PROJECT REPORT

A two-base encoded DNA sequence alignment problem in computational biology

PROBLEM PRESCRIBED BY

DR. CLAUDIA RANGEL-ESCARENO

NATIONAL INSTITUTE OF GENOMIC MEDICINE, MEXICO

Contents

1 Participants

This team consists of eleven students, two faculty members and one industry researcher.

STUDENTS		
Chiaka Drakes	Simon Frazer Univ. British Columbia	cdrakes@sfu.ca
Daniel Salazar	UC Santa Barbara	dsalazar@math.ucsb.edu
Deidrey Langat	Univ. of Massachusetts, Lowell	tamitiny@yahoo.com
Erya Huang	Claremont Graduate University	erya.huang@gmail.com
Hem Wadhar	UC Los Angeles	hwadhar@math.ucla.edu
Jose Pacheco	CSU Long Beach	Sgacho@gmail.com
Joseph mcGrath	Univ. of Massachusetts, Lowell	Jmcgrathee@hotmail.com
Man Vu	CSU Long Beach	mhvu@hotmail.com
Mark Morabito	Univ. of Massachusetts, Lowell	mjmorabito@gmail.com
Nancy Rodriguez	UC Los Angeles	nrodriguez@math.ucla.edu
Qin Wu	West Virginia University	qinwu@math.wvu.edu
FACULTY		
Jen-Mei Chang	CSU Long Beach	jchang9@csulb.edu
Rao Vemuri	UC Davis	rvemuri@ucdavis.edu
INDUSTRY		
Pat McLaughlin	Lockheed Martin	pat.mclaughlin@lmco.com

1.1 Daily Summary Reports

Tuesday, July 28, 2009	Hem & Chiaca
Wednesday, July 29, 2009	Mark
Friday, July 30, 2009	Joe, Qin, Man, Jose, Deidrey, Jen-Mei

1.2 Participation

Sequence Alignment Code	Hem, Qin, Joe, Chiaca, Emily and Mark
Pseudo-code	Qin
Two-color base Algorithm	Everybody
Parallel Approach	Man, Jose, and Jen-Mei
Presentation slides	Danny, Nancy, Emily, Hem, Mark, Deidrey, Man, Jose, and Jen-Mei
Report	Claudia, Jen-Mei, Man, Hem, and Qin

2 Overview

Computational methods in the field of biology have become a key factor since the advent of the human genome project in 1990. Since then many other genomes have been sequenced, generating a wide variety of sequence analysis problems. The sequencing of the human genome and the HapMap project have impacted the study of human disease in significant ways and enabled many genome-wide association studies that aim to elucidate the genetic component of complex diseases. The recent introduction of instruments capable of producing millions of DNA sequence reads in a single run is rapidly changing the landscape of genetics, providing the ability to answer questions with heretofore unimaginable speed.

2.1 Introduction

DNA sequences are strings of letters from a four-letter alphabet called nucleotides (**A, C, G, T**). The length of a sequence is variable and sometimes we require the alignment of lengthy and highly variable or extremely numerous sequences. Hence, constructing algorithms to produce high-quality sequence alignments using four letters becomes a real challenge. In general, computational approaches to sequence alignment are classified as either global or local alignments. By global alignment, we consider aligning the entire scope of all query sequences against a reference sequence. On the other hand, the method of local alignment identifies isolated regions of high similarity within the entire sequence, which makes the technique a better choice in some situations but a more complex one in general.

A variety of computational algorithms have been constructed for the sequence alignment problem. Pair-wise sequence alignment methods are used to find the best-matching piecewise (local or global) alignments of two

query sequences. They are efficient to calculate and are often used for methods that do not require extreme precision. Three common methods of producing pair-wise alignments are dynamic programming, dot-matrix methods, and word methods. Multiple sequence alignment is an extension of pair-wise alignment that incorporates more than two sequences at a time but can also align pairs of sequences. One of the major challenges in the alignment problem is that not all sequences are of the same length. For instance, DNA sequences can have substitutions (change of one nucleotide for another), insertions and deletions (gain or loss of one or more nucleotides) and therefore algorithms should include the possibility of gaps. There are some biological assumptions about the start and the end of a sequence that are useful for algorithm development. However, there could be gaps at these positions as well. The SOLiD^{TM} next generation sequencing platform has implemented a color-based approach leading to a two-base encoded type of data. The two-base encoding scheme means that data is first collected in color space, in which the color provides information about two adjacent bases to ease error detection and hence detecting SNPs with higher accuracy. Color-based data must then be decoded into sequence data.

For the purpose of this study, we consider two dynamical programming techniques that are most frequently used in the literature. Namely, the Needleman-Wunsch [4] and Smith-Waterman [5] algorithms for global and local alignment, respectively.

2.2 Objective

The primary objective of the sequence alignment problem is to search for a new algorithm that facilitates the use of two-base encoded data for large-scale re-sequencing projects. This algorithm should be able to perform local sequence alignment as well as error detection and correction in a reliable and systematic manner, enabling the direct comparison of encoded DNA sequence reads to a candidate reference DNA sequence. In the current report, we will first briefly review two well-known sequence alignment approaches in Sections 3.1 and 3.2 and provide a rudimentary improvement for implementation on parallel systems in Section 3.4. In Section 4, we carefully examined a unique sequencing technique known as the SOLiD^{TM} System that can be implemented followed by the results from the global and local sequence alignment. Finally, we summarize our findings in Section 5.

3 Sequence Alignment

We start with the most basic sequence analysis task: Are two sequences related? DNA sequences are strings from a 4-letter alphabet of nucleotides: adenine (**A**), thymine (**T**), guanine (**G**), and cytosine (**C**). Let us denote a DNA sequence as $X = \{x_1, x_2, ..., x_n\}$ and assume that a nucleotide x occurs at random with probability q_x, independent of all other nucleotides in the sequence. Then the probability of the occurrence of such a sequence of length n is

$$\prod_{i=1}^{n} q_{x_i}$$

Other kinds of biological sequences are the protein sequences which are made of a 20-letter alphabet of amino acids – alanine (ala, **A**), arginine (arg, **R**), asparagine (asn, **N**), aspartic acid (asp, **D**), cysteine (cys, **C**), glutamine (gln, **Q**), glutamic acid (glu, **E**), glycine (gly, **G**), histidine (his, **H**), isoleucine (ile, **I**), leucine (leu, **L**), lysine (lys, **K**), methionine (met, **M**), phenylalanine (phe, **F**), proline (pro, **P**), serine (ser, **S**), threonine (thr, **T**), tryptophan (trp, **W**), tyrosine (tyr, **Y**), valine (val, **V**). Two sequences are strongly related if the number of matched positions after alignment is greater than a learned threshold value and weakly related if otherwise.

It is worth noting that biological considerations need to be taken into account in determining whether a match happens at random or by a sound relationship. In a very general framework the process includes three major steps [3].

1. **Alignment method**. Given two sequences, what is the best way to align them against one another?
2. **Scoring system, i.e., ranking**. How to assess the quality of any given alignment?
3. **Statistical analysis**. Can the alignment be explained by chance, or is it due to a shared history between the sequences, e.g., homology, relatedness or pure coincidence?

In summary, the purpose of sequence alignment is to measure sequence similarity with an appropriate form of metric and this can be done either globally or locally. In short, a *global alignment* method seeks a region of high similarity in the entire sequence scope while a *local alignment* method searches for segregated regions of high similarity. The regions of low similarity are denoted as gaps in both cases. In the subsections that follow, we will compare and contrast the difference between the two methods with Needleman & Wunsch [4] and Smith & Waterman [5] Algorithms.

3.1 Global Alignment

Let us start by defining two terms. A *mutation* is seen as a mismatch in the sequence alignment and *indels* (**in**sertions and **del**etions) are either insertions or deletions in either sequence. These are represented by gaps in one of the sequences.

