Implementation of Global Alignment Algorithms
Final Project
CS466 - Intro to Bioinformatics

Group Members

Rayna Patel & Sreenidhi Vijayaraghavan

Link to GitHub Repository: https://github.com/sreenidhi2002/Global_Alignment_Algorithms

Introduction

Global alignment allows for the comparison of sequences of DNA, RNA, or proteins to identify similarities between them. This process is used in bioinformatics for various evolutionary and functional analyses. Genomic sequencing utilizes global alignment by aligning entire genomes for evolutionary studies and can allow researchers to identify mutations. Using global alignment for evolutionary research gives insight into the relationships between different species and how genetic changes occurred over time. Phylogenetics also implements global alignment to construct phylogenetic trees and identify relationships between species to aid in their classification.

For this project, our group chose to implement the Needleman-Wunsch and Hirschberg alignment algorithms to compare their performance in terms of both time and memory used. The Needleman-Wunsch algorithm takes a dynamic programming approach and performs global alignment in O(m*n) time and space where m and n represent the lengths of the two aligned sequences. The Hirschberg algorithm is an optimization of global alignment algorithms and reduces the space being used by implementing a divide and conquer approach which lowers the space complexity to O(n) which allows for very large sequences to be aligned efficiently as well.

Using Python 3 functionalities, we compare the performances in terms of time and memory of the two algorithms by running the algorithms on randomly generated DNA and protein sequences of various lengths.

Methods

The computational objective of global alignment is to find an optimal and accurate alignment with the highest score. In implementing this algorithm, the balance between computational speed, memory usage, and accuracy was explored and compared for two global alignment methods: Needleman-Wunsch and Hirschberg.

To facilitate the experimentation and evaluation of the Needleman-Wunsch and Hirschberg algorithms, a random sequence generator was developed to create input sequences for the alignment processes. The generator produced sequences composed of the four nucleotide

bases—adenine (A), thymine (T), cytosine (C), and guanine (G) based on the input length desired by the user. The length of the generated sequences, denoted as m and n, represented the lengths of the two sequences being aligned.

The Needleman-Wunsch algorithm is a dynamic programming algorithm that aligns two sequences to maximize their overall similarity. The algorithm assigns scores to match, mismatch, and gap penalties and calculates the optimal alignment based on these scores. The implementation of this algorithm is referenced from the algorithm worked on in HW1 of CS 466 in the Fall 2023 semester. The global_align_needleman_wunsch function implements the Needleman-Wunsch algorithm for global sequence alignment. It initializes a scoring matrix M and a backpointer matrix pointers. The matrix has dimensions (m+1) x (n+1), with m and n being the lengths of the two sequences being aligned. The matrices are filled based on dynamic programming, considering match, mismatch, and gap penalty scores. The first row and first column were initialized with cumulative gap penalties and for each cell in the matrix, the score was calculated based on the maximum of the match score, mismatch score, and gap penalty scores. The optimal alignment score is obtained, and the traceback_global function is used to retrieve the actual alignments. The function returns the aligned sequences and the alignment score. The traceback_global Function performs traceback to reconstruct the optimal alignment from the back pointer matrix. It starts from the bottom-right corner of the matrix and follows the path of maximum scores, considering different movements (UP, LEFT, TOP LEFT) to handle matches, insertions, and deletions back to the top-left corner. The aligned sequences are constructed during this process, and the result is returned. Needleman-Wunsch was used as a baseline to compare with the results, runtime, and memory usage of Hirschberg.

The Hirschberg algorithm is a space-efficient approach for global sequence alignment. This algorithm utilizes a divide-and-conquer strategy that reduces the memory requirements from quadratic to linear space. To implement this algorithm, the problem was divided into smaller subproblems by finding the middle column of the dynamic programming matrix used in traditional alignment algorithms and reducing the problem into two halves. The algorithm then recursively solves the two subproblems created in the division step until the sequences are short enough to be aligned directly (base case). Lastly, the solutions to the subproblems are combined to obtain the final alignment. This involves determining the optimal alignment in the middle column, which serves as a "pivot" for combining the solutions from the two halves. Our code implementation was divided as follows: The get_prefix_suffix function is a helper function for the Hirschberg algorithm. It calculates the scores for either prefix or suffix alignments using dynamic programming. It considers gap penalties and matching scores based on the input sequences. The combine_alignments function combines the results of recursive calls in the Hirschberg algorithm. It concatenates the left and right alignments obtained from the recursive calls. The main Hirschberg algorithm function starts by checking a base case, using the

Needleman-Wunsch algorithm when the sequences are short. Otherwise, it calculates the midpoint of the sequences and recursively applies Hirschberg to the left and right halves. The results are combined using the combine_alignments function. This divide-and-conquer approach optimizes memory usage, reducing it from quadratic to linear space.

These algorithms collectively provide a comprehensive approach to global sequence alignment, with Needleman-Wunsch serving as a benchmark and Hirschberg offering a more memory-efficient alternative.

We expect the following insights from running the two algorithms:

If the Hirschberg algorithm consistently demonstrates significantly lower execution times than Needleman-Wunsch, particularly for larger inputs, it implies that the compromise for reduced memory usage in Hirschberg is translating to heightened computational efficiency in practical scenarios. Additionally, Persistent lower memory usage in Hirschberg across diverse input sizes suggests a successful reduction in space complexity.

Results:

Figure 1: Execution Time vs Input Size for NW and Hirschberg Algorithms

We ran test cases with varying sequence lengths to compare the performance of the Hirschberg algorithm with the Needleman-Wunsch algorithm (our baseline). Given its dynamic

programming nature and O(m×n) time complexity (where ‘n’ and ‘m’ are the lengths of the sequences being aligned), the execution time of the Needleman-Wunsch algorithm is expected to grow quadratically with increasing input sizes. As we thought, the Needleman-Wunsch baseline algorithm took increasingly longer to run as the size of the input sequence increased. This aligns with the expected runtime of the Needleman-Wunsch baseline algorithm. What was found for execution time is that Hirschberg surprisingly performed worse than the Needleman-Wunsch baseline algorithm- the longest time taken for Hirschberg was 72000ms, and for the Needleman-Wunsch algorithm we found the largest time recorded in the same run to be 30000ms for the input sequences of length 4000.

While both the Needleman-Wunsch and the Hirschberg algorithms have an Expected Time Complexity of O(m×n), one possible reason that might have caused a higher time being noted for the Hirschberg algorithm is the multiple recursive calls that might have led to longer execution times in practice, especially for larger input sizes.

We had originally anticipated Hirschberg to exhibit nearly equal execution times due to its reduced space requirements and identical time complexity. However, the magnitude of the difference may vary based on the specific characteristics of the input sequences. For example, we considered the case of input structure leading to longer execution times in Hirschberg. If the input sequences have a structure that does not align well with the divide-and-conquer strategy of Hirschberg, it could result in less favorable performance. There may have also been other strategies for optimization within our implementation of the Hirschberg algorithm that may have led to longer execution times. For example, the number of variables we are storing and copying may lead to more time taken for the code to execute. Identifying and fixing any such optimized code should result in lower execution times for the Hirschberg algorithm.

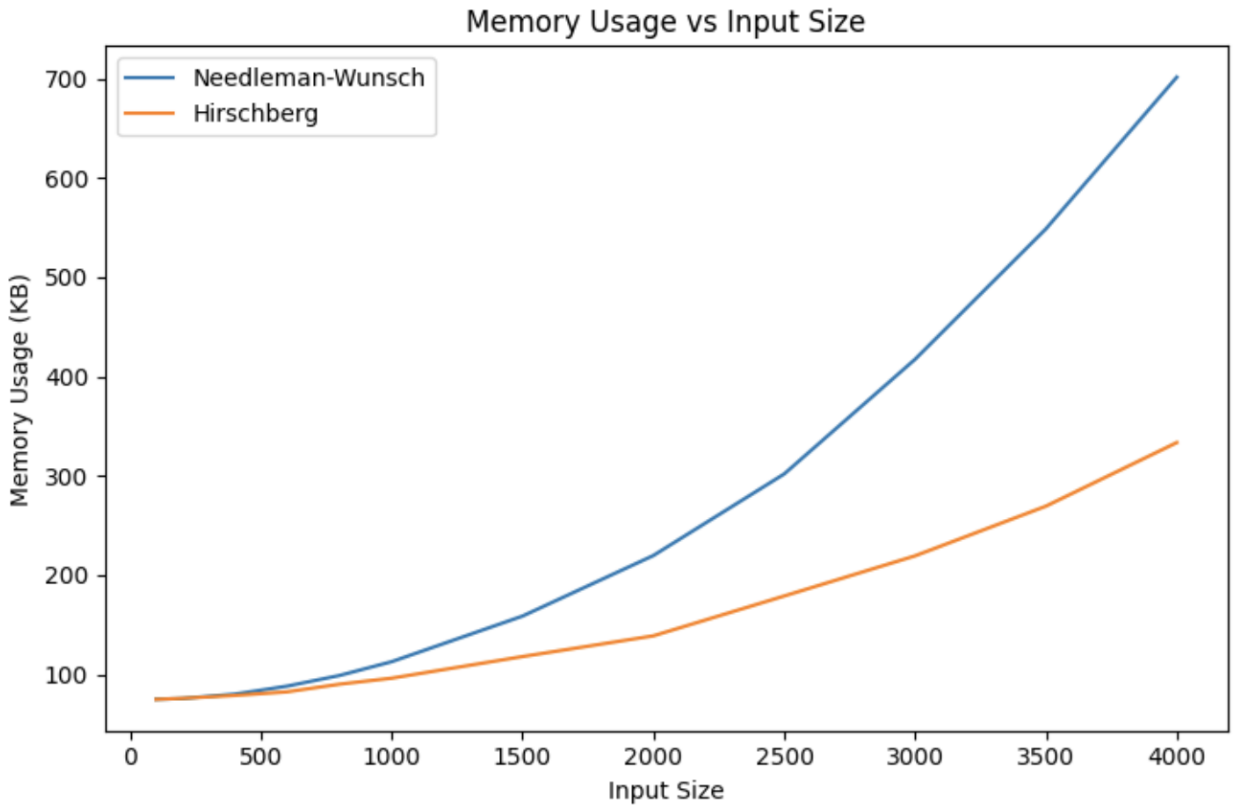

Figure 2: Memory Usage vs Input Size for NW and Hirschberg Algorithms

For memory usage, the results are as we had hypothesized. While the baseline Needleman-Wunsch algorithm keeps increasing quadratically as a function of input size and uses over 2500KB of memory, the Hirschberg algorithm performs very well. This is in line with the space-time complexity as the Needleman-Wunsch algorithm requires O(m×n) space to store the scoring matrix, making it less memory-efficient, especially for long sequences. Meanwhile, the Hirschberg algorithm was expected to use significantly less memory as it had a reduced space complexity of O(min(m,n)), making it well-suited for long input sequences. Overall, since the Hirschberg algorithm consistently exhibited lower memory usage than the Needleman-Wunsch algorithm across the various input sizes tested, this displays that the Hirschberg algorithm truly does present a more space-optimized approach for alignment and is successful in reducing space complexity.

Conclusion

The algorithm implemented and published for public use will make the global alignment of genetic sequences easy for researchers and users to perform. By using Needleman-Wunsch as a baseline function to compare with the results of the Hirschberg algorithm, the accuracy of the global alignment and scoring was confirmed, and a comparison of the time and memory used by the algorithms for sequences of different input lengths demonstrated a much more space-efficient approach by the Hirschberg algorithm thanks to its divide-and-conquer strategy. One possible area of improvement for our project would be to re-optimize our code implementation of the Hirschberg algorithm to get rid of any unnecessary function calls or variables that might be causing the algorithm to run slower than we had initially expected. By performing these optimizations, we would be able to present a highly efficient version of the Hirschberg that not only beats the baseline Needleman-Wunsch Algorithm in its space efficiency but also performs well in terms of the time taken for it to align long sequences.

Overall, we hope that the public finds these implementations useful and that our project has helped not just us, but also others online learn more about the process of global alignment in Bioinformatics, and that this work inspires others to do projects that build upon these algorithms.

References

Jones, N. C. & Pevzner, P. A. (2004), An Introduction to Bioinformatics Algorithms , MIT Press , Cambridge, MA .