Marcos Valerio marcosv2
Sebastian Valerio valerio4
Github Link: https://github.com/svalerio274/CS466-Final-Project.git

For this project we chose to implement the Needleman-Wunsch aligner with support for affine gap penalties. The global alignment problem involves finding an alignment for 2 strings v and w with a maximum score according to a scoring function δ. The Needleman-Wunsch algorithm performs global alignment in O(mn) time and space. However, it doesn't take into account that consecutive gaps are more common in real world data due to slippage errors. Adding affine gap penalties will make many separated gaps cost more than consecutive gaps.

First, we implemented the Needleman Wunsch algorithm without affine gap support. We implemented this using a 2d list of tuples to store both the score and the backpointer at each location. This algorithm ran in O(nm) time and space. Next, we modified this algorithm to work with affine gaps by modifying the recurrence. We used a 3d list for the insertion, deletion, and match/mismatch recurrences. We added a cost ρ for switching between these recurrences, making separate gaps cost more. We stored tuples with the score and backpointers at each element in the list like before. This algorithm ran in O(nm) space and O(nm) time.

To test our Needleman-Wunsch implementation with a ρ value of 0, we wrote 3 sequences of varying lengths and compared the resulting alignments of our function with the alignments from an online calculator. All 3 tests passed. To test the affine gap penalties, we wrote a test_affine function that writes DNA sequences of a specified length and number. It prints out the number of continuous gaps of all resulting alignments with a ρ of 0 and a ρ or 10 to show by how much it decreases. We ran it with 100 sequences of length 1000 and found that a ρ of 0 gave 23588 continuous gaps and a ρ of 10 gave 1024. This is a difference of 22,564 or about 23.035 times less. This shows that our implementation can successfully find alignments that prioritize continuous gaps.

In conclusion, we were successfully able to implement the Needleman Wunsch algorithm with support for affine gap penalties. Overall, our algorithm passed the tests that we wrote. However, our code could have benefitted from more thorough testing in order to better test the output of the code. We were unable to find a reliable way to quickly and accurately test large amounts of long outputs. Furthermore, one possible future improvement could be to improve the

space usage of the algorithm. The algorithm could use something similar to the Hirschberg algorithm to use less space. This would make it easier to calculate alignments for very long strings. Finally, another possible future improvement would be to add more parameters to the function. This could allow for more customizability and flexibility.

Sources:

https://bioboot.github.io/bimm143_W20/class-material/nw/
Used to test our NeedlemanWunsch algorithm