Given two sequences $X = \{x_i\}_{i=1}^{n}$ and $Y = \{y_j\}_{j=1}^{m}$ with lengths n and m, respectively, let c be the total length of the alignment we have, i.e.,

$$max(n,m) \leq c \leq n+m.$$

The alignment is represented by a $2 \times c$ matrix, $A(X,Y)$, where the rows are the sequences and the columns are matches and indels.

Example: Let X = {G A A T T C A G T T A} and Y = {G G A T C G A}. Here $n = 11$ and $m = 7$ and one possible arrangement can be defined as

$$A(\mathrm{X},\mathrm{Y}) = \begin{matrix} G & A & A & T & T & C & A & G & T & T & A \\ G & G & A & - & T & C & - & G & - & - & A \end{matrix}$$

It is clear that there are different ways of arranging the sequences and find mismatches and indels, so we are interested in the best possible arrangement. This can be done with a *scoring function* and it can be defined as follows. Let x_i and y_i be the symbols appearing in the i^{th} position of their respective query sequence in a given alignment. A scoring function between x_i and y_i is denoted $\sigma(x_i, y_i)$. Notice that x_i and y_i could possibly be gaps. This one-to-one correspondence gives rise to a final score of

$$G = \sum_{i=1}^{c} \sigma(x_i, y_i).$$

There are numerous ways to define σ depending on the applications of interest. For instance, a simple scoring function can be defined in terms of penalties and awards such as

$$\begin{cases} -1, & \text{indels and mismatches} \\ +1, & \text{matches.} \end{cases}$$

Thus, for all $x_i \neq y_i$, $\sigma(-, y_i) = \sigma(x_i, -) = \sigma(x_i, y_i) = -1$ and $\sigma(x_i, y_i) = 1$ for all $x_i = y_i$. Notice that this definition of the scoring function does not guarantee a unique alignment. Furthermore, the "best" alignment

$\sigma(\cdot,\cdot)$	action
match	$+1$
mismatch	0
gap	-1

Table 1: An example scoring function.

depends on the choice of the scoring function and the values assigned to the penalties and awards. Any biological constraint can be translated to an appropriate scoring function. Conversely, any choice of the scoring function corresponds to some biological interpretation of the results.

3.1.1 Needleman-Wunsch Algorithm

The Needleman-Wunsch algorithm performs a global alignment on two query sequences and is used widely in bioinformatics to align protein or nucleotide sequences. We will briefly describe the steps of the algorithm next.

Given two sequences $X = \{x_i\}_{i=1}^{n}$ and $Y = \{y_j\}_{j=1}^{m}$, their similarity matrix, G, will be of size $(n+1)\times(m+1)$. Entries in the first row can be found according to the following rule

$$G_{1,j} = \sum_{k=1}^{j} \sigma(-, y_k) \tag{1}$$

while entries in the first column can be found according to the rule

$$G_{i,1} = \sum_{k=1}^{i} \sigma(x_k, -) \tag{2}$$

For example, if we follow the assignment in Table 1,

then Figure 1 a) shows the result of the similarity matrix after the first step.

Next, we compute each entry from the upper left corner using the following recursive relation

$$G_{i,j} = \max \begin{cases} G_{i-1,j} + \sigma(x_i, -) & \text{gap in sequence } Y \\ G_{i,j-1} + \sigma(-, y_j) & \text{gap in sequence } X \\ G_{i-1,j-1} + \sigma(x_i, y_j) & \text{match or mismatch} \end{cases}$$

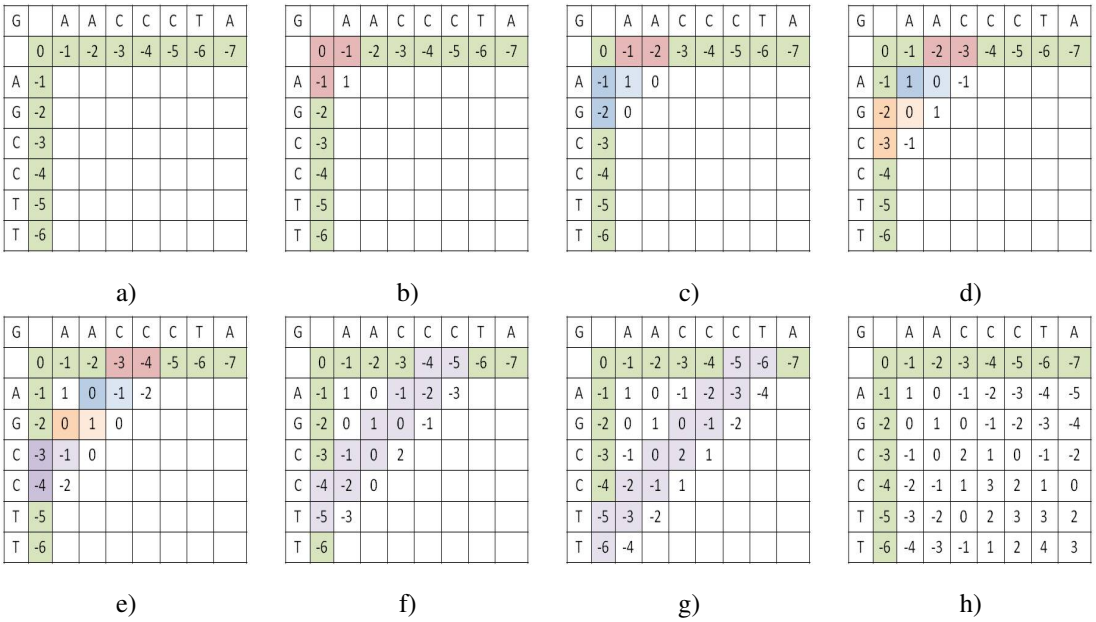

Figure 1: An iterative procedure for calculating a similar matrix between the sequences **AACCCTA** and **AGC-CTT** using Needleman-Wunsch algorithm and the scoring function described in Table 1.

Figures 1 b), c), d), e), f), g) show a sample of results at the selected iteration while Figure 1 h) gives the final similarity matrix, G, based on the penalty and award criteria in Table 1.

Once the G matrix is computed, note that the maximum score for any alignment resides in the lower right hand corner of the matrix. The last step in the sequence alignment problem is to extract the alignment information by tracing the similarity score in the decreasing fashion. To do this, one can start from the bottom right cell, and compares the value with the three possible sources (its immediate left, immediate top, upper left diagonal) to see which it came from. A general rule of thumb is to follow the diagonal path whenever possible in the case of a tie. If the next step lands on an upper left diagonal cell, then x_i and y_j are aligned, if it is the immediate left cell, then x_i is aligned with a gap, and in the case of immediate top, y_j is aligned with a gap. Note that several choices may have the same value in general, leading to alternative optimal alignments. In the example used above, if we trace back the highlighted path in Figure 2, then the result of the alignment (with a score of 3) is given below.

A A C C C T A
A G C C T T –

G		A	A	C	C	C	T	A
	0	-1	-2	-3	-4	-5	-6	-7
A	-1	1	0	-1	-2	-3	-4	-5
G	-2	0	1	0	-1	-2	-3	-4
C	-3	-1	0	2	1	0	-1	-2
C	-4	-2	-1	1	3	2	1	0
T	-5	-3	-2	0	2	3	3	2
T	-6	-4	-3	-1	1	2	4	3

Figure 2: The highlighted entries on the diagonal traced from lower right hand corner (high similarity score) to upper left hand corner (low similarity score) describes how to align the two sequences.

