LOCAL ALIGNMENT ON HIGHLY UNBALANCED DNA SEQUENCE LENGTHS BY REDUCING SEARCH SPACE

by

Wilfredo Enrique Lugo-Beauchamp

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTING AND INFORMATION SCIENCE AND ENGINEERING

University of Puerto Rico
Mayagüez Campus

2016

Approved by:

Bienvenido Vélez, Ph.D Member, Graduate Committee	Date
Wilson Rivera-Gallego, Ph.D Member, Graduate Committee	Date
Emmanuel Arzuaga, Ph.D Member, Graduate Committee	Date
Belinda Pastrana, Ph.D. Graduate Studies Representative	Date
Jaime Seguel, Ph.D President, Graduate Committee	Date
Wilson Rivera-Gallego, Ph.D CISE Ph.D. Program Coordinator	Date

Abstract of Thesis Dissertation Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of DOCTOR OF PHILOSOPHY

LOCAL ALIGNMENT ON HIGHLY UNBALANCED DNA SEQUENCE LENGTHS BY REDUCING SEARCH SPACE

By

Wilfredo Enrique Lugo-Beauchamp

December 2016

Chair: Jaime Seguel
Major Department: COMPUTING AND INFORMATION SCIENCE AND ENGINEERING

DNA local sequence alignments provide biological insights that can help scientists identify genetic diseases, map newly obtained sequences to known genomes, or identify common genomic patterns on same species. Even when optimal sequence alignment algorithms have been well understood since more than 3 decades ago, the technological advancements of Next Generation Sequencing and the genomic data explosion they produced made them impractical today. Moreover, there is an increasingly necessity of fast comparison of very small sequences (less than 5,000 base pairs) against full genomes (greater than 100M base pairs). This thesis focuses on the local alignment problem for sequences with extreme length disparity and presents an *Improved Search for a Local Alignment* (ISLA) algorithm which provides an iteration based algorithm that achieves near optimal results by focusing local alignment only on specific areas of interest. ISLA also provides a probabilistic model to understand the chances of achieving a higher score.

Resumen de Disertación Doctoral Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de DOCTOR EN FILOSOFIA

ALINEAMIENTO LOCAL PARA SECUENCIAS DE DNA CON LARGOS SUMAMENTE DISPARES REDUCIENDO EL ESPACIO DE BÚSQUEDA

Por

Wilfredo Enrique Lugo-Beauchamp

Diciembre 2016

Consejero: Jaime Seguel
Departamento: Ciencias e Ingeniería de la Información y la Computación

Los alineamientos locales de secuencias de ADN son usados para proveer información biológica relevante que puede ayudar en la detección de enfermedades genéticas, parear secuencias nuevas con genomas conocidos o identificar patrones comunes en una misma especie. Aún cuando los algorithmos de alineamientos de secuencias locales son conocidos desde hace mas de treinta años, los avances tecnológicos causados por los nuevos mecanismos de secuenciación y la explosión de información de genomas causado por estos, ha provocado que no se puedan utilizar hoy en día. Mas aún, hay una necesidad apremiante de poder ejecutar comparaciones rápidas de secuencias cortas (menores de 5000 nucleótidos) contra genomas completos (mayores de 100M nucleótidos). Esta tesis se enfoca en el problema de alineamiento local de secuencias de largos sumamente dispares y presenta un nuevo método que reduce el espacio de búsqueda para lograr alineamientos locales cercanos o igual al alineamiento óptimo. El algoritmo también provee un modelo probabilístico el cual da una medida de cuán buena es la puntuación del alineamiento obtenido.

To the core of my meaning in life: Lisie, Isaac and Kiara...

