Implementation of Needleman-Wunsch and Smith–Waterman Algorithm for Global and Local Alignment

By. Kashan Saeed

Introduction

In order to sequence DNA the strand is broken up randomly into relatively short strands which are then analyzed by a computer program. To assemble the sequence back together scientists look for identical regions and overlap them with one another. There are two ways that scientists can align sequences, global alignment, and local alignment. Global alignment attempts to align whole sequences with one another and works well when the sequences in the set are similar and roughly the same size. Local alignment identifies similar regions between sequences and is very useful for sequences that are very different from one another but are suspected to contain regions of similarity. To calculate global alignment we use the Needleman-Wunsch algorithm which was proposed in 1970, and for local alignment we use the Smith-Waterman algorithm first proposed in 1981. Both algorithms are quite similar with the exception that in Smith-Waterman algorithm instead of having negative scores we set those cells to zero. These algorithms can be done by hand but just to align two 100 sequence strands would take around 30,000 calculations. For that reason, scientists implement these algorithms in code which results in much faster, easier, and accurate results. In this project I implemented these algorithms using C++.

Method

I used C++ because it very efficient in terms of performance and memory. I intended to maintain good coding standards, reusability, and efficiency. The intent with this was to be able to give all the best global and local alignments. To implement the algorithm I created a matrix

made up of nodes, where each node held its own score, and a list of the nodes it's score came from. I then began with the first node with score 0 and calculated the whole matrix using the proper algorithm, depending on which alignment the user wanted. Upon calculating the whole matrix, for global alignment I began with the last node and recursively traced back to the first node, all the while keeping track of which nodes I had followed to get there. Whenever I had to choose between two different paths to trace back using, I would copy the trace back we have so far and continue recursing backwards along the two separate paths. Upon reaching the first node in the matrix the traceback's results were printed. For local alignment I looped through the matrix and found the highest scoring nodes, and from there traced back using the same technique I used for global alignment until I reached a node with a score of zero.

Results

I was able to get the alignments for all of the sequences that I tried. I crossed checked the results with online alignment tools and with sequences I aligned by hand. I was also able to get multiple alignments from the implementation if there was more than one alignment for the sequences that were being compared. The first three figures (Figures 1.1, 1.2, and 1.3) were tested for global alignment, and the second two (figures 2.1 and 2.2) were local alignments. As you can see the program also printed the matrices with the paths.