3.2 Local Alignment

Global alignments, which attempts to align every entry in the query sequences, are most useful when the sequences in the query set are similar and of roughly equal size. On the other hand, local alignments are more useful for dissimilar sequences that are suspected to contain regions of similarity. With sufficiently similar sequences, there is no difference between local and global alignments. See, for example, Table 2 for an illustration of this point.

Adopting the notations used in Section 3.1, we will describe next a general local alignment technique constructed by Temple Smith and Michael Waterman in 1981 [5].

3.2.1 Smith-Waterman Algorithm

Like the Needleman-Wunsch algorithm, Smith-Waterman [5] is a dynamic programming algorithm that has the desirable property that it is guaranteed to find the optimal local alignment with respect to the scoring system being used. The main difference between Smith-Waterman and Needleman-Wunsch algorithms is that negative scoring matrix cells are set to zero, thus allowing a "fresh start" in the search. Back tracing starts at

Global	**F**	**T**	**F**	**T**	**A**	**L**	**I**	**L**	**L**	**A**	**V**	**A**	**V**
	|			|	|	|		|	|	|		|	|
	F	–	–	**T**	**A**	**L**	–	**L**	**L**	**A**	–	**A**	**V**

Local	**F**	**T**	**F**	**T**	**A**	**L**	**I**	**L**	**L**	–	**A**	**V**	**A**	**V**
			|	|	|	|		|	|		|	|	
	–	–	**F**	**T**	**A**	**L**	–	**L**	**L**	**A**	**A**	**V**	–	–

Table 2: An illustration of global and local alignments. Notice the gappy quality of global alignment when sequences are sufficiently dissimilar.

the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment.

Given two sequences $X = \{x_i\}_{i=1}^{m}$ and $Y = \{y_j\}_{j=1}^{n}$ (e.g., X = AACCCTA and Y = AGCCTT), the following recursive formula gives the scoring matrix that is then used to find the optimal alignment between X and Y.

$$L_{i,j} = \max \begin{cases} L_{i-1,j} + \sigma(x_i, -) & \text{gap in sequence } Y \\ L_{i,j-1} + \sigma(-, y_j) & \text{gap in sequence } X \\ L_{i-1,j-1} + \sigma(x_i, y_j) & \text{match or mismatch} \\ 0 & \text{to allow a “fresh start”} \end{cases}$$

With this definition and the scoring function given in Table 1, the entire first row and first column of the similarity matrix, L, will attain a value of zero. Figures 3 b), c), d), e), f), g) show a sample of results at the selected iteration while Figure 3 h) gives the final similarity matrix, L, based on the penalty and award criteria in Table 1.

Once the L matrix is computed, note that the maximum score for any alignment resides in the lower right hand corner of the matrix. To extract the alignment information, one starts from the bottom right cell, and

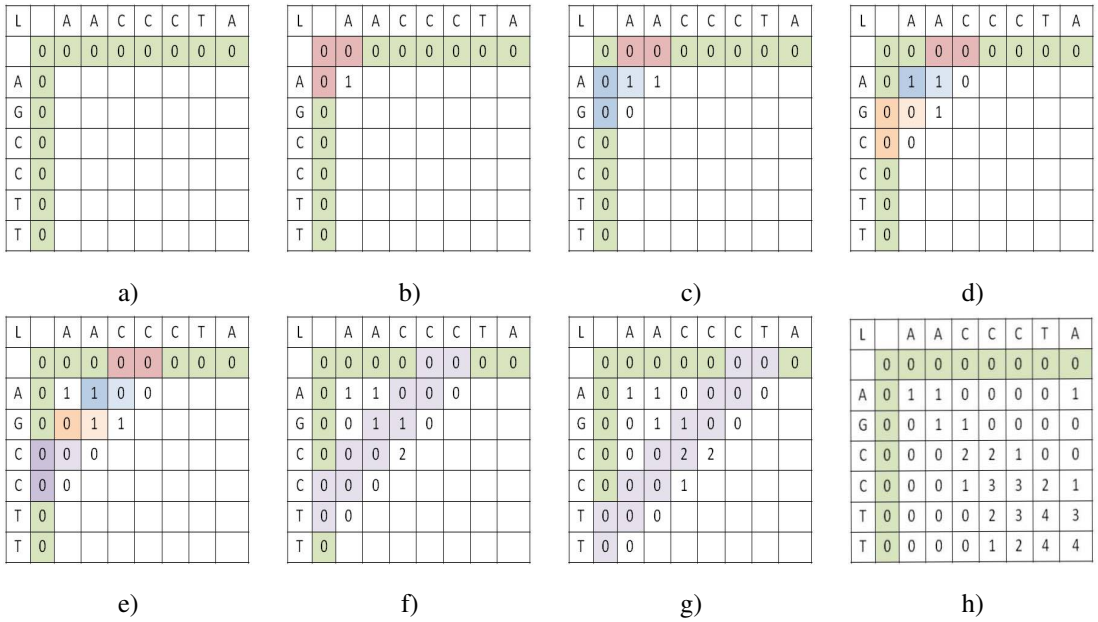

Figure 3: An iterative procedure for calculating a similar matrix between the sequences **AACCCTA** and **AGC-CTT** using Smith-Waterman algorithm and the scoring function described in Table 1.

compare the value with the three possible sources (its immediate left, immediate top, upper left diagonal) to see which it came from. A general rule of thumb is to follow the diagonal path whenever possible in the case of a tie. If the next step lands on an upper left diagonal cell, then x_i and y_j are aligned, if it is the immediate left cell, then x_i is aligned with a gap, and in the case of immediate top, y_j is aligned with a gap. If we trace back the highlighted path in Figure 4, then the result the alignment (with a score of 3) is given below.

A A C C C T A
| | | | | | |
A G C C – T T

L		A	A	C	C	C	T	A
	0	0	0	0	0	0	0	0
A	0	1	1	0	0	0	0	1
G	0	0	1	1	0	0	0	0
C	0	0	0	2	2	1	0	0
C	0	0	0	1	3	3	2	1
T	0	0	0	0	2	3	4	3
T	0	0	0	0	1	2	4	4

Figure 4: The result of a local sequence alignment between **AACCCTA** and **AGCCTT**. The highlighted entries on the diagonal traced from lower right hand corner (high similarity score) to upper left hand corner (low similarity score) describes how to align the two sequences.

3.3 Implementation

Let $X = x_1x_2...x_M$ be a reference sequence and $Y = y_1y_2...y_N$ be a test sequence whose similarity matrix has the following structure

$$\begin{bmatrix} & x_1 & x_2 & x_3 & \cdots & x_M \\ y_1 & & & & & \\ y_2 & & & & & \\ \vdots & & & & & \\ y_N & & & & & \end{bmatrix}.$$

Furthermore, let $\sigma(l_1, l_2)$ be the scoring function (matrix) between l_1 and l_2. In the following pseudo codes, A gives all possible pairs of the best global (or local) alignments of the sequence X and Y. Specifically, the first column of A gives the re-ordering of X while the second column of A gives the re-ordering of Y after alignment; S is the similarity matrix calculated based on σ; D is the direction matrix that contains information about how to trace back.

3.3.1 Global Alignment

Function: $[A] = GlobalAlignment(X, Y, \sigma)$

Description: Given the weighting matrix, and two sequences X and Y, find all possible pairs of the best global alignments of the sequences X and Y.

Pseudo code:

Step-1. Find the score matrix S and direction matrix D.

$[S, D] = GlobalScore(X, Y, \sigma)$

Step-2. Get A (all possible pair of alignments of the sequence X and Y).

$[A] = GlobalTraceBack(X, Y, S, D)$

Function: $[S, D] = GlobalScore(X, Y, \sigma)$

Description: Given the weighting matrix, and two sequences X and Y, find the global similarity matrix and the global direction matrix for the best score alignment.

Pseudo code:

Step-1. Initialization.

M=length of the sequence X

N=length of the sequence Y

$S_{0,0} = 0$

Step-2. Initialize the first row of the score matrix.

for $j = 1 : N$

$S_{0,j} = S_{0,j-1} + \sigma('-', y_j)$,

save the trace back direction($'LEFT'$) into $D_{0,j}$,

end

Step-3. Initialize the first column of the score_matrix

for $i = 1 : M$

$S_{i,0} = S_{i-1,0} + \sigma(x_i, '-')$,

save the trace back direction($'UP'$) into $D_{i,0}$,

end

Step-4.

for $i = 1 : M$

for $j = 1 : N$

$s\_diag = S_{i-1,j-1} + \sigma(x_i, y_j)$;

$s\_left = S_{i,j-1} + \sigma('-', y_j)$;

$s\_up = S_{i-1,j} + \sigma(x_i, '-')$;

$S_{ij} = max(s\_diag, s\_left, s\_up)$;

save all possible trace back directions which give the maximum value $S_{i,j}$ into $D_{i,j}$

end

end

Function: $[A] = GlobalTraceBack(X, Y, S, D)$

Description: Given two sequences X and Y and corresponding direction matrix, find all possible pairs of the best global alignments of the sequences X and Y.

Pseudo code:

Step-1. Initialization.

$x\_alignment = ''$; $y\_alignment = ''$;

$x\_index = N$; $y\_index = M$;

$A = [x\_alignment,\ y\_alignment]$;

$A\_index = [x\_index,\ y\_index]$;

Step-2. While there exist rows in A_index such that $A\_index_{t,1} > 0$ or $A\_index_{t,2} > 0$, repeat Step-3; otherwise, STOP.

Step-3. For $i = 1$: number of rows in A, do the following

$x\_alignment = A_{i,1}$; $y\_alignment = A_{i,2}$;

$x\_index = A\_index_{i,1}$; $y\_index = A\_index_{i,2}$;

if $x\_index = 0$ && $y\_index > 0$,

add y_index copies of $'-'$ at the beginning of $x\_alignment$;

copy $y_1 ... y_{y\_index}$ to the beginning of $y\_alignment$;

$A_{i,:} = [x\_alignment,\ y\_alignment]$;

x_index_0; $y\_index\_n = 0$

if $x\_index > 0$ && $y\_index = 0$,

add $x_index$copies of $'-'$ at the beginning of $y\_alignment$;

copy $x_1 ... x_{x\_index}$ to the beginning of $x\_alignment$;

$A_{i,:} = [x\_alignment,\ y\_alignment]$;

x_index_0; $y\_index\_n = 0$

if $x\_index > 0$ && $y\_index > 0$,

d =all trace back directions in D_{x_index, y_index}

$num\_direction$=total number of trace back directions in d;

for the first trace back direction (say d_1), find the letters be added to alignment of X and Y, and also find the x index and y index of next alignment step by the function $TraceOneBit()$.

$[x\_letter, y\_letter, x\_index\_n, y\_index\_n]$

$= TraceOneBit(x_{x\_index}, y_{y\_index}, x\_index, y\_index, d_k)$;

append x_letter to the beginning of the $x\_alignment$;

append y_letter to the beginning of the $y\_alignment$;

$A_{i,:} = [x\_alignment,\ y\_alignment]$;

for $k = 2 : num\_direction$,

$[x\_letter, y\_letter, x\_index\_n, y\_index\_n]$

$= TraceOneBit(x_{x\_index}, y_{y\_index}, x\_index, y\_index, d_k)$;

add x_letter to the beginning of the $x\_alignment$;

add y_letter to the beginning of the $y\_alignment$;

$A = [A;\ x\_alignment,\ y\_alignment]$; (add one more row to the Alignment matrix A).

$A\_index = [A\_index;\ x\_index\_n, y\_index\_n]$;

end

Function:

$[x\_letter, y\_letter, x\_index\_n, y\_index\_n]$

$= TraceOneBit(x, y, x\_index, y\_index, direction)$;

Description: Given current location (x_index_n, y_index_n) of the similarity matrix S and corresponding x, y letters, find the letters be added to alignment of X , Y, and the x index and y index of the score matrix S for the next alignment step.

Input:

x: letter at the given index of the sequence X;

y: letter at the given index of the sequence Y;

x_index: index in x direction of the current step;

y_index: index in y direction of the current step;

$direction$: one direction of the alignment at the give spot;

Output:

x_letter: letter to be added to the alignment of X;

y_letter: letter to be added to the alignment of Y;

x_index_n: index in x direction for next alignment step;

y_index_n: index in y direction for next alignment step;

Pseudo code:

if ($direction =' LEFT'$)

$x\_letter =' -'$; $y\_letter = y$; $x\_index\_n = x\_index - 1$; $y\_index\_n = y\_index - 1$;

if($direction =' UP'$)

$x\_letter = x$; $y\_letter =' -'$; $x\_index\_n = x\_index$; $y\_index\_n = y\_index - 1$;

if ($direction =' DIAGONAL'$)

$x\_letter = x$; $y\_letter = y$; $x\_index\_n = x\_index - 1$; $y\_index\_n = y\_index - 1$;

if ($direction =' STOP'$)

$x\_letter = x$; $y\_letter = y$; $x\_index\_n = 0$; $y\_index\_n = 0$;

3.3.2 Local Alignment

Function: $[A] = LocalAlignment(X, Y, \sigma)$

Pseudo code:

Step-1. Find the score matrix S and direction matrix D.

$[S, D] = LocalScore(X, Y, \sigma)$

Step-2. Get A (all possible pair of alignments of the sequence X and Y).

$[A] = LocalTraceBack(X, Y, S, D)$

Function: $[S, D] = LocalScore(X, Y, \sigma)$

Description: Given the weighting matrix, and two sequences X and Y, find the local score matrix and the local direction matrix for the best score alignment.

Pseudo code:

Step-1. Initialization.

M=length of the sequence X

N=length of the sequence Y

$S_{0,0} = 0$

Step-2. Initialize the first row of the score matrix.

for $j = 1 : N$

$S_{0,j} = 0$,

save the trace back direction($'STOP'$) into $D_{0,j}$,

end

Step-3. Initialize the first column of the score_matrix

for $i = 1 : M$

$S_{i,0} = 0$,

save the trace back direction($'STOP'$) into $D_{i,0}$,

end

Step-4.

for $i = 1 : M$

for $j = 1 : N$

$s\_diag = S_{i-1,j-1} + \sigma(x_i, y_j)$;

$s\_left = S_{i,j-1} + \sigma('-', y_j)$;

$s\_up = S_{i-1,j} + \sigma(x_i, '-')$;

$S_{ij} = max(s\_diag, s\_left, s\_up, 0)$;

save all possible trace back directions which give the maximum value $S_{i,j}$ into $D_{i,j}$

end

end

Function: $[A] = LocalTraceBack(X, Y, D, S)$

Description: Given two sequences X and Y and corresponding direction matrix, find all possible pairs of the best local alignments of the sequences X and Y.

Pseudo code:

Step-1. Find all index pairs of the maximum value of the score matrix S. For each pair of the index (x_index, y_index), do Step 2 to Step 4:

Step-2. Initialization.

$x\_alignment = ''$; $y\_alignment = ''$;

$A = [x\_alignment,\ y\_alignment]$;

$A\_index = [x\_index,\ y\_index]$;

Step-3. While there exist rows in A_index such that $A\_index_{t,1} > 0$ and $A\_index_{t,2} > 0$, repeat Step-4; otherwise, STOP.

Step-4. For $i = 1$: number of rows in A, do the following

$x\_alignment = A_{i,1}$; $y\_alignment = A_{i,2}$;

$x\_index = A\_index_{i,1}$; $y\_index = A\_index_{i,2}$;

d =all trace back directions in D_{x_index, y_index}

$num\_direction$=total number of trace back directions in d;

for the first trace back direction (say d_1), find the letters be added to alignment of X and Y, and also find the x index and y index of next alignment step by the function $TraceOneBit()$.

$[x\_letter, y\_letter, x\_index\_n, y\_index\_n]$

$= TraceOneBit(x_{x\_index}, y_{y\_index}, x\_index, y\_index, d_k)$;

append x_letter to the beginning of the $x\_alignment$;

append y_letter to the beginning of the $y\_alignment$;

$A_{i,:} = [x\_alignment,\ y\_alignment]$;

for $k = 2 : num\_direction$,

$[x\_letter, y\_letter, x\_index\_n, y\_index\_n]$

$= TraceOneBit(x_{x\_index}, y_{y\_index}, x\_index, y\_index, d_k)$;

add x_letter to the beginning of the $x\_alignment$;

add y_letter to the beginning of the $y\_alignment$;

$A = [A;\ x\_alignment,\ y\_alignment]$; (add one more row to the Alignment matrix A).

$A\_index = [A\_index;\ x\_index\_n, y\_index\_n]$;

end

3.4 A Parallel Approach

With a simple test using a reference sequence of lengths 15, 20, and 25 and a test string of length 5, we notice that the time it takes to compute the sequence alignment grows exponentially, as illustrated in Table 3. With the computing resources available nowadays, we are well positioned to consider a parallel approach that capitalizes both time and resources. To this end, we describe our initial attempt to tackle this issue in

Reference Sequence Length	Global (sec)
15	25
20	726
25	7481

Table 3: A sample of computational time for a sequence alignment of various lengths with Needleman-Wunsch algorithm. These computations were done using MATLAB on a Windows workstation with 2GB RAM and a 2.4 GHz processor.

the following paragraphs. We remark that in this model, practical implementation issues are ignored for simplicity.

As stated before, these sequence alignments are computationally challenging. Each DNA sequence has approximately $2-3$ billion base pairs and the reference sequence is about several hundreds of thousands base pairs. In order to process a completed matrix, G or L, the required processing time is on the order of days. To reduce the computation time, we introduce a parallel processing method to split the computations amongst the many different computers available. It is worth noting that using this method, the number of computations remains the same while the computation time is reduced.

Let $M_{(n+1)\times(m+1)}$ be the matrix that we will compute using either the local (L) or the global (G) algorithm. Using Equations (1) and (2), we are able to define the first column $M_{i,1}$ and the first row $M_{1,j}$ of M. The remaining entries in the matrix will vary depending on the algorithm used. Figure 5 a) shows the $M_{9\times19}$ matrix with the first row and first column shaded in gray. The value, $\star$, is found using the chosen algorithm, G or L. Pictorially, $\star$ is dependent upon the upper, left, and upper-left values.

Without a parallel approach, computation of each subsequent value relies on the values of its three adjacent neighbors. However, once $\star$ is found, as illustrated in Figure 5 b), we can proceed to find the values to the right and below $\star$ simultaneously, as illustrated in Figure 5 c). Proceeding in this manner, we can simultaneously compute three cells in the next time stamp and four cells in the next stamp and so on, as shown in Figures 5 d), e), and f). Using this pattern to our advantage, we assign to each computer a column, as illustrated in Figure 6. The idea is to have one computer start off then update the information for the second computer to use. In the next time step, while the first computer works on the second value, the second computer will be working on the first value. On the third time step, the second computer will again update this information for

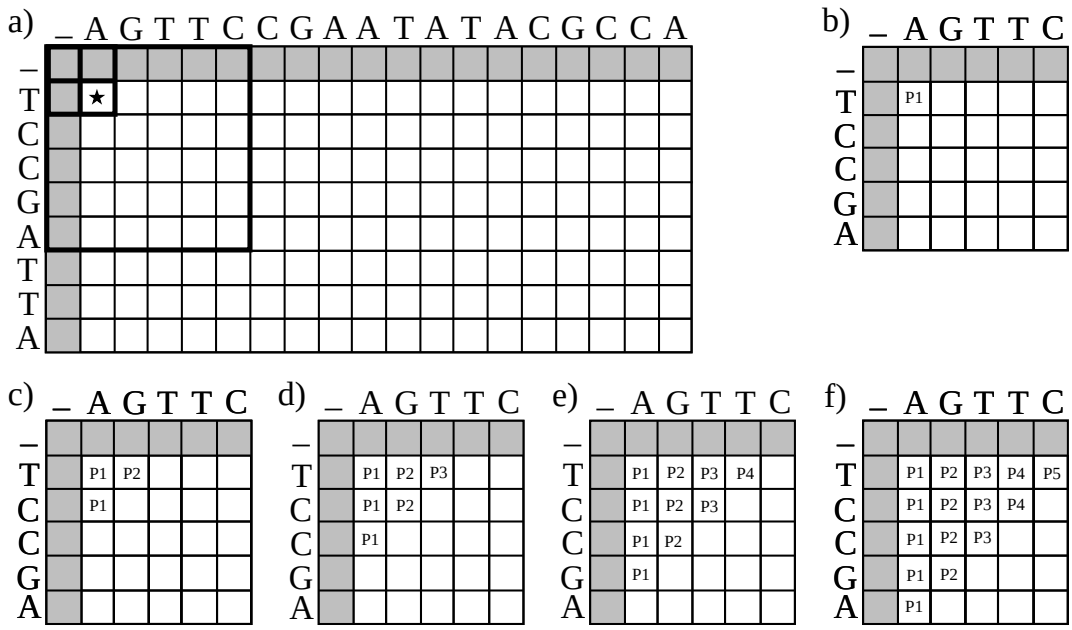

Figure 5: In a), the matrix, $M_{9\times 19}$, with the first column and row shaded in gray. These values are found using Equations (1) and (2) respectively. The value $\star$ can be found once the upper, left, and upper-left values, with respect to $\star$, are known. b) through f) are cropped regions within the bold lining of a). P1 through P5 represent the five different computers. The first time step is shown in b), the second in c) and so on until the fifth time step in f).

the third computer. At this time the first computer will be working on the third value, the second computer will be working on the second value and the third computer will be working on the first value and so on. This process is repeated until the maximum number of available CPUs is reached.

With this method, it is possible to find values horizontally, across the matrix, or vertically, down the matrix. However, when dealing with matrices that are extremely wide, the vertical direction is the optimal choice and vice versa. Let us consider the case of five computers and sequences of length 6 against 15, therefore a wide matrix. Furthermore, assume it takes one time unit (**tu**) to compute each element in M. If we compute horizontally, at some point the first five rows will be completed and there will only be one computer working on the last row. The time in which the number of computers working is less than maximum number of computers, we define that as the *lag time*. In this case, the lag time is 14 **tu**s but if we work vertically, the lag time is only 4 **tu**s.

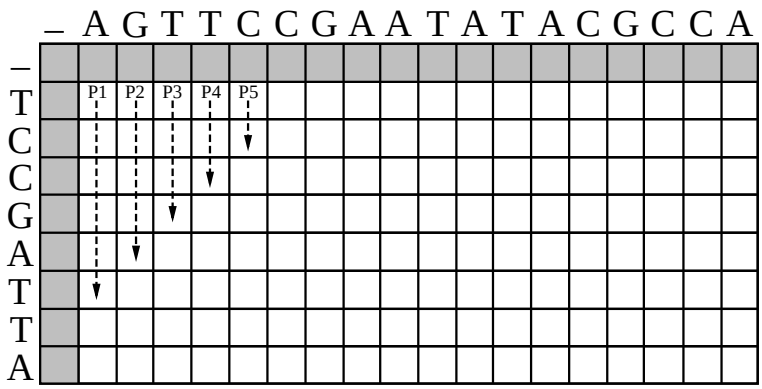

Figure 6: An $M_{9\times 19}$ matrix with the first column and first row shaded in gray. P1-P5 represents five distinct CPUs. The downward arrows show the subsequent pattern in which one computer will start right after the other.

