***Sequence Alignment Problem***

***Global Alignment, Needleman–Wunsch algorithm***

Kafi Khan, Monika Hossian, Sharmin Islam

2017-2-60-078, 2016-2-60-127, 2016-2-60-088

Department of Computer Science and Engineering, East West University, Dhaka-1212, Bangladesh.

Email: kafik\_han@outlook.com, hossainmonikaewu2016@gmail.com, sharminmou28@gamil.com

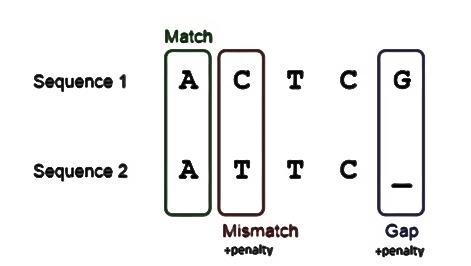
***Abstract* - Evolution has preserved functional elements in the genome. Such preserved elements between species are often homologs – either orthologous or paralogous sequences. Orthologous gene sequences are of higher interest in the study of evolutionary paths due to the higher influence of purifying, since such regions are extremely well preserved. A common approach to the analysis of evolutionary similarities is the method of aligning sequences, primarily solved using computational methods (e.g., dynamic programming). This paper discusses the sequence alignment problem, the technique of dynamic programming, and a specific solution to the problem using this technique.**

***Keywords - Evolution, genome, Orthologous, aligning sequences, dynamic programming.***

1. INTRODUCTION

Alignment of two or more sequences of numbers or letters is common in several in several fields, such as molecular biology, speech recognition and computer science. Sequence alignment is particularly important in molecular biology where it has been critical in the study of evolution, the control of gene expression, and in the analysis of protein structure/function relationships.

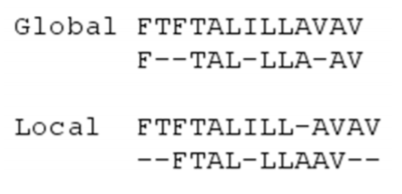
In comparison of two sequences of characters, should characters match, it does not cause any penalties. Other cases such as gap and mismatch, cause penalty thus implying the extent of dissimilarly between the sequences. This is demonstrated below;



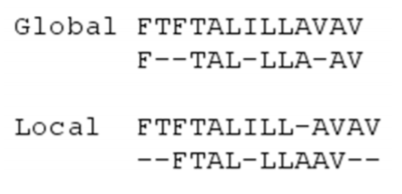
1. ALIGNMENTS

According to the length of sequence it is being compared of following two types:

1. *Global Alignment:* It attempt to align every residue in every sequence, are most useful when the sequences in the query set are similar and of roughly equal size. (This does not mean global alignments cannot start and/or end in gaps.) A general global alignment technique is the Needleman–Wunsch algorithm, which is based on dynamic programming



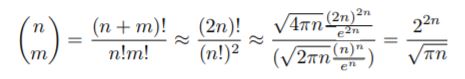
1. *Local Alignment:* Local alignments are more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context. The Smith–Waterman algorithm is a general local alignment method based on the same dynamic programming scheme but with additional choices to start and end at any place.



This paper is going to foucs on global alignments.

1. APPROACH

Brute-force Approach: Given a metric to score a given alignment, the simple brute-force algorithm enumerates all possible alignments, computes the score of each one, and picks the alignment with the maximum score. This leads to the question, ‘How many possible alignments are there?’ If you consider n > m, the number of alignments is (n and m being the size of two sequences)



For some small values of n such as 100, the number of alignments is already too big (> 1060). the number of possible alignments for such n exceeds the number of atoms in the universe. Thus, using a better algorithm than brute-force is a necessity.

Dynamic Approach: Dynamic programming algorithms find the best solution by breaking the original problem into smaller sub-problems and then solving. Dynamic programming may be used to solve problems with:

* *Optimal Substructure*: The optimal solution to an instance of the problem contains optimal solutions to subproblems.
* *Overlapping Subproblems*: There are a limited number of subproblems, many/most of which are repeated many times.

Dynamic programming is usually, but not always, used to solve optimization problems, similar to greedy

algorithms. Unlike greedy algorithms, which require a greedy choice property to be valid, dynamic programming works on a range of problems in which locally optimal choices do not produce globally optimal results. To align two sequences, dynamic approach is slow, but effective as it produces the correct result in all cases. The Needleman-Wunsch algorithm is a dynamic programming algorithm for optimal sequence alignment.

1. NEEDLEMAN-WUNSCH ALGORITHM

The Needleman-Wunsch algorithm is an example of dynamic programming, it was developed by Saul B. Needleman and Christian D. Wunsch and published in 1970. The basic idea is to build up the best alignment by using optimal alignments of smaller subsequences.

* 1. NEEDLEMAN-WUNSCH IN PRACTICE

This algorithm requires a (n+1 \* m+1) matrix; n and m being the size of the given sequences and one extra row and column for gap. The next task is to form a scoring matrix. The final step is trace backing.

Assume we want to align two sequences S and T, where

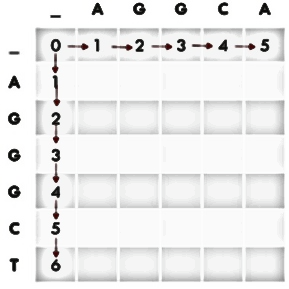
S = AGGGCT Gap penalty: 1

T = AGGCA Mismatch penalty: 2

The first step is placing the two sequences along the margins of a matrix and initializing the matrix cells.

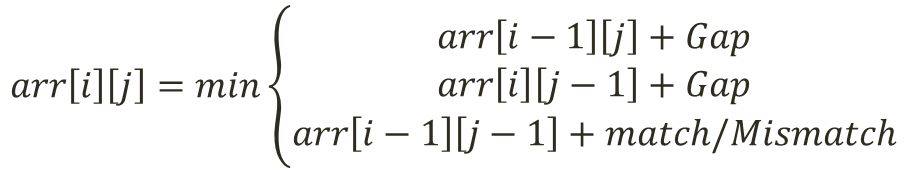
To initialize we assign a 0 to the first entry in the matrix and then fill in the first row and column based

on the incremental addition of gap penalties, as in Figure below. Although the algorithm could fill in the first row and column through iteration, it is important to clearly define and set boundaries on the problem. An arrow to determine where the value of a cell is coming from is widly used, it comes in handy in the later part of this procedure.

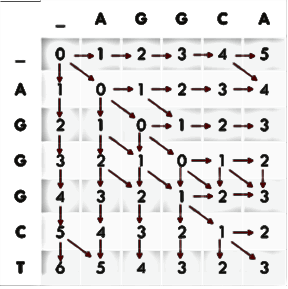


The next step is iteration through the matrix. The algorithm proceeds either along rows or along columns, considering one cell at time. For each cell three scores are calculated, depending on the scores of three adjacent matrix cells (specifically the entry above, the one diagonally up and to the left, and the one to the left). The minimum score of these three possible tracebacks is assigned to the entry and the

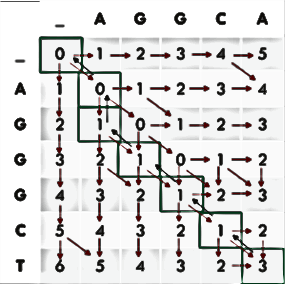
corresponding pointer is also stored.



The resultant matrix is as follows:



The final step of this algorithm is trackbacking. This step is as the name suggests, we have to trace back from the last cell to the first cell through the arrow/pointers that lead us to the last cell in the last step. Along the way we push the constituent characters in the cell into two stacks made for two sequences. If the value of a cell is coming from the previous diagonal cell, both characters should be pushed, if it is coming from the top/adjacent cell a gap has to be pushed in the stack instead of character to the constituent sequence’s stack

.

Sequence 1: AGGGCT

Sequence 2: A\_GGCA

With minimum penalty 3

* 1. ALGORITHMS

The array of the scoring matrix is denoted as arr

Stacks are denoted as stack1, stack2

The algorithm for scoring matrix is as following:

1. *function SCORING-MATRIX()*
2. *arr[0][0] := 0*
3. *for i := 0 to T-1*
4. *arr[0][i] := i\*Gap*
5. *end for*
6. *for i := 0 to S-1*
7. *arr[i][0] := i\*Gap*
8. *end for*
9. *for i:=1 to S-1*
10. *for j:=1 to T-1*
11. *MIN = min(arr[i][j-1] + Gap,*

*arr[i-1][j] + Gap,*

*arr[i-1][j-1] + match/mismatch)*

1. *arr[i][j] = MIN*
2. *end for*
3. *end for*
4. *end function*

The algorithm for trackbacking is as following:

1. *function TRACEBACKING(i, j)*
2. *if(i == 0 and j == 0)*
3. *return;*
4. *end if*
5. *if(arr[i][j] == arr[i-1][j-1] + match/mismatch)*
6. *stack1.push(sequence1[i-1])*
7. *stack2.push(sequence2[j-1])*
8. *TRACEBACK(i-1, j-1)*
9. *end if*
10. *else if(arr[i][j] == arr[i-1][j] + Gap)*
11. *stack1.push(sequence1[i-1])*
12. *stack2.push(‘ ’)*
13. *TRACEBACK(i-1, j)*
14. *end else if*
15. *else if(arr[i][j] == arr[i][j-1] + Gap)*
16. *stack1.push(‘ ’)*
17. *stack2.push(sequence2[j-1])*
18. *TRACEBACK(i, j-1)*
19. *end else if*
20. *end function*
    1. TIME COMPLEXITY

Time complexity of the entire program is O(S\*T).

* 1. PROOF OF OPTIMAL SUBSTRUCTURE

It can be observed from an optimal solution, for example from the given sample input, that the optimal solution narrows down to only three candidates.

1. xm and yn.

2. xm and gap.

3. gap and yn.

We can easily prove by contradiction. Let X - xm be X' and Y - yn be Y'. Suppose that the induced alignment of X', Y' has some penalty P, and a competitor alignment has a penalty P\*, with P\* < P.

Now, appending xm and yn, we get an alignment with penalty P\* + pxy < P + pxy. This contradicts the optimality of the original alignment of X, Y.

Hence, proved.

1. ACKNOWLEDGEMENT

Due to limitations of time, in depth understanding of this topic was not possible. Furthermore, the given topic is an advanced topic in Biology, thus understanding it thoroughly requires preconceived knowledge. As we are students of Computer Science, we do not possess such knowledge; As a result we had to work with surface lever information.

1. REFERENCES

* Carrillo, H. and Lipman, D. (1988). The Multiple Sequence Alignment Problem in Biology. *SIAM Journal on Applied Mathematics*, 48(5), pp.1073-1082.
* Web.mit.edu. (2018). [online] Available at: http://web.mit.edu/6.047/scribe\_notes/scribe\_notes\_local/2012-bak/Lecture02\_DynamicProgramming/Lecture02\_DynamicProgramming\_standalone.pdf [Accessed 24 Nov. 2018].
* Chaturvedi, A. (2018). *Sequence Alignment problem* [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/sequence-alignment-problem/ [Accessed 25 Nov. 2018].