Acknowledgments

During this journey there were a few people that in one way or another were key contributors on this expedition. First, I want to thank my advisor Dr. Jaime Seguel which in a leap of faith decided to take a part-time PhD student under his umbrella. I will be always grateful. As a PhD student with a full-time industry job during the day, this would not have been possible without the help of my company Hewlett Packard Enterprise and, specifically, my managers during this period: Freddy Hilerio, Patricia Heim, William Navas and Robert McCarthy. They never hesitated to support me while taking courses during the day or providing flexible working hours while studying for the qualifiers. They also allowed me to travel to present my work on conferences and provided me company resources and time to work on my dissertation. My deepest gratitude to them. I don't think I can ask for a better place to work. I also want to thank Dr. Juan Valera who was my qualifier study partner and without him everything would have been a lot more difficult. Last but not least, I want to thank my partner in crime, Wilo. I am convinced that combining us together we make a really good student, without you all those core courses would have been almost impossible to complete, while working full time.

Thanks...

Table of Contents

List of Tables

List of Figures

ABBREVIATIONS LIST

ACMG	American College of Medical Genetics and Genomics
BLAST	Basic Local Alignment Search Tool
BLAT	BLAST-Like Alignment Tool
BP	Base Pairs
CCD	Charge Coupled Device
cDNA	Complementary DNA
DNA	Deoxyribonucleic acid
DCT	Discrete Fourier Transform
EVD	Extreme Value Distribution
FAMA	Fast and Accurate Mapping Assembly
FFT	Fast Fourier Transform
FFTW	C subroutine library for computing the discrete Fourier transform
HHM	Hidden Markov Model
HSP	High Scoring
IFFT	Inverse Fast Fourier Transform
ISLA	Iterative and Scalable Local Alignment
LINE	Long Interspersed Nuclear Elements
LTR	Long Terminal Repeats
MAQ	Mapping and Assembly with Quality
NGS	Next Generation Sequencing
PCR	Polymerase Chain Reaction
RNA	Ribonucleic acid
SINE	Short Interspersed Nuclear Elements
SMP	Symmetric multiprocessing
SNP	Single Nucleotide Polymorphism
SW	Smith-Waterman
TE	Transposable Element

1. Introduction

1.1 Biological Sequence Alignments

Over the last two decades, the advancements on sequencing technologies triggered an avalanche of biological data without precedent in the computational domain. Even when most of the best known biology related algorithms were discovered in the last century, none of them anticipated the amount of data we are generating in a single day. Most of these algorithms are simply computationally intractable using available data. Thus a new set of heuristics and probabilistic methods that sacrifice accuracy for speed came into play.

Biological systems store structural and functional information in biochemical strings. Biologists use DNA as the *master plan* of the organism which contains the coded genetic sequence[1]. Knowledge of these sequences are nowadays indispensable for biological research and their applications in medicine and biotechnology. Once a whole DNA sequence (or genome) is obtained for that organism, they can start comparing that genome to establish relationships within the same species and to understand how the physical characteristics (phenotypes) can be correlated with differences in genomes[2]. Moreover, they also want to compare inter-species genomes to understand evolutionary relationships. The decision on how these sequences are related (or not) is based on what is called a pairwise alignment.

Pairwise alignment (or sequence alignments) is no different than comparing two strings of letters for differences between them[3]. However, from the biological perspective is not enough to know that two strings are different or equal, some metric or score is needed to understand how similar, or how different those two sequences are between each other. This scoring mechanism can help them embrace or discard results and draw conclusions based on those scores.

DNAs are not the only sequences of interests on the computational biology domain. The central dogma of biology describes how the DNA transforms into messenger ribonucleic acid (mRNA)[4] and how mRNA is traduced into amino acids which are the building blocks of proteins[5]. All these sequences are of interest for biologists, and when the term pairwise alignment or sequence alignment is used, it refers indiscriminately to DNA, mRNA or amino acid sequences.

Below sections will highlight some key applications of sequence alignments in computational biology.

1.2 Applications

Disease Diagnosis and Treatment. It is estimated that each cell in the human body experiments at least ten thousand DNA lesions per day[6]. These mutations can be triggered by internal or external factors. External causes can range from radiation (including sunlight), chemical and food exposures. Internal factors can be attributed to physiological means like DNA mismatches during replication, DNA strand breaks, hydrolitic reactions and reactive oxygen compounds such as ozone. Almost all of these mutations are repaired and corrected by cells internal mechanisms without causing major disruptions. However, there are cases when these reactive mechanisms are not working as expected and lesions can replicate and generate genome instability. Genome instability is known to produce at least 40 known diseases including most forms of cancer[7].

