Brendan Mikolajczyk, Eric Mustee, Matt Naso

CS 190

Dr. Victor Lee

2 May 2013

Final Project Report

For our final project, our group chose to implement the Needleman-Wunsch and Smith-Waterman sequence alignment algorithms in Java using the processing environment. The processing environment allows for us to create flexible user interface elements as well as easily manipulatable graphics to display to the user. (Some of the code that we used in order to create the buttons and checkboxes was code that had been reused from Eric’s summer research project using the processing environment). This allowed for us to use a graphical interface for the user to load their input sequences, as well as to view the output based on what the user wants. This allows for the user to manipulate the behavior of our program to suit their needs with a few simple button clicks.

Our program allows users to input their information in the form of text files. This information includes things such as the actual input strings (sequences) which the user wishes to align, as well as a substitution matrix such as BLOSUM or PAM to score the alignment. Our program also allows the user to select what type of alignment they would like our program to output (semi-global or global alignments) through the use of radio buttons. We originally had planned to also add an option for the user to choose local alignment (Smith-Waterman), however this idea didn’t make into the final version, more on that in the problems section. We have a button which allows the user to copy the output to the clipboard, allowing for them to easily and efficiently move the output into other places where they may need it. We have also included a button that allows the user to bring up a scoring table for the sequences so that the user can see a visual representation of the sequence alignment.

In order to verify the correctness and functionality of our program we had to conduct tests comparing our program to other real world tools for sequence alignment. We used global alignment with the PAM250 substitution matrix loaded when running the experiments. One real world tool we compared our program to was LALIGN. We compared the two by inputting the same sequences and comparing the results obtained from our program with the results obtained from LALIGN to make sure they matched. Upon viewing the results obtained from the two programs, we found that our programs output nearly matched the output of the LALIGN tool; both found 832 exact matches. Clustal-W and Clustal-Omega both yielded 832 exact matches as well. Our results from using these external programs have been included in this zip.

We added some additional features originally not planned for our application. One feature not planned was the ability to see the scoring matrix. This feature was implemented using the Java Swing toolkit. A JTable in a JFrame forms the scoring matrix. The scoring matrix may be opened or closed in an external window. Another additional feature added is the ability to copy the output of the program to the clipboard. This method of outputting results is very easy, the user simply clicks the clipboard icon and the sequence alignments are placed on the user’s clipboard, ready to paste into application of their choice. The user can scroll through the aligned sequence with the scroll wheel or by dragging the data. The data from our experiments can be found in the experiments folder in the included zip file.

While attempting to implement these algorithms for sequence alignment, we were able to get everything to work correctly except for the fact that we don’t have local alignment as stated earlier. The problem with this is that in order to add the ability to do local alignments we would have to add affine gap penalties which we ran out of time for, therefore we decided not to include local alignment in our program and just refined the global alignment and semi-global alignment. Other than this we really didn’t run into any other major problems. For bugs that remain in our program, the only other bug we were able to find was that at first when the user used the button to copy the output it wouldn’t copy the entire output, only part of it. However we fixed this bug so it is no longer an issue. We have also unsuccessfully attempted to break our program multiple times which has led us to believe that we do not have any more bugs in our program.

To run our application, simply double click the included jar. To view the source for our project, run eclipse, and switch to the java perspective. Go to File > Import, and select existing project into workspace. Select import archive file and import the EclipseProject.zip. The project will be named “CS190-Final\_Project” in the workspace, and can be renamed after it is imported. We advise starting at the SequenceAligner class. All the actual scoring occurs near the end of the Scorer class. To run the application from eclipse, make sure to run it is a java application and not applet. The version of processing we used was included in the zip folder, and the necessary library is named core.jar. If there are issues with the project, make sure core.jar is included in the build path (although this should not happen if the project was imported using the steps described in this paragraph).