more divergent than the given percentage value) only; thus reducing the contribution of closely related sequences. The percentage used was appended to the name, giving BLOSUM80 for example where sequences that were more than 80% identical were clustered.

It should be noted that the current code does not read Fasta files and olny reads .txt file.

However, I had used some code that read Fasta file as well but it was removed and I did not have time to write it again.

To start the program please run following file: **main.py**