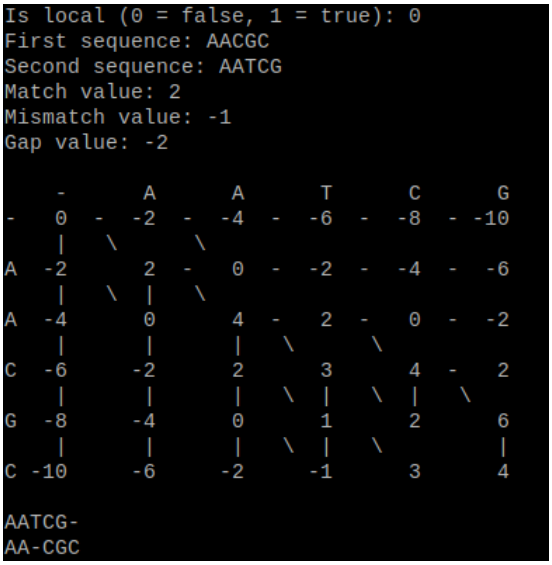

Figure 1.1

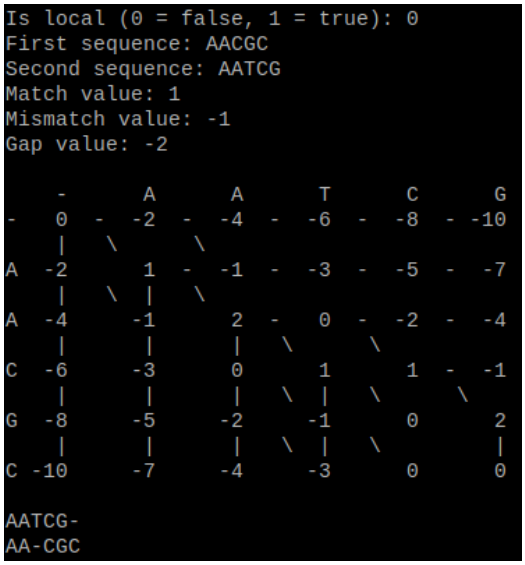

Figure 1.2

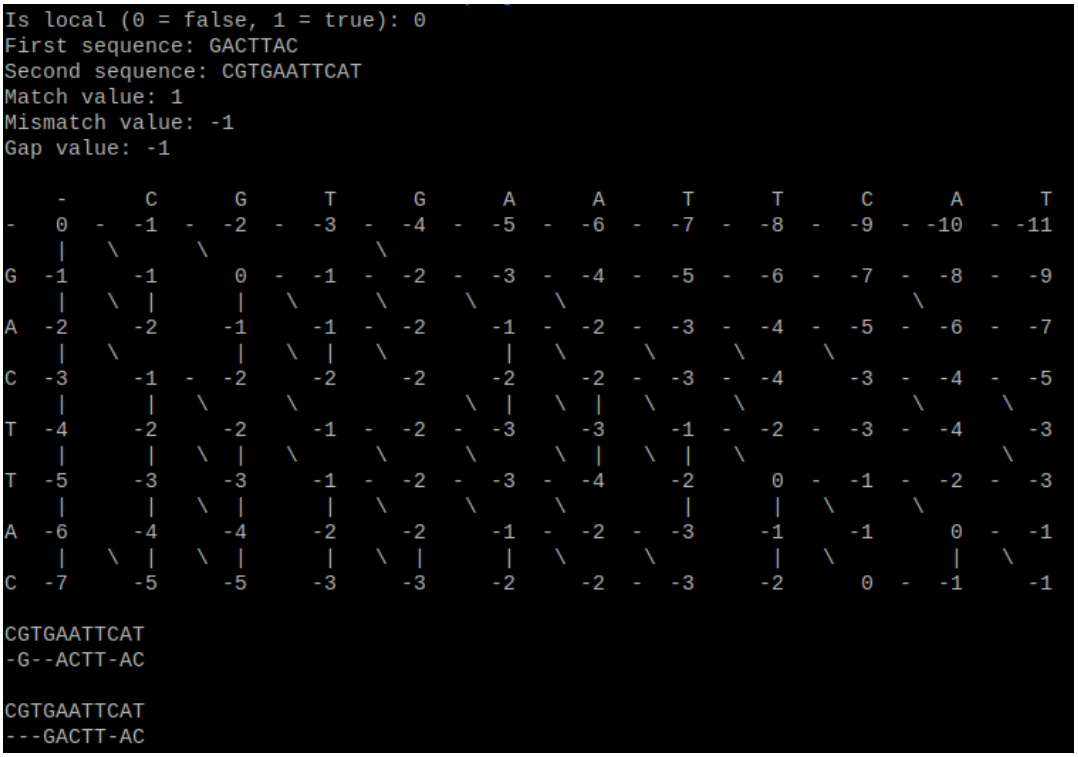

Figure 1.3

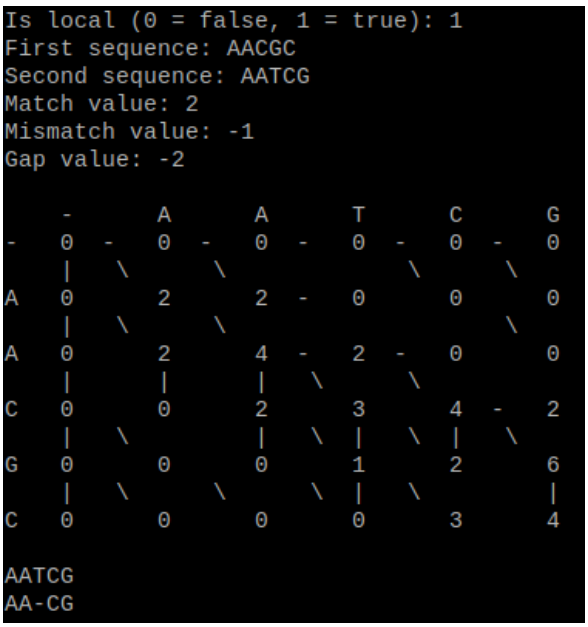

Figure 2.1

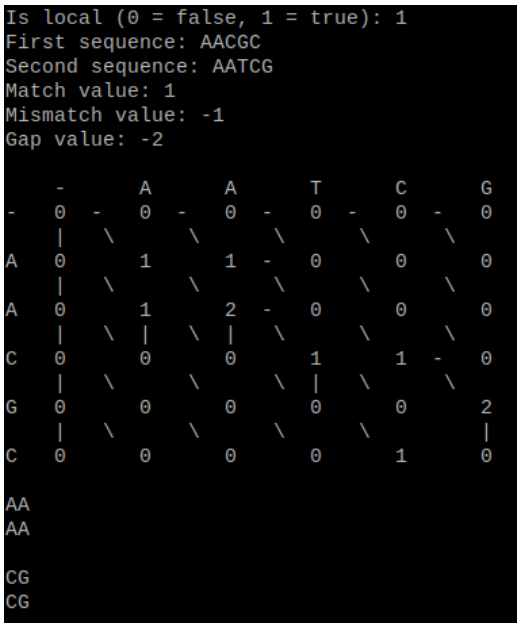

Figure 2.2

Discussion

The time complexity for this algorithm is theta(nm) where n is the length of first sequence and m is length of second sequence. There are some ways to improve the complexity but those will have to be in the scope of another project. Despite that, the global and local alignment algorithms work very well and are straightforward to implement. I have attached the source code, a demo file that when ran will allow you to input your sequence and values, and have packaged my code as a C++ library and attached that too. The program was written on Linux and as such the demo will not run on windows, more instructions are placed in the folder. I will send two folders, one a tar.gz so that you can open it on Linux if you intend and one as a zip in case you want to open it on windows.