Danmarks Tekniske Universitet

DTU

22110 Python and Unix

Project 10. Pairwise Alignment

Authors
Chen Chen (s210168)
Yi Huang (s210304)

November 29, 2021

Contents

1 Introduction

Pairwise Alignment is used to compare two sequences of either protein or nucleic acid to find similar regions which can provide information of the structural, functional, and evolutionary study, and it is a fundamental compute-intensive problem in bioinformatics. The similar regions closely aligned are usually conserved from previous generations. Applying pairwise alignment can help biologists detect pathogens, predict the ancestral, and identify common genes[1].

According to the computational method of dynamic programming used today, pairwise alignment is divided into local and global sequence alignment. Global alignment aims to find a global optimization by aligning the entire query sequence, which is more suitable for closely related sequences and sequences of the same length. By contrast, local alignment aims to find the best approximate sub-sequence match within the two sequences, which is more suitable for divergent or distantly related sequences or sequences that differ in length. In our study, we mainly focus on global alignment by applying the most commonly-used algorithm called Needleman-Wunsch[2].

2 Theory

2.1 The scoring scheme

Global pairwise sequence alignment is the establishment of residue-to-residue correspondence between two sequences from the beginning to the end. To measure the overall similarity of the pairwise sequences, a score value is calculated for each possible alignment given by the query and target sequence. The scoring scheme is based on adding gap penalties to the substitution scores instead of scoring only by similarity score of the two sequence components. In a word, the alignment score is the sum of substitution scores and gap penalties.

2.2 Substitution Matrices

As the main part of global alignment, substitution matrices are a collection of scores for each pair of nucleotides or amino acids in the aligned sequences which are assigned according to the physical and chemical similarity between the pairs[3]. Since some of the amino acids have similar physical or chemical properties, it is more likely for them to substitute for one another compared to the amino acids presenting dissimilar properties. Therefore, instead of using the simplest Match/Mismatch matrix for DNA, we employ another different substitution matrix called Point Accepted Mutation (PAM) matrix for protein sequence comparison.

2.2.1 Match/Mismatch matrix for DNA

As shown in Fig. 1, Match/Mismatch matrix is a simple scoring scheme for DNA by assigning +1 as a reward for a match, -1 as the penalty for a mismatch to form a substitution matrix. This kind of matrix is still commonly used for DNA sequence comparison based on the fact that there are only four nucleic acids and their physical/chemical properties are similar.

	C	T	A	G
C	1	-1	-1	-1
T	-1	1	-1	-1
A	-1	-1	1	-1
G	-1	-1	-1	1

Figure 1: The simple scoring scheme for DNA sequence pairs.

2.2.2 PAM matrix for protein

The protein substitution matrix is more complex because proteins are composed of various amino acids whose number is more than that of DNA. Additionally, the physical-chemical properties of individual amino acids vary considerably.

The relative rates that the amino acids will replace one another over evolutionary time were estimated so that the stochastic model of protein evolution was built as well as the substitution matrix. Thus the model that one point accepted mutation(PAM) which is called PAM1 represents the substitution of 1 percentage of all amino acids by another was brought up where the amino acids in some position would remain fixed while others might change several times even would return to the original residue.

To calculate the amino acid replacement probabilities for a longer time, PAM1 can be multiplied by itself corresponding to the number of times. The use of the PAM250 substitution matrix, which is shown in Fig. 2, describing the replacement probabilities given 250 PAM units of time were advocated for alignments with approximately 20% identical residues, and it is commonly used for aligning two very similar sequences.[4]

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	J	Z	X
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	-1	0	-1
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	-3	0	-1
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	-3	1	-1
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	-3	3	-1
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-5	-1
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	-2	3	-1
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	-3	3	-1
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	-4	0	-1
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	-2	2	-1
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	3	-2	-1
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	5	-3	-1
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	-3	0	-1
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	3	-2	-1
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	2	-5	-1
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	-2	0	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	-2	0	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	-1	-1
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-3	-6	-1
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-1	-4	-1
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	2	-2	-1
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	-3	2	-1
J	-1	-3	-3	-3	-5	-2	-3	-4	-2	3	5	-3	3	2	-2	-2	-1	-3	-1	2	-3	5	-2	-1
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	-2	3	-1
X	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 2: PAM250 substitution matrix.

2.3 Gap Penalty

Most alignment algorithms use a substitution matrix in conjunction with gap penalty, which is a major determinant of the alignment accuracy, to find the best alignment. The gap represents either insertion, deletion mutation among the sequences and the gap has to be penalized with the gap penalty value that has to be subtracted from the total alignment score.

The simple *linear gap penalty* is applied in our project which means the penalty depends linearly on the size of a gap instead of considering opening a gap and extending a gap which refers to *affine gap penalties*[5]. The gap could maximize the chances of alignment of the residues which is called optimal alignment that will be the path through the array that has the highest score which also means it has the largest number of matches and fewest mismatches and gaps. However, assigning penalty values can be more or less arbitrary because there is no evolutionary theory to determine a precise cost for introducing insertions and deletions[6].

3 Needleman-Wunsch Algorithm

Needleman-Wunsch algorithm is considered as the optimal option for global alignment over the entire length of two sequences with similar length and a significant degree of similarity throughout. The algorithm aims to assign a score to every possible alignment based on the substitution matrix and gap penalty, and the best alignment is identified with the maximum alignment score. The three main steps in this algorithm are demonstrated below.

3.1 Initialization of the matrix

Matrix of $N + 1$ rows and $M + 1$ columns is first created where N and M refer to the length of two sequences to be aligned. The scores of the first row and the first column presented in Fig. 3 are assigned by adding the gap penalty -2 gradually along the row and the column to initialize the main matrix.

		G	A	G	A	C	C	G
	0	-2	-4	-6	-8	-10	-12	-14
A	-2							
G	-4							
A	-6							
C	-8							
C	-10							
C	-12							
A	-14							

Figure 3: Initialization of the matrix.

3.2 Matrix filling with maximum scores

The score of any cell(i,j) is the maximum of:

$$q_{diag} = C(i-1, j-1) + S(i, j), \tag{1}$$

$$q_{up} = C(i-1, j) + g, \tag{2}$$

$$q_{left} = C(i, j-1) + g, \tag{3}$$

where $S(i, j)$ is the score found in the substitution matrix for letter i and j, and g is the gap penalty. The value of the cell $C(i, j)$ depends only on the values of the immediately adjacent northwest diagonal, up, and left cells shown in Fig. 4.

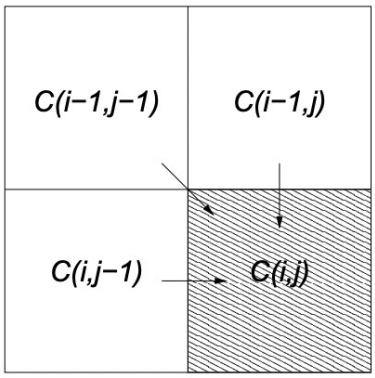

Figure 4: Matrix filling.

3.3 Traceback for approriate alignment

The best way of alignment is found in this step by calculating the origin of the target score. We will always start from the right bottom corner of the matrix where the highest value lies and begin tracing back up to the upper left corner from where we have started the matrix filling process. There are three possible moving directions for each step, namely the diagonal direction (towards the top-left corner of the matrix), the upper, or the left for us to trace the arrows back which leads us to the end corner. As long as it fits the situation that the value of the target cell originated from the value in the diagonal cell, the traceback will go diagonally, otherwise, the traceback will go towards the highest valued arrow which is used when filling the matrix[7]. One example of the Traceback step is presented in Fig. 5 below.

		G	A	G	A	C	C	G
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1	-1	-3	-5	-7	-9	-11
G	-4	-1	-2	0	-2	-4	-6	-8
A	-6	-3	0	-2	1	-1	-3	-5
C	-8	-5	-2	-1	-1	2	0	-2
C	-10	-7	-4	-3	-2	0	3	1
C	-12	-9	-6	-5	-4	-1	1	2
A	-14	-11	-8	-7	-4	-3	-1	0

Figure 5: An example of Traceback.

4 Program Design

To better explain the design of our program, we constructed an overview of the program pipeline illustrated in Fig.6. Seven functions were made and implemented to achieve the alignment purpose from the beginning of the input fasta file to the end of an output file with the aligned sequences in it.

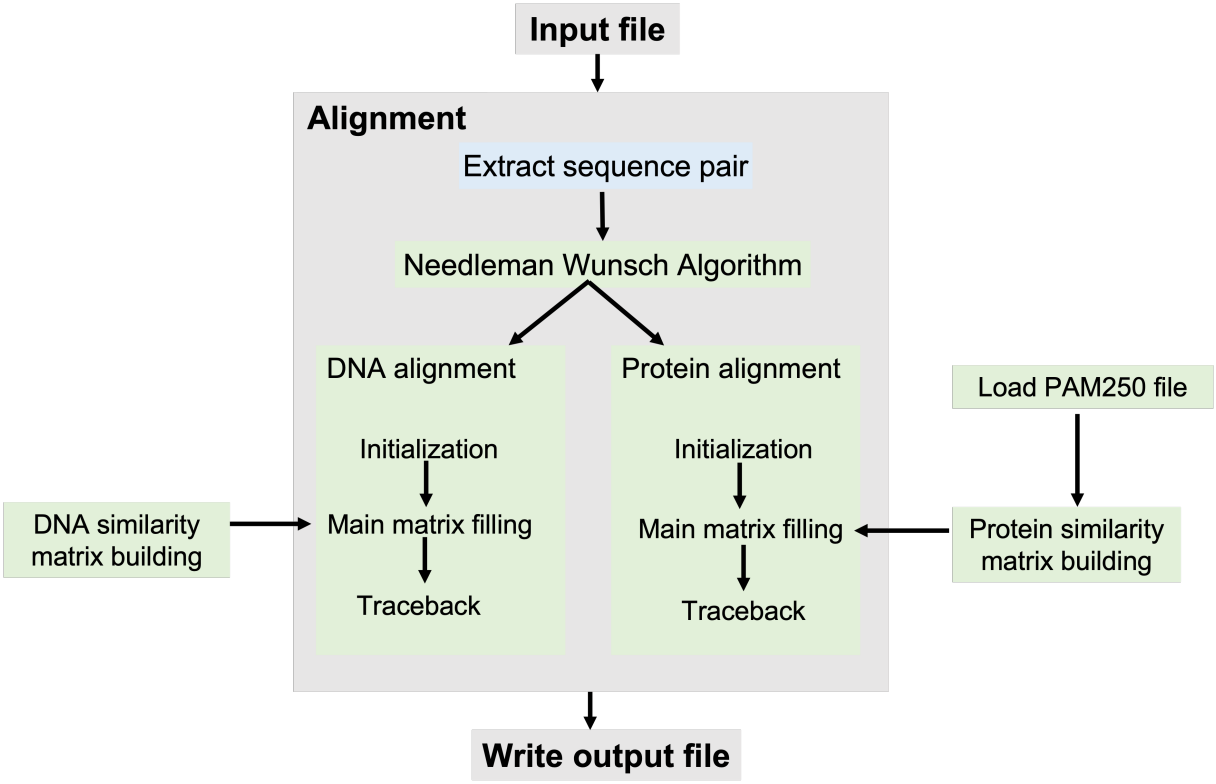

Figure 6: Program pipeline of pairwise alignment.

The second main step **Alignment** is implemented by using the function *Align()* in our program, which takes the initial fasta file as input and returns the final aligned sequences as string variables *aligned1* and *aligned2*, and their corresponding identifiers as a list called *identifiers*. The results of this step is then taken as an input of *write_result()* function to finally write the output file we want.

Inside of the *align()* function, the other two functions are called one by one to return the three variables. The first function is *extract_seq()* which returns a dictionary with the keys being the identifiers and values being the extracted two sequences in the input file. The second function *needleman_wunsch()* is the place where the main algorithm introduced before was implemented. The function takes two individual sequences as inputs, and will first estimate the category of these two sequences to assign the corresponding substitution/similarity matrix to the variable called *simMatrix* which will later be used in the calculation of **Main matrix filling**, the second step of Needlman Wunsch algorithm.

As for DNA alignment, *simMatrix* is returned from the function *DNA_similarity_matrix()* to record the match or mismatch score of each pair in the input DNA sequences. If the input sequences are amino acids, then *load_PAM()* is first implemented to load and transform the substitution matrix data in *PAM250.txt* to a python dictionary which is then used in *AA_similarity_matrix()* to return the *simMatrix*. The .txt file is originally form Here.

5 Program Manual

Our python program is in *Pairwise_align.py*, and it should be executed in the terminal of a Unix system. The input file should be a fasta file with the standard format that the first line is a header starting from ">" followed by the specific date sequence. The whole fasta file should only include two sequences of either the nucleotide sequences or the amino acid sequences. The example input files of both DNA and protein sequences are submitted together called *dna.fsa* and *protein.fsa*.

To run the program, two options can be considered. One is by typing down the command *./Pairwise_align.py*, and the program will then ask you to type the name of your input fasta file. The other way is to add one more argument of <*filename*> after *./Pairwise_align.py* in the terminal.

After execution of the program, "Job done. Please check the result in aligned.fsa" will be printed on the screen and the results for both DNA alignment and protein alignment corresponding to each input file can be checked in the files called *dna_aligned.fsa* and *protein_aligned.fsa*, which are also submitted as the example ouput files together with other files.

6 Runtime Analysis

Big-O notation is used for dealing with approximations according to the growing run time and space behavior depending on the growing inputs[8]. To study the efficiency of algorithms, Big-O notations will be applied[9]. The Big-O notation has been calculated for all sections we applied and will be elaborated on as follows.

Big-O notation for the function *load_PAM()* is O(n*n) that is simplified from O(n + n*n). Firstly, the *PAM250.txt* file needs to be loaded into memory and it contains n lines which give us O(n). Secondly, the substitution matrix will be built into a dictionary where O(n*n) is introduced because the length of the keys and values in the dictionary are the same based on the symmetry of the PAM250 substitution matrix. The Big-O notation for the function *AA_similarity_matrix()* is O(n*m) that is simplified from O(n*n + n + n*m). Firstly, the *load_PAM()* is called in the function so that O(n*n) is obtained. Secondly, the O(n) is the notation for initializing the similarity matrix with zero. Thirdly, O(n*m) is from filling the similarity matrix according to PAM250 where the n and m are the dimensions of the matrix which refers to the lengths of the two sequences. Theoretically, the size of the two sequences should be much bigger than the size of the PAM250 substitution matrix so that the n*n part will be discarded. The Big-O notation is same as *AA_similarity_matrix()* could be inferred for the *DNA_similarity_matrix()* analogously.

After having the substitution matrix, the **Needleman-Wunsch algorithm** which has been elaborated before will be applied by using the function *needleman_wunsch()*. And the Big-O notation is O(n*m) that is simplified from O(n*m + n + 1 + n*m + n*m). Firstly, the *simMatrix* either from *AA_similarity_matrix()* or *DNA_similarity_matrix()* must be obtained before calculating the algorithm and the Big-O notation is O(n*m) which has already been explained. Secondly, the Big-O notation for the first step of the algorithm called *initialization of the matrix* is O(n +1) where the original matrix will be built which gives us O(n) and then the first row and column filled by adding gap penalty which gives us O(1*1). Thirdly, the Big-O notation for the second step of the algorithm called *matrix filling* is O(n * m) which is simplified from O((n-1)*(m-1)) where (n-1) and (m-1) are the dimensions of the remaining matrix that need to be filled after initialization of the matrix. Fourthly, the last step called *traceback* is realized and the Big-O notation of it is O(n*m) where the n and m are the dimensions of the matrix.

After finishing the algorithm calculation, the main task of our project of pairwise alignment could be implemented by calling *extract_seq()* and *needleman_wunsch()* in the main function *Align()*. The Big-O of it is O(n*m) that is simplified from O(n + n*m). The Big-O of extracting the accession numbers and sequences that is implemented by calling *extract_seq()* is O(n), because the extraction step is done by looking the file line by line. And the Big-O of *needleman_wunsch()* has been explained before. Additionally, the Big-O of writing the alignments by calling the function *write_result()* is also O(n) that has the similar theory with *extract_seq()*.

Big-O analysis has been performed on all sections or functions in our program and the total Big-O should be O(n*m) which is simplified from O(n*n + 4n*m). The value of n*m is theoretically higher than n*n based on the size of the main matrix refers to the *Needleman-Wunsch algorithm* should be larger than the size of the PAM250 substitution matrix.

7 Conclusion

7.1 Results Achievements

Either two similar protein sequences or DNA sequences could be aligned by running our program and the result could be checked in the file *aligned.fsa*. From the result, we can identify the regions of similarity by looking at the correspondences of the residue-residue which will help researchers analyze the gene regulation, the patterns of conservation, and the evolutionary relationships between the two biological sequences.

7.2 Weaknesses of the Program

Firstly, the protein substitution matrix we introduced is *PAM250* which is used to score alignments between closely related protein sequences which means that scoring alignments between evolutionary divergent protein sequences cannot be ideally achieved by running our program. Secondly, the match/mismatch matrix applied for DNA is also the simplest one indicates that we didn't consider the influence given by the difference between *purines* and *pyrimidines* where A and G are purines (pyrimidine ring fused to an imidazole ring) while T and C are pyrimidines (one six-membered ring). Thirdly, the simplest gap penalty called *linear gap penalty* is implemented in our program instead of using more complicated or precise gap penalties which might consider the gap opening and gap extending where the penalty for gap opening is higher than gap extending since that it is easier to extend a gap that has already been started.

8 Contribution

Both authors make their own contribution to this project and generally they are equally contributed. Details can be found in Table 1 below.

Table 1: Contribution table.

Section	Coding		Report				
	DNA related alignment	Protein related alignment	Section 1	Section 2-3	Section 4-5	Section 6	Section 7
Chen Chen	30%	70%	30%	70%	30%	70%	50%
Yi Huang	70%	30%	70%	30%	70%	30%	50%

References

(1) Haque, W.; Aravind, A.; Reddy, B. In *Proceedings of the 2009 conference on Information Science, Technology and Applications*, 2009, pp 96–103.

(2) Needleman, S. B.; Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* **1970**, *48*, 443–453.

(3) Altschul, S. F. Substitution Matrices. *eLS* **2008**.

(4) Wheeler, D. Selecting the right protein-scoring matrix. *Current protocols in bioinformatics* **2003**, 3–5.

(5) Altschul, S. F. Generalized affine gap costs for protein sequence alignment. *Proteins: Structure, Function, and Bioinformatics* **1998**, *32*, 88–96.

(6) Madhusudhan, M.; Marti-Renom, M. A.; Sanchez, R.; Sali, A. Variable gap penalty for protein sequence–structure alignment. *Protein Engineering Design and Selection* **2006**, *19*, 129–133.

(7) Likic, V. The Needleman-Wunsch algorithm for sequence alignment. *Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne* **2008**, 1–46.

(8) Chivers, I.; Sleightholme, J. In *Introduction to programming with Fortran*; Springer: 2015, pp 359–364.

(9) Devi, S. G.; Selvam, K.; Rajagopalan, S. An abstract to calculate big o factors of time and space complexity of machine code. **2011**.