The disease diagnostics and treatment application domain deals with the detection of these DNA mutations (which can also happen in newborns) and the capability of providing individualized treatments for these diseases. In order to perform effective and rapid detection, the person's DNA must be sequenced and compared against known disease mutations. There is no way this can be done without some kind of pairwise alignment technique.

Evolution History. Regardless of today's diversity across humans, DNA has been providing new information on human history and human migration. By analyzing genetic DNA markers between living people and human fossils, scientists are now able to establish relationships on humans origins. Recent studies show that the entire world population descended from African migrants around 50,000 years ago[8]. This conclusion is reached by analyzing DNA samples from human (or human-like) fossils across different geographical locations and building a common ancestry or phylogenetic tree. To measure this genetic variation across fossils and/or living humans, some kind of sequence alignment is needed.

Conservation. Conservation genomics is a new area of study that is applying DNA analysis for species preservation purposes. The primary goal is to reduce the current rate of extinction and preserve (and promote) biodiversity of species[9]. DNA interspecies comparisons are performed to identify common origins and to trigger cross breeding techniques that can add genomic diversity to endangered species. Environmental changes and their effects on DNA are also studied as a way to identify how certain species can be helped to adapt faster to certain external factors. The area of conservation genomics relies heavily on full genome comparisons that are only accomplished by the use of next generation sequencing along with some kind of sequence alignment algorithm.

Genetic Fingerprinting. Advancements in sequencing technologies are also impacting our justice system[10]. DNA profiling applies sequence alignment and comparative techniques to the investigative criminal discipline. Its focus is not only on the aim of convictions on the guilty, but on exoneration of the innocent who are imprison wrongfully. Current criminal databases contains millions of reference genomes of known suspects and offenders on which DNA obtained from recent crime scenes are compared against. These databases also contain cDNA information on repeat or serial sex offenders.

Genetic fingerprinting is also used to determine relationships between two individuals and understand how biologically related they are to each other. The most common usage of this technology is what we know as paternity (or maternity) tests. In this type of testing, DNA samples are obtained from the individuals of interests (normally the child and the individual under study) and compared against each other. This test produces a similarity score (from 0% to 100%) where a high score increments the probability of a relationship between the two subjects. Thousands of these tests are conducted each year around the world and sequencing alignments are the backbone of them.

Agricultural. DNA analysis in agriculture has been predominant in recent years. Plants, seeds and animal genomes are being modified genetically to increase resistance to diseases and/or plagues, increase yields, boost nutritional value, increment shelf life, make them weather resistant, or even being injected as vaccines into crops [11]. Specific gene identification and correlation to specific factors is the product of hundreds and hundreds of DNA comparisons using some kind of sequence alignment algorithm.

2. Related Work

This chapter will discuss in detail the related work in the area of Sequence Alignments. It will start with the mathematical definition of a sequence alignment and its evolution and then will discuss the new technologies on DNA sequencing and current computational challenges they impose. The last section will introduce the concept of unbalanced sequence alignments and will present two biologically relevant challenges in the discipline.

2.1 Sequence Alignments

A two sequence alignment problem is defined based on two sequences A and B where:

$$A = a_1a_2a_3 \ldots a_{N-1}a_N$$

$$B = b_1b_2b_3 \ldots a_{M-1}a_M$$

$$N = |A|$$

$$M = |B|$$

$$a_i, b_j \in \Sigma_{ALL} 1 \leq i \leq N, 1 \leq j \leq M$$

$$\Sigma_{ALL} = \Sigma_{DNA}|\Sigma_{RNA}|\Sigma_{AMN}$$

$$\Sigma_{DNA} = \{G, A, T, C\}$$

$$\Sigma_{RNA} = \{G, A, U, C\}$$

$$\Sigma_{AMN} = \{C, S, T, P, A, G, N, D, E, Q, H, R, K, M, I, L, V, F, Y, W\}$$

As noticed, a single sequence is based on one of three alphabets: Σ_{DNA}, Σ_{RNA} and Σ_{AMN}. Each of these alphabets represents the symbols of a DNA sequence, RNA sequence and a protein sequence, respectively. Since there are repeated symbols across the different alphabets, the context of where the sequence is coming from provides the information needed to understand what type of sequence it is. However, an alignment is only possible between two sequences of the same alphabet (e.g. two DNA sequences or two RNA sequences or two protein sequences).