We seek to compare the time it takes to compute the scoring matrix of two finite length sequences with and without parallelization. For a matrix $M_{n\times m}$, the regular approach will take nm seconds. As a side note, M initially is of dimension $(n+1)\times(m+1)$, but when counting the time we only worry about the area in the matrix which have not yet been determined, which is of dimension $n\times m$. To count the time for the parallel method, notice that in the first **tu** only one computer is working, in the second **tu** two computers are working, in the third **tu** three computers are working and so on until the maximum number of computers is reached. Towards the end, the same schematic, as previously described, occurs. The remaining elements within the matrix are all computed using the maximum number of computers as shown in Figure 7. With this in mind, we are able to derive Equation (3) which calculates the time it takes to calculate matrix G or L with the help of parallel computing. In this equation, f is dependent upon n, m, and k which represent the number of rows, columns, and computers respectively. However, Equation (3) only works under the assumption that k divides mn.

$$f(n,m,k) \quad = \quad 2(k-1)+\frac{mn-2\sum_{i=1}^{k-1} i}{k} \tag{3}$$

If this condition is not met, then the approximate amount of time it costs under this implementation is given by Equation (4). Notice that the terms in Equation (4) must satisfy the divisibility condition established

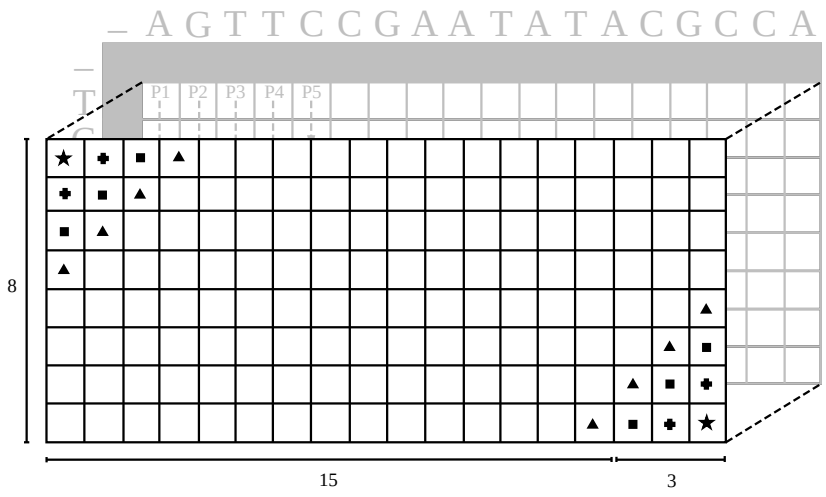

Figure 7: A similarity matrix of sequences of length 9 and 19 is first reduced to $M_{8\times 18}$ by eliminating the first row and first column. If the maximum number of computers is five and $\star$, $+$, ■, and ▲ represent the values calculated in the first, second, third, and fourth **tu**, then there will be five cells calculated at a time at each time step that follows until it reaches the end.

previously in Equation (3). Taking a closer look, we see that if m/k is not an integer then we can find a remainder, r, so that $k \mid m-r$ where Equation (3) can be applied. This leaves a portion of the matrix unaccounted. However, we can employ Equation (3) again to find the remaining time by forcing the number of computers to be r and now the number of column is also r so the limiting condition is met. The sum of these two terms as shown in Equation (4) will compute the minimum time it takes to process all of the elements.

$$f(n,m,k) \quad = \quad f(n,m-r,k)+f(n,r,r) \tag{4}$$

A few remarks are in order. If $k \mid mn$ then $k \mid n$ or $k \mid m$, the numerator in Equation (3) will always be positive because n,m is much greater than k, and we can force k to be r in Equation (4) because the remainder is never greater than the divisor. To better understand how these equations work, take an example where $n = 8, m = 18, k = 5$. We first compute $f(n,m-r,k) = f(8,15,5)$ using Equation (3). Then we calculate $f(n,r,r) = f(8,3,3)$ also using Equation (3). We can use Equation (3) in both cases because the limiting condition is met since $5 \mid 15$ and $3 \mid 3$. This results in $f(8,18,5) = 28+14 = 42$. Therefore, it takes roughly 42 **tu**s to complete this matrix via parallelization and $nm = 144$ **tu**s with the regular method. We see that our

solution is logical because with five computers the time it takes to compute the score for each element of the matrix using parallel computing should be roughly one-fifth of what it takes without parallelization plus the lags. In this case we calculated the maximum theoretical time so we expect it to be a bit more than one-fifth. As expected, the solution is approximately 29% of the regular time.

From a theoretical standpoint, this method can significantly speed up the processing time. However, the communication time between nodes is neglected in this framework. This issue, as well as the viability of the analysis, will be explored in a future investigation. In any regard, the method of parallel computing will significantly improve the efficiency of large-scale sequence alignment problems in computational biology.

4 Two-base Color Coding

Single nucleotide polymorphisms (SNPs) are single nucleotide variations of DNA base pairs that can be used as genetic markers in disease studies. Variations in the DNA sequences can also tell us about response to pathogens, chemicals, drugs, vaccines, and other agents. SNPs are also thought to be key enablers in realizing the concept of personalized medicine. Third generation DNA sequencing technology like Applied Biosystems SOLiD^{TM} , generates sequence data at a super large scale up to 4Gb in a single run. This creates computational challenges and one of the top ones is the sequence alignment process as it intends to map individual sequence read (lengths 25, 35 or 50) to a reference genome. The SOLiD^{TM} system uses a 2-base encoding algorithm that represents a single data point using two adjacent bases that are meant to detect SNPs with higher accuracy. A single nucleotide may be changed in which case we have substitution, or can be removed having a deletion or also added leading to insertions to the polynucleotide sequence.

4.1 Introduction

The SOLiD^{TM} system uses four fluorescent dyes to encode for the sixteen possible two-base combinations. This scheme represents DNA as a sequence of overlapping dimers, with each dimer corresponding to one of four colors based on a degenerate coding scheme. See Figure 8 for an illustration of this color-coding scheme.

A single strand of DNA can be sequenced using the color scheme by identifying the first base and then the subsequent colors. For our convenience, we introduce a table of numbers which are equivalent to the colors as demonstrated in Figure 9. Then, for instance, the sequence **ATCAAGCCTC** can be written as **A321023022**

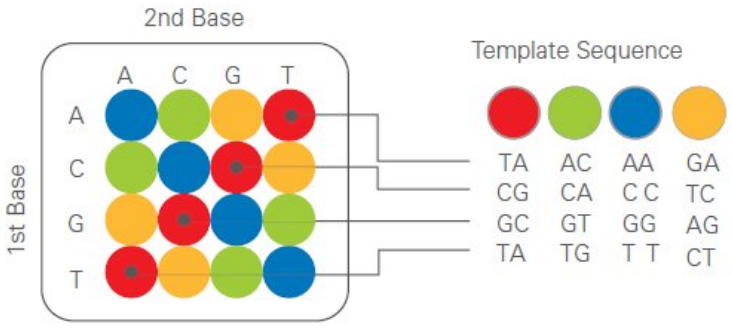

Figure 8: The colors are assigned by these rules for each dimer.

using the rules just described. Note that the identification of the first base is crucial, as using only a sequence of colors does not uniquely identify a strand of DNA.

		Second Base			
		A	C	G	T
First Base	A	0	1	2	3
	C	1	0	3	2
	G	2	3	0	1
	T	3	2	1	0

Figure 9: A numerical equivalence of Figure 8

4.2 Mathematics of Di-base Sequencing

Furthermore, we can think of the color as an operation on a previous base, which transforms it to the following base. Motivated by Figure 9, we observe the following rules of the transformation.

1. Color 0 is the identity transformation (Since $f_0(b) = b$ for every base b).
2. Color 1 interchanges A and C, G and T.
3. Color 2 interchanges A and G, C and T.

4. Color 3 interchanges A and T, C and G.

These operations define an Abelian group which is isomorphic to the Klein-4 group with addition table given by Table 4, since 0 is the identity element as $0 \oplus b = b$ for all $b \in \{0,1,2,3\}$, every element is its own inverse, $b \oplus b = 0$ for all b, and we have the following addition $1 \oplus 2 = 3, 2 \oplus 3 = 1$, and $3 \oplus 1 = 2$.

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Table 4: Addition rules for colors.

4.3 Application to SNPs Detection

The group structure of the colors is helpful in detecting true isolated single-base variants while identifying sequencing errors. To analyze this, we consider a simple example given in Figure 10.

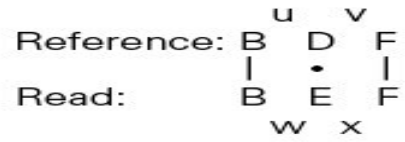

Figure 10: A typical SNP.

Here $D \neq E$ are mismatched bases. In order for the colors u, v, w, x to be consistent, they must satisfy the following properties.

c.1 $u \neq w$ since we have assumed that $D \neq E$.

c.2 $u \oplus v = w \oplus x$ since both uv and wx transform B to F.

c.3 $v \neq x$ follows from 1 and 2.

Thus, for a general SNP, we must have a color mismatch both before and after the base in question (c.1 and c.3) and using the addition rules specified in Table 4 we must have that $u \oplus v = w \oplus x$ (c.2). To make this point even clearer, we further examine a concrete example in Figure 11. Here the dot indicates the isolated single base variant (SNP). Notice that the transition before and after the SNP are unequal ($1 \neq 2$ and $3 \neq 0$, respectively) in the reference and read sequences, but we do have that $1 \oplus 3 = 0 \oplus 2 = 2$ which is the color that takes base **G** to base **A**.

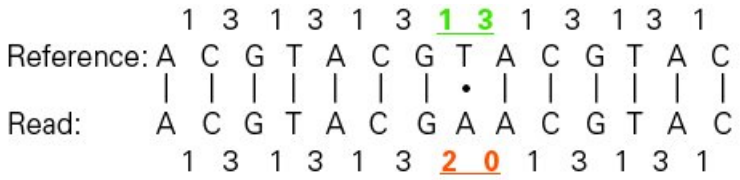

Figure 11: An illustration of how the Klein-4 algebra used to detect a SNP.

If one of these three conditions is not satisfied, then we have a sequencing error rather than a SNP. This process can be summarized generally with the following theorem.

Theorem. [2] *Let $c = c_1 c_2 \ldots c_k$ be a k-color substring of a read aligned with a the corresponding color reference $r = r_1 r_2 \ldots r_k$. Then c encodes an isolated $(k-1)$-base change if and only if the base position preceding c is not a variant, and the following two equations hold under the color addition table:*

$$\sum_{j-1}^{k} c_j = \sum_{j-1}^{k} r_j \tag{5}$$

$$\sum_{j-1}^{i} c_j \neq \sum_{j-1}^{i} r_j \qquad \forall i = 1,2,\ldots,k-1 \tag{6}$$

For interested readers, see [1, 2] for a detailed proof.

5 Conclusions

DNA and protein sequence alignment algorithms have been repeatedly studied in molecular and computational biology for a variety of reasons. There is yet a new motivation to re-visit such algorithms: next

generation sequencing technologies. In the past, two or more sequences not so different in length were compared for similarity analysis. However, a typical re-sequencing experiment using next-generation sequencing technology such as AB SOLiD^{TM} system generates millions of short sequence reads with an average length of 35 bases. These reads are expected to be aligned to a reference genome whose length is huge compared to the usual read length. The human genome reference sequence, for example, could be about 3.1 billion bases in length. Performing the alignment under these conditions needs to be adapted in such a way that we are still able to maximize a similarity score and to use a combination of weights for gaps and penalties, but doing so in a reasonable amount of time and with efficient use of computing resources.

In this report, the team presents an explanation of the algorithms for color space sequence data from the high-throughput re-sequencing technology and a theoretical parallel approach to the dynamic programming method for global and local alignment [5]. The combination of the di-base approach and dynamic programming provides a possible viewpoint for large-scale re-sequencing projects. We anticipate the use of distributed computing to be the next-generation engine for large-scale problems like such. Further investigation on ways to parallelize the problem will be highly valuable and beneficial.

6 Remarks

Every discussion, every brainstorm during our long daily sessions were productive and truly motivating. The enthusiasm of everyone in the team was contagious. The team exceeded the expectations regarding a possible solution to the problem. I would like to thank all of the students and senior members in this team. Particularly to Dr. Jen-Mei Chang for such an awesome job coordinating, organizing, and leading the 15 members of the team.

Claudia Rangel

7 Appendix

7.1 MATLAB Implementations

7.1.1 generateString.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% This code generates a sequence of length n (specified  %%
%% by user) of random choice of nucleotide A, C, G, and T.%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Outstr = generateString(n)

x = round(3*rand(n,1)) + ones(n,1)
%% This line can be commented out if user has a
%% pre-specified string
for i = 1:n
    if x(i) == 1
        Outstr(1,i) ='A';
    elseif x(i) == 2
        Outstr(1,i) ='C';
    elseif x(i) == 3
        Outstr(1,i) ='T';
    elseif x(i) == 4
        Outstr(1,i) ='G';
    end
end
```

7.1.2 L2N.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% This code convert the sequence X into its corresponding %%
%% numerals (A, C, G, and T).                              %%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = L2N(XL)

m = length(XL) ;
x = zeros(1,m);
for i = 1:m
    if XL(i) == 'A',
        x(i) = 1;
    elseif XL(i) == 'C',
        x(i) = 2;
    elseif XL(i) == 'T',
       x(i) = 3;
    elseif XL(i) == 'G',
        x(i) = 4;
    end
end
```

7.1.3 W.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% This code defines the penalty and award for match,    %%
%% mismatch, and gap between 2 sequences. f = match,     %%
%% g = gap, m1 = bad mismatch, m2 = good mismatch.       %%
%% This functon takes in (f,g,m1,m2) and returns a 5x5   %%
%% scoring matrix.                                       %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function weight = W(f,g,m1,m2)

weight(4,4) = f; weight(1,1) = f; weight(2,2) = f;
weight(3,3) = f; weight(5,5) = inf; weight(5,1) = g;
weight(5,2) = g; weight(5,3) = g; weight(5,4) = g;
weight(2,1) = m1; weight(3,1) = m2; weight(4,1) = m1;
weight(1,2) = m1; weight(1,3) = m2; weight(1,4) = m1;
weight(1,5) = g; weight(2,3) = m1; weight(2,4) = m2;
weight(2,5) = g; weight(3,2) = m1; weight(3,4) = m1;
weight(3,5) = g; weight(4,2) = m2; weight(4,3) = m1;
weight(4,5)=g;
```

7.1.4 TraceBack.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Inputs: seq_x:sequence in x direction, seq_y:sequence %%
%% in y direction direction: direction matrix. Outputs:  %%
%% tracecell: two columns, multiple rows, each row       %%
%% represent one alignment of seq_x and seq_y            %%
%% column#1-x alignment, column#2-y alignment.           %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [tracecell] = TraceBack(seq_x, seq_y, direction)

global X Y D
X = seq_x;
Y = seq_y;
D = direction;

xlen = length(X); ylen = length(Y);
tracecell = cell(1,2);
traceinf = zeros(1,3);
traceinf(1,1) = xlen;
traceinf(1,2) = ylen;
traceinf(1,3) = 1; %1-continue 0-stop

trace_num = 1; continue_num = 1;
while (continue_num > 0)
    clear new_tracecell;
    new_trace_num= 0 ;
    new_continue_num = 0;
    for curr_trace = 1:trace_num
        curr_tracecell_x = tracecell{curr_trace,1};
```

```
        curr_tracecell_y = tracecell{curr_trace,2};
        curr_traceinf = traceinf(curr_trace,:);
        [split_tracecell,split_traceinf,split_num,
        split_continue_num] = split_trace(curr_tracecell_x,
        curr_tracecell_y,curr_traceinf);
        new_continue_num = new_continue_num + split_continue_num;
        new_trace_num = new_trace_num + split_num;

        if curr_trace == 1
            new_tracecell = split_tracecell;
            new_traceinf = split_traceinf;
        else
            new_tracecell = [new_tracecell;split_tracecell];
            new_traceinf = [new_traceinf;split_traceinf];
        end
    end
    tracecell = new_tracecell;
    traceinf = new_traceinf;
    trace_num = new_trace_num;
    continue_num = new_continue_num;
end

function[split_tracecell,split_traceinf,
split_num,continue_num] = split_trace(curr_tracecell_x,
curr_tracecell_y,curr_traceinf);

global X Y D
split_num = 0;
continue_num = 0;
```

```
if (curr_traceinf(3) == 0 )
    tmp_tracecell = cell(1,2);
    tmp_tracecell{1,1} = curr_tracecell_x;
    tmp_tracecell{1,2} = curr_tracecell_y;
    split_tracecell = tmp_tracecell;
    split_num = split_num + 1;
    split_traceinf = curr_traceinf;
else
    x_id = curr_traceinf(1);
    y_id = curr_traceinf(2);
    x_i = X(x_id);
    y_j = Y(y_id);
    direction = D(y_id,x_id);
    Bin_D = dec2bin(D(y_id,x_id),4);
    direction = find(Bin_D == '1');
    num_direction = length(direction);
    split_tracecell = cell(num_direction,2);
    split_traceinf = zeros(num_direction,3);

    for k = 1:num_direction
        [x_bit,y_bit,x_id_p,y_id_p] =
        TraceOneBit(x_i,y_j,x_id,y_id,direction(k));

        split_tracecell{k,1} = strcat(x_bit,curr_tracecell_x);
        split_tracecell{k,2} = strcat(y_bit,curr_tracecell_y);

        split_traceinf(k,1) = x_id_p;
        split_traceinf(k,2) = y_id_p;
        split_num = split_num + 1;
```

```
        if ( x_id_p == 0 )
            split_tracecell{k,1} = ...
            strcat(repmat('-',1,y_id_p),split_tracecell{k,1});
            split_tracecell{k,2} = ...
            strcat(Y(1:y_id_p),split_tracecell{k,2});

            split_traceinf(k,3) = 0;
        else
            if ( y_id_p == 0 )
                split_tracecell{k,1} = ...
                strcat(X(1:x_id_p),split_tracecell{k,1});
                split_tracecell{k,2} = ...
                strcat(repmat('-',1,x_id_p),split_tracecell{k,2});
                split_traceinf(k,3) = 0;
            else
                split_traceinf(k,3) = 1;
                continue_num = continue_num + 1;
            end
        end
    end
end

function [x_bit,y_bit,x_id_p,y_id_p] = ...
TraceOneBit(x_i,y_j,x_id,y_id,direction);

D_DIAG = 3; D_UP = 4;
D_LEFT = 2; D_STOP = 1;

switch direction
    case D_UP
```

```
        x_bit = '-';
        y_bit = y_j;
        x_id_p = x_id;
        y_id_p = y_id-1;
    case D_LEFT
        x_bit = x_i;
        y_bit = '-';
        x_id_p = x_id-1;
        y_id_p = y_id;
    case D_DIAG
        x_bit = x_i;
        y_bit = y_j;
        x_id_p = x_id-1;
        y_id_p = y_id-1;
    otherwise
        error('wrong input');
end
```

7.1.5 SMAT.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Function SMAT takes in sequences X, Y, z for global or %%
%% local, f for the match score, g for the gap score, m1  %%
%% and m2 for the bad and good mismatch score. S is the   %%
%% output similarity matrix, D is the directional vector. %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [tracecell,S,D] = SMAT(X,Y,z,f,g,m1,m2)
m = length(X)+1;
n = length(Y)+1;

S = zeros(m, n); D = zeros(m-1, n-1);

weight = W(f,g,m1,m2);
Xnum = L2N(X);
% convert the sequence X into it's corresponding numerals
Ynum = L2N(Y);
% convert the sequence Y into it's corresponding numerals


%for global scoring matrix
if ( (z=='g') || (z=='G'))
    for i = 2:n  %populate the first column
        S(1,i) = (g)*(i-1);
    end
    for i = 2:m  %populate the first row
        S(i,1) = (g)*(i-1);
    end
    for i = 2:m
```

```
        for j = 2:n
            V = [S(i-1,j) + weight(Xnum(i-1),5),S(i-1,j-1) + ...
            weight(Xnum(i-1),Ynum(j-1)),S(i,j-1) +
            weight(5,Ynum(j-1))];
            S(i,j) = max(V);
            r1 = find(V == max(V));
            r2 = zeros(1,3);
            r2(1,r1) = 1;
            x11 = strcat(num2str(r2(1,1)),num2str(r2(1,2)),
            num2str(r2(1,3)));
            D(i-1,j-1) = bin2dec(x11);
        end
    end
elseif ( (z=='l') || (z=='L') )
    Xnum = L2N(X);
    % convert the sequence X into its corresponding numerals
    Ynum = L2N(Y);
    % convert the sequence Y into its corresponding numerals

    for i = 2:m
        for j = 2:n
            V = [S(i-1,j) + weight(Xnum(i-1),5), S(i-1,j-1) + ...
            weight(Xnum(i-1),Ynum(j-1)),S(i,j-1) +
            weight(5,Ynum(j-1)),0];
            S(i,j) = max(V);
            r1 = find(V == max(V));
            r2 = zeros(1,4);
            r2(1,r1) = 1;
            x11 = strcat(num2str(r2(1,1)),num2str(r2(1,2)),
            num2str(r2(1,3)));
```

```
            D(i-1,j-1)=bin2dec(x11);
        end
    end
end
tracecell = TraceBack(X,Y,D');
```

7.1.6 main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% This code outputs the time it takes to complete a sequence %%
%% alignment between reference sequences of length 15, 20, 25 %%
%% and a test sequence of length 5 using the local alignment. %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

count = 1;
for n = 15:5:25;
    reference = generateString(n);
    test  = generateString(5);
    tic;
    SMAT(reference, test, 'l', 1, 1, 0, 0);
    t(count) = toc;
    count = count + 1;
end
```

References

[1] Principles of di-base sequencing and the advantages of color space analysis in the SOLiD system. Technical Report Publication 139AP10-01, Applied Biosystems, 2008.

[2] A theoretical understanding of 2 base color codes and its application to annotation, error detection, and error correction. Technical Report Publication 139WP01-02, Applied Biosystems, 2008.

[3] Nello Cristianini and Matthew W. Hahn. *Introduction to Computational Genomics: A case studies approach*. Cambridge University Press, 2007.

[4] Saul Needleman and Christian Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[5] Temple Smith and Michael Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.