Project Work for Tik-61.181 Bioinformatics

"Pairwise Alignment: Global alignment"

Antonio Villena (55809H)
Antonio.Villena-Borrego@nokia.com

SUMMARY. In this research paper, we consider the protein structure alignment problem, which is a very important problem in molecular biology. Although great differences may occur when applying different algorithms to solve this problem with the same target strings, the fact is that structure alignment of proteins can help us in finding regions of similar sequence that defines a conserved consensus pattern (leading us to think that two proteins may or may not have a similar biochemical function or structural fold) or, if the alignment is particularly strong, used the aligned positions to derive possible evolutionary relationships among the sequences.

Particularly, this paper centers on the existing differences between applying Global alignments and Local alignments to solve the introduced problem. With that purpose the paper also introduces the most extended algorithms of each trend, the *"Needleman-Wunsch algorithm"* and the *"Smith-Waterman algorithm"* respectively, trying to make a small comparison of their efficiency and complexity.

1. - Introduction:

The problem of finding the optimal alignment of two different sequences (if they are the same, the problem is trivial) is addressed as follows: given two sequences of elements X (x_1 x_2 …x_m) and Y (y_1 y_2 … y_n), and an M times N score matrix P (which describes the differences between elements i and j from the first and second sequence respectively) where M and N are the numbers of elements in the first and second sequence respectively, the goal is to obtained an optimal set of equivalencies that match elements of the first sequence to elements of the second sequence. The optimal set of equivalencies is the one that obtains the highest alignment score, defined as the sum of scores corresponding to the matched elements.

The next step in our problem is determining whether or not we allow non-equivalenced elements in our alignment. This is known as the "Gap problem".

Sometimes we want no-gap alignments, where deletion or insertions in the chain of amino acids would be likely to destroy the biochemical function of the protein. In such regions, we want to align using matches or replacements only. In our dynamic programming algorithm this can be achieved by setting the gap penalty to infinity (or something close to it).

But the most realistic option would be to consider that evolution tends to insert or delete entire substrings of amino acids. This is done by establishing a certain setup cost for introducing a new gap in the resulting alignment, whereas extending an existing gap would always be less expensive.

Finally, we must consider if we want to perform an exhaustive alignment of the two sequences, that is look for the best match from one end to another (problem known as "Global Alignment"), or only look for the best alignment between subsequences of x and y (problem know as "Local alignment"). This second setting arises when looking for share domains or patterns in the given sequences. That is because some part of the proteins evolving structure might have been under strong selection, while the rest of the protein might have suffer strong mutations leading to unlike alignments. As we will see later on, the basis for both points of view are almost the same, differing the local alignment in that it affirms that it's better to restart an alignment that to carry a negative score from previous matchings.

A Word on scoring matrixes.

As previously stated, the scoring matrixes used in the algorithms, gives us an idea of how different two elements from different sequences really are, and the probability they have to be matched. Because of this, pairwise alignment algorithm´s depends totally on the scoring matrix used to align both sequences, to the point that an alignment based on the same algorithm but on different scoring matrixes, can lead to contradictory results.

Although it is not the objective of this paper to settle the basis for defining which scoring matrix should be used, the reader should take into account that there are different types of scoring matrixes (eg. Based on a metric's measurement of distance, or on unit's cost) and should be able to distinguish which type of scoring matrix he has to apply in each case.

2. - Methods and parameters used:

- The *"Needleman-Wunsch"* algorithm:

The idea behind this algorithm is to use previous solutions for optimal alignments of smaller subsequences. The algorithm consist of 2 procedures:

1st Construct the matrix F(i, j) which is the score of the best alginment between the initial segment $x_{1..i}$ of x up to x_i and the initial segment $y_{1..j}$ of y up to y_j .

// Initialize procedure

$$f(0,0) = 0$$
$$f(i,0) = (-i) * d$$
$$f(0,j) = (-j) * 0$$

// Recursive algorithm

$$f(i,j) = \max\begin{cases} f(i-1, j-1) + p(xi, yj) \\ f(i-1, j) - d \\ f(i, j-1) - d \end{cases}$$

d represents the gap, and depending of the gap model used, it should be replaced for one value or the other. In our case the gap will consist only of a gap open penalty, which follows a linear distribution.

$$\gamma(g) = -d * g$$

Although this gap model it´s sufficient for the purpose of the paper, it should be wise to introduce a more realistic gap model consisting of a gap open penalty, and a gap extension penalty, like:

$$\gamma(g) = -d - (g-1)\, e$$

2nd Construct the *"traceback"* matrix where each element will contain the element from which it was derived. We have to bear in mind that there could be more than one possibilities leading to different optimal alignments.

This is the case of the implemented algorithms, where not only one of the possible best alignments is obtained, but all of them. This is the wiser option when presenting the results of the alignment, but we have to be aware of the complexity it introduces when dealing with very long strings.

Once we have constructed both matrixes the problem of constructing the aligned sequence is trivial. We start from the element at the bottom right corner of the traceback matrix and we follow it to the first element (that would be the one in the top left hand corner) adding the respective scoring matrix element to sum up the alignment cost.

- The *"Smith-Waterman"* algorithm:

The bases for the Smith-Waterman algorithm are the same than for the Needleman-Wunsch algorithm. There are only two differences:

- The first one is that during the construction of matrix P the alignment is restarted each time the score goes negative. It’s better to start a new alignment than to extent a new one.

- The second difference is that the backtracking starts from the highest element of the matrix, wherever it is, and doesn´t have to end in the first element of the matrix.

Tanking into account both differences, the resulting equations for the algorithm are:

// Initialize procedure

$$f(0,0) = 0$$
$$f(i,0) = 0$$
$$f(0,j) = 0$$

// Recursive algorithm

$$f(i,j) = \max\begin{cases} 0 \\ f(i-1, j-1) + p(xi, yj) \\ f(i-1, j) - d \\ f(i, j-1) - d \end{cases}$$

3. - Description of the results:

With the purpose of comparing both algorithms, they´ve been implemented using C programming language in a UNIX environment, using a digital platform. Although these implementations differ quite much from being the most optimal and most efficient of both algorithms, they still work quite well, giving an outline of how both algorithms work, and what should be the future guidelines if considering their optimatization.

Both algorithms have been tested using the *"blosum50"* scoring matrix, and the results have been cross out with a real alignment tool known as "alionment" and found at http://dna.stanford.edu/alion/. There has been a testbench of 20 pairs of sequence on both algorithms, and the success target was of 95%.

Because of the complexity of both algorithms, and because the main scope of this paper is the efficient implementation of both algorithms, sequences containing more than 20 aminoacids have not been tested.

About the complexity obtained, it's $O(n^3)$ in time and $O(n^2)$ in space . Although both of the algorihtms have the same complexity, Waterman-Smith algorithm is likely to have less complexity than Needleman-Wunsch because the first would give an alignment that is a subset of the second one:

$$Needleman's\_alignment \subseteq Waterman's\_alignment$$

For more information about the output of the program, or information how to use it, please refer to Appendix 1 "Using the progam".

4. - Conclusions:

Three important conclusions can be infered from this paper:

- Pairwise alignment can help us determine whether two proteins have evolved from the same evolving branch, or, evolved from different ancestors.

- As far as a complete alignment is not needed, Waterman´s algorithm is always more efficient than Needleman´s.

- Special attention to the combinatorial explosion of the traceback procedure should be payed when trying to make all the possible best alignments.

5. - References:

[1] "Introduction to computational Molecular Biology"

Setubal / Meidanis

[2] "Computational Methods in Molecular Biology"

S.L Salzberg / D.B Searls / S. Kasif

Appendix 1. "Using the program"

A. - Output of the program:

- Global alignment: Needleman-Wunsch.

alpha bio 54 % align ent1.txt ent2.txt

Select an algorithm [l to list]: l
1. Needleman-Wunsch algorithm [global alignment]
2. Waterman-Smith algorithm [local alignment]

Select an algorithm [l to list]: 1
loading Blosum 50 matrix
Enter the opening gap penalty: 8

Source string [1]: HEAGAWGHEE
Source string [2]: PAWHEAE

There are 3 possible best alignments with an overall cost of -108 units:

[1] alignment:

HEAGAWGHE-E
--P-AW-HEAE

[2] alignment:

HEAGAWGHE-E
-PA--W-HEAE

[3] alignment:

HEAGAWGHE-E
-P--AW-HEAE

- Local alignment: Waterman-Smith.

alpha bio 55 % align ent1.txt ent2.txt

Select an algorithm [l to list]: l
1. Needleman-Wunsch algorithm [global alignment]
2. Waterman-Smith algorithm [local alignment]

Select an algorithm [l to list]: 2
loading Blosum 50 matrix
Enter the opening gap penalty: 8

Source string [1]: HEAGAWGHEE
Source string [2]: PAWHEAE

There are 1 possible best alignments with an overall cost of 87 units:

[1] alignment:

AWGHE
AW-HE

B. – Use of the program:

> Align file1 file2

Where file1 and file2 contains the strings to be aligned respectively. Both should be in ascii format.

C. – Compiling the source code:

As stated before in the paper, all the algorithms have been implemented using C language in a UNIX environmnet using a digital platform. Because some specific non-posix libraries have been used, the source code won't porbably compile in other platform which is not digital. For that purpose, if no digital computer is available for testing the code, I recommend using the digital computers in Maarintalo. If there is still any problem running the code, I'll be glad to make a short demostration of the code.

Commands for compiling the source code:

> gcc align.c –o align