Now that the sequences A and B definitions are presented, an alignment of the sequences A and B is obtained if three specific conditions are satisfied:

- Condition 1

$$|A'| = |B'| = |N'|$$

$$|N'| \leq N + M$$

$$A' = a'_1 a'_2 a'_3 \ldots a'_{N'-1} a'_{N'}$$

$$B' = b'_1 b'_2 b'_3 \ldots b'_{N'-1} b'_{N'}$$

$$a_k, b_k \in \Sigma'_{ALL} 1 \leq k \leq N'$$

$$\Sigma'_{ALL} = \Sigma_{ALL} \bigcup \{'-'\}$$

- Condition 2

$$a'_k = \text{‘}-\text{’} \Rightarrow b'_k \neq \text{‘}-\text{’}, 1 \leq k \leq N'$$

- Condition 3

$$A = f(A')$$

$$f : \Sigma'^{*}_{ALL} \rightarrow \Sigma^{*}_{ALL}$$

$$x = yz, yz \in \Sigma'^{*}_{ALL}$$

$$f(yz) = \begin{cases} \lambda, yz = \lambda \\ \lambda f(z), y = \text{‘}-\text{’} \\ y f(z), y \neq \text{‘}-\text{’} \end{cases}$$

The first condition simply states that after a valid alignment, the new aligned sequences A' and B' must have the same length. The second condition makes sure that none of the aligned sequences can contain an interruption marker (‘-’) at the same position. Last condition guarantees that the order of the original sequences is preserved. This is done by defining a function that removes interruption markers from the aligned sequence. An aligned sequence without interruption markers must be exactly the same as the original sequence. Figure 2–1 shows different example alignments between two protein sequences.

```
(a)  -  E  A  E  H  W  A  -  -  P
     H  E  A  G  A  W  G  H  E  E


(b)  -  E  A  -  E  H  W  A  -  -  P
     H  E  -  G  A  -  W  G  H  E  E


(c)  -  E  A  -  -  E  H  W  A  -  -  -  -  P
     H  E  A  G  A  -  -  W  G  H  E  E  -  -
```

Figure 2–1: **Alignment examples between protein sequences A=EAEHWAP and B=HEAGAWHEE. (a) Alignment with N'=10, (b) Alignment with N'=11, (c) Alignment with N'=14.**

2.1.1 Alignment Quality Criteria

The previous section explained the definition of a two sequence alignment and it was shown how there will be several possible alignments for the same two sequences. The main problem arises when selecting the best alignment from all possible alignments. This is done by assigning grades or global scores to each possible alignment and then selecting the one with the highest score. All scoring strategies are based on probability models created by studying symbol occurrences on real sequenced data. Based on these studied sequences, the probability that certain nucleotide or amino acid a_i could transform or mutate into b_j across a certain period of evolutionary time could be obtained. Based on this single symbol probability ($p_{a_i b_j}$), it could be establish the probability for the whole alignment as:

$$P(A, B|M) = \prod p_{a_i b_j}$$

The above probability considers only the likelihood that both sequences are related based on a match model M. However, to construct a more robust approach, there is a need to understand the probability when both sequences are totally unrelated based on the studied sampled data. This is obtained by using a simple probabilistic random model R, where it is assumed at symbol a_i occurs independently with a frequency q_a. Based on this, the probability of random occurrence of a mutation between nucleotide or amino acid for an alignment could be calculated as the product of all individual probabilities of each symbol:

$$P(A,B|R) = \prod_{i}^{N} q_{a_i} \prod_{j}^{M} q_{b_j}$$

Based on these two probability models, a ratio for overall alignment score could be obtained as follows:

$$\frac{P(A,B|M)}{P(A,B|R)} = \frac{\prod p_{a_i b_j}}{\prod\limits_{i}^{N} q_{a_i} \prod\limits_{j}^{M} q_{b_j}} = \prod_{i,j}^{N,M} \left(\frac{p_{a_i b_j}}{q_{a_i} q_{b_j}} \right)$$

Above ratio is commonly known as the *odds ratio*. Since it is more difficult to implement an alignment based scoring system which uses products of probabilities, an additive score system is produced by applying logarithms to above ratio:

$$\begin{gathered} S = \sum_{i,j}^{N,M} s(a_i, b_j) \\ where: \\ s(x,y) = log\left(\frac{P_{x,y}}{q_x q_y}\right) \end{gathered} \tag{2.1}$$

Above equation 2.1 is known as the *log-odds ratio* and it is the primary metric for sequence alignment scores [3]. The *log-odds ratio* is based on an evolutionary time assumption. Since each a_i element may mutate more than one time depending on the time elapsed, different calculations are needed to be performed depending on the estimated evolutionary time between the sequences under comparison. These values are calculated a priori and stored in lookup tables or substitution matrices. These matrices are commonly available and are developed by biologists based on specific needs. The most popular substitution matrices are the Point Accepted Mutation (PAM)[12] and the BLOck SUbstitution Matrix (BLOSUM)[13]. Both of them are used exclusively for amino acids/proteins and there exists different matrices depending on the estimated evolutionary time between sequences. PAM matrices are preferred when working with closely related proteins but don't work very well for dissimilar sequences. On the other hand, BLOSUM matrices uses blocks of preserved protein segments across multiple alignments, and uses a threshold-clustering approach to reduce the bias caused by closely related proteins. In this way, important preserved segments are identified and have more scoring weight than non preserved segments. Substitution matrices are commonly input arguments to the sequence alignment problem and changes on the matrix used will change the final score of the alignment.

The alignment scoring system explained above does not take into account interruptions or gaps on the alignment (represented by the interruption marker symbol '-' on section 2.1). These gaps represent the addition or removal of symbols in a sequence. These are also called insertion and deletions and they account for mutations or changes occurred via *natural selection*. Each one of the gaps in an alignment has a negative impact on the final score. This penalty is normally referred as a *penalty gap*. These penalties could be based on a linear score or an affine score. On the linear score each gap has the same penalty regardless how many consecutive gaps were detected. In the affine gap model, there is a gap-extension factor which mitigates gaps on long insertions or deletions across the alignment. The gap function to be used depends on the known characteristics of the sequences and it is also an input to the sequence alignment. As in the substitution matrix, a change in the gap function will cause a change on the final alignment score.

2.1.2 Sequence Alignment Computational Complexity

The brute force two sequence alignment algorithm is an $\mathrm{O}\left(N'!\right)$ problem since it requires the alignment of all possible combinations of alignments of size N'. The total number of combinations could be obtained from:

$$\binom{N'}{N'_{min}} = \left(\frac{N'!}{(N'-N'_{min})!N'_{min}!}\right)$$
$$N'_{min} = min\{N, M\}$$
$$N' = M + N$$

Things changed dramatically in 1970 when Needleman and Wunch, using dynamic programming techniques, designed an algorithm to compute global sequence alignments for two sequences with an $\mathrm{O}(N^2)$ complexity (Assuming $M = N$)[14]. Until today, this algorithm (or one of its many variations) is still being used for optimal sequence alignments. Needleman-Wunsch recursion is defined on 2.2, where F is the similarity matrix obtained from the recursion, $f(g)$ is the gap penalty function used on insertions or deletions which is a function of the gap length g. Last, $s(a_ib_j)$ is the substitution matrix function that returns the similarity score between a_i and b_j. Also, $F_{0,j} = g * j$ and $F_{i,0} = g * i$ are the basis for the recursion.

Even when the maximum similarity score is obtained on sequences A and B (provided by F), the similarity pathway P is needed to identify the gaps (if any) across both sequences. This is normally accomplished by having separated two dimensional array tracking the decisions made for each F_{ij} element and performing a backtracking when F calculation is completed. In this way, homologies between both sequences can be identified and a global alignment score between the two sequences is produced. This global alignment total score is stored on element F_{NM}. It also needs to be highlighted that the space complexity for Needleman-Wunsch is also $\mathrm{O}(N^2)$ since the similarity matrix F needs to be stored.

$$F_{ij} = max \begin{cases} F_{i-1,j-1} + s(a_ib_j) \\ F_{i-1,j} - f(g) \\ F_{i,j-1} - f(g) \end{cases} \tag{2.2}$$

2.1.3 Local Alignments

Even when Needleman-Wunsch could detect homology sections on pair sequence alignments and even when some of these could be inferred by analyzing high score sections on the similarity matrix F, this analysis tends to be impractical. The limitation with Needleman-Wunsch is that a continuous segment of gaps (insertions or deletions) will affect subsequent locals and global homologies, lowering the global score and masking homology segments on the alignment.

To address the local alignment problem, in 1981 Smith-Waterman proposed a slight modification to the Needleman-Wunsch recursion[15]. They added a threshold to the F_{ij} computation where the function will ignore all scores obtained that are below that threshold. Equation 2.3 shows the changes to the F_{ij} calculation performed by Smith-Waterman. $F_{0,j} = 0$ and $F_{i,0} = 0$ are the basis for the recursion. As in Needleman-Wunsch, space complexity of Smith-Waterman is $\mathrm{O}\left(N^2\right)$.

$$F_{ij} = max \begin{cases} F_{i-1,j-1} + s(a_i b_j) \\ F_{i-1,j} - f(g) \\ F_{i,j-1} - f(g) \\ 0 \end{cases} \tag{2.3}$$

Above section presented a detailed description and definition of sequences alignments and its global and local variants. Next section will focus on understanding the technology behind Next Generation Sequencers and the computational challenges they imposed.

2.2 Massively Parallel Sequencers

Sequencing is the process of determining the precise order of the characters that compose a cDNA sequence. cDNA is synthetically produced by the sequencing technology by using the reverse transcriptase enzyme. During the last ten years, advancements in sequencing technologies had made possible the increased throughput of new sequences to numbers unthinkable 20 years ago. Current technology is able to produce up to 30Gbp per day of new data which represents an additional challenge for biologists. This new trend of sequencing technology is also known as Next Generation Sequencers (NGS). Table 2–1 shows 3 examples of leading commercial sequencers and their respective capabilities[16].

Technology	**Maximum Read Length (bp)**	**Accuracy (%)**	**Reads**	**Advantages**
Roche 454	700	99.9	1M	Longer Read Lengths
Illumina	50 Single Ended, 50 Pair Ended	98	3G	High Coverage
SOLiD	50+35bp	99.94	1.2M-1.4M	Accuracy

Table 2–1: **Massively Parallel Sequencers Overview**

All NGS share a common sample preparation method, regardless of the technology used. These pre-sequencing steps that are common to all sequencers, produce hundreds of millions of cDNA fragments. As shown in Figure 2–2, these steps include random fragmentation of the DNA under study, ligation of the fragments with custom linkers or adapters (each technology has their own proprietary adapters) and amplification. The amplification step is used in all the sequencers to facilitate the measurement of the detection of the fluorescence signal emitted during the sequencing processes.

All of these sequencers rely on SDS-PAGE[17] or fragment separation and the separate fluorescent tag which identifies each nucleotide. There are 4 separate reactions, one for each nucleotide. Based on these reactions, each nucleotide (A, T, G and C) is targeted one at a time. Some characteristics in the excitation wavelength of these reactions allow the identification of each nucleotide and with it, the sequencing of the cDNA. The next section is a brief description of current sequencing technologies.

2.2.1 Illumina GA/HiSeq System

In the Illumina System, the amplification process produces thousands of identical copies of the same DNA fragment and locates them into a single cluster. Each cluster is made up of thousands of identical fragments and each cluster contains a different fragment of the original sequence of interest. Based on the fluorescent signals produced by the chemical reactions, the sequencer detects each type of nucleotide fragment at a time, until all DNA fragments in the cluster are identified. Hundreds of millions of clusters can be sequenced in a massively parallel sequencer. A measure of interest for some practitioners is the coverage of the sequencer, which is defined as the ratio between the size and number of reads and the size of the genome. The Illumina sequencer is claimed to provide one of the highest coverage ratios[16]. Figure 2–2 illustrates coverage differences between regions of a genome.

2.2.2 SOLiD System

The Sequencing by Oligonucleotide Ligation and Detection (SOLiD) sequencing framework, which was developed by Life Technologies, is based on sequencing by ligation. This technology relies on Emulsion PCR[18] for amplification. With this amplification mechanism, a single bead produces thousands of copies of the original fragment. The method uses also a double base encoding in which each base position is queried two times with different fluorescence tags. This fluorescence tag should match perfectly the fragment under study, ensuring the nucleotide is accurately matched with the wavelength of the signal emitted. The SOLiD system is the preferred choice of users that require higher accuracy.

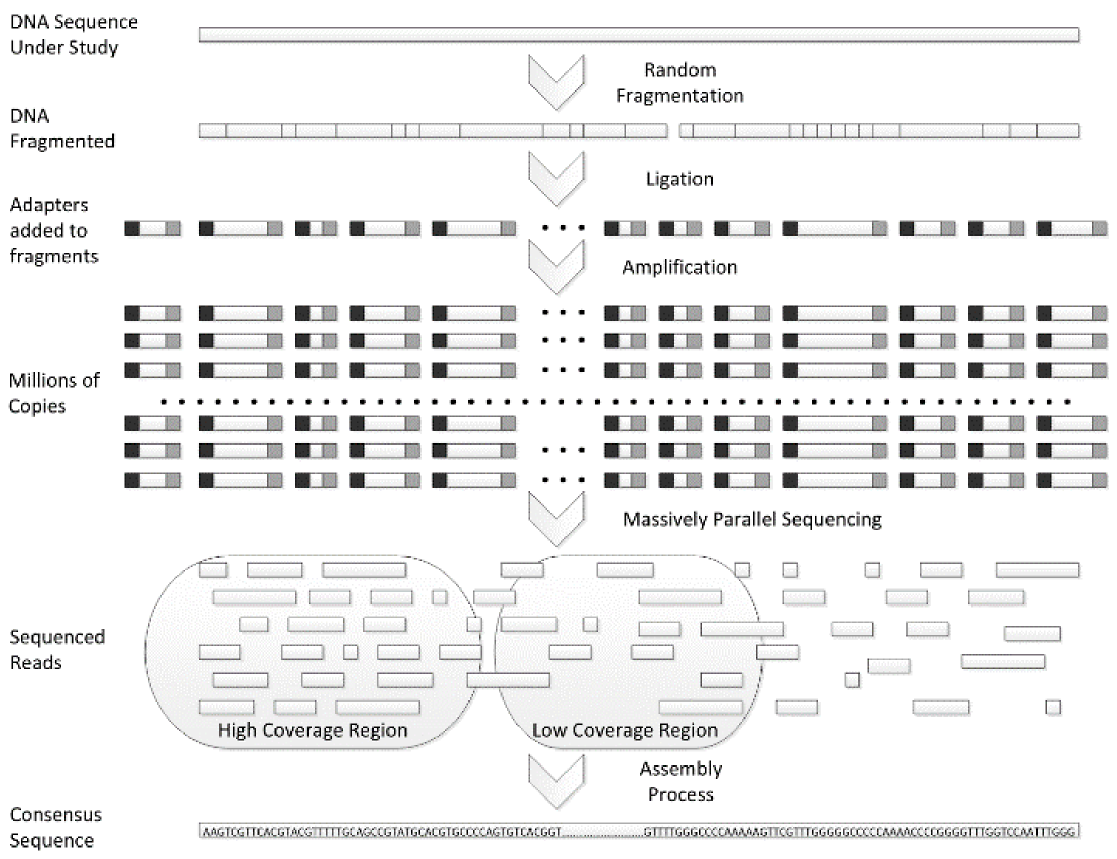


Figure 2–2: **DNA Sequencing and Assembly Process Overview for the NGS**

2.2.3 Roche 454

The Roche 454 relies on pyrosequencing technology, which measures the emission of light when complementary nucleotides are incorporated into a growing fragment strand. This system also uses Emulsion PCR[18] for the amplification process. Each single bead is then placed into a sampling well on which the sequencing reaction will occur. A high sensitivity CCD camera registers the light emission of the millions of reactions, which are then used to properly classify each base. Among all current massively parallel sequencers, the Roche 454 system produces the sequence fragments of longest length, facilitating, the computational assembly of the genome.

2.2.4 Computational Impact

The NGS systems explain on section 2.2 return millions of small DNA segments (or reads) of the sampled sequence. In order to be useful, these reads need to be assembled into a single sequence. For already sequenced organisms this process is known as *Mapping Assembly*. One way to do this is by comparison with a similar, known genome. Such approach involves the application of millions of two-sequence alignments, which turns sequence alignment procedures into a true computational bottleneck of NGS systems. This amount of data could not be analyzed in a timely fashion with a precise alignment algorithm.

Moreover, NGS systems are now producing hundreds of daily sequenced data and most of it ends being stored in databases for later processing[19, 20]. Normally a bioinformatics expert first step after generating a new sequence is to compare it against what is already available and studied to better characterize the newly sequenced DNA. Since DNA genome sequences can range from a 20M base pairs (bp) to 129,907M base pairs, sequence alignments between the new sequence (query) and the sequenced data bases (references) is not a tractable problem since it becomes a $\mathrm{O}\left(RMN\right)$ where R is the quantity of references to be looked at, M is an average query length and N represents the average genome reference size.

Based on these constraints, bioinformaticians use heuristically based algorithms that sacrifice accuracy for speed. These methods rely on inputs and knobs that most of the times are difficult to understand and experiment with. In the assembly process, this results in cases where the same genome data will probably generate different assembly results based on each biologist's custom parameters. Moreover, it can not be determined which result is closest to the optimal assembly.

The next chapter will focus on understanding a subset of the sequence alignment challenges on characterizing small sequences against a substantially longer sequences. In those cases, the queried sequence is of short length when compared to the reference sequence.

2.3 Unbalanced Sequences

Through the rest of this thesis work, the genome concept will be used broadly to explain the challenges on comparing small sequences to longer sequences. Most of the time, these longer sequences are species genomes. Next subsection aims at providing an overview of a genome and how it is represented. After that, next subsections will explain in detail the Transposable Element (TE) and the Genome Assembly problems, which are selected challenges that need a better way to handle unbalanced sequences comparisons.

Genome

A genome is a DNA sequence (or RNA on some viruses) which represents all chromosomes for the species under study. Even when most people think of a genome as a very long single sequence of DNA, the truth is that it is not. A genome can be seen as a set of chromosomes and each chromosome is kind of independent from each other. In the computational domain, a full genome is not always achieved. In most cases a genome can be completed and a consensus sequence is reached, but not on 100% of it. There may be missing, or disconnected fragments.

A genome is normally contained in a single file which has the consensus chromosomes along with the missing and/or cDNA fragment sequences. Below list contains the basic components of a genome file (not all genomes contain them all):

1. Chromosome - consensus cDNA sequence attributed to a specific chromosome.
2. Non-Chromosome - Long DNA sequences that hasn't been assigned to a chromosome (yet). These maybe mitochondrial or viral fragments.
3. Scaffold - Long sequences that are known to be in the correct order but that hasn't being connected to a proper chromosome or another continuous area of a known sequence.

2.3.1 Transposable Elements

Transposable Elