REED COLLEGE
Department of Computer Science

RESEARCH PROJECT
Optimizing RNA sequencing local alignment algorithm

Name of Candidate: Daksh Shami

Date of Submission: April 28, 2021

1. Project Introduction:

In this project, I am analyzing RNA sequencing using the Smith Waterman Algorithm. RNA sequencing is one of the algorithmic challenges in computational biology, and has quite a lot of real world implications, including in vaccine development for COVID-19. In 1981, Temple Smith and Michael Waterman developed a dynamic programming algorithm for local alignment of string sequences (in contrast to the global alignment Needleman-Wunsch algorithm). Local alignment is more applicable to study for RNA as the primary purpose of studying the sequences is to make inferences about their local homology. As such, we not only study Smith-Waterman algorithm, but also attempt to make it computationally feasible by parallelizing it and running it on a Reed's patty machine.

2. Smith-Waterman Algorithm: Overview

Smith-Waterman algorithm (in concept) starts by generalizing deletions and insertions (markers of mutation) as "gaps". The substitutions are seen as mismatches. However, to evaluate how these gaps affect the common matching sequence, the algorithm needs an appropriate gap-penalty scheme. While the gap penalty scheme can be any function that appropriately marks gaps and matches differently, it should be computationally inexpensive. After this, the scoring matrix is initialized, with number of rows being 1+length of first sequence, and the column being 1+length of second sequence. All elements of the first row and first column are initialized to zero.
Now scoring begins. We go from left to right, top to bottom and consider the insertions and deletions (non-diagonal entries which mark the gaps) and substitutions (diagonal entries). If no score is greater than 0, then the element in that row/column is 0, since 0 is the least possible value in Smith Waterman Algorithm (unlike Needleman-Wunsch which can have negative values).
After going through the scoring, start at the highest scoring entry and recursively traceback until 0 is encountered. The segment with highest similarity score is generated as output.

3. Smith Waterman Algorithm: Pseudocode

Let the two sequences be $A = a_1a_2\cdots a_m$ and $B = b_1b_2\cdots b_n$. Let the similarity score be $s(a_i, b_i)$ such that

$$s(a_i, b_i) = \begin{cases} 1 & a_i = b_i \\ -1 & a_i \neq b_i \end{cases}$$

Initialize a $(m+1) \times (n+1)$ matrix (let's call it M) such that the first row and first column are initialized to 0. Each gap of length k has penalty W_k. Thus each entry in M is defined as follows:

$$M_{ij} = max \begin{cases} 0 & \text{This is the minimum score required} \\ H_{i-1,j-1} + s(a_i, b_j) & \text{This is the score of aligning } a_i \text{ and } b_j \\ max_{k \geq 1}\{H_{i-k,j} - W_k\} & \text{If } a_i \text{ is at the end of gap of length k} \\ max_{l \geq 1}\{H_{i,j-l} - W_l\} & \text{If } b_j \text{ is at the end of gap of length l} \end{cases}$$

After this we initiate traceback starting at the highest scoring entry in M to an entry that is 0 to get the best local alignment based on the source of entry.
The code for C++ implementation of the sequential part is attached with the document.

4. Optimizing Smith-Waterman algorithm

It's worth noting that while Fork-Join is not the most efficient way of parallelizing the algorithm, it is the most natural way in which I can translate a sequential code into a highly efficient parallel one (which will use a synchronous barrier). The approach here will be similar to that of Sample Sort. Let's suppose that we have P workers. The initialization of the matrix is a fairly inexpensive process which can be done by just one processor without any overhead. However, the computationally intensive part is computing the matrix values. We can optimize it by noticing that row maximum and column maximum are calculated for each entry in the matrix, resulting in extra $O(mn(m+n))$ work. This can be avoided by changing the entry algorithm as follows: let the previous row sum be denoted by $R_{i,j}$ and previous column sum be denoted by $C_{i,j}$. Thus we have

$$R_{i,j} = -W_k + Max \begin{cases} R_{i,j-1} \\ H_{i,j-1} - W_k \end{cases}$$

$$C_{i,j} = -W_k + Max \begin{cases} C_{i-1,j} \\ H_{i-1,j} - W_k \end{cases}$$

$$H_{i,j} = Max \begin{cases} (H_{i-1,j-1} + s(a_i, b_j) \\ R_{i,j} \\ C_{i,j} \\ 0 \end{cases}$$

Thus the classic way to parallelize this algorithm in a way that all processes can sequentially go through the matrix is to go through all the anti-diagonals in the matrix.

5. Citations

1. O. Gotoh, An Improved Algorithm for Matching Biological Sequences, J Mol Biol 162 (1982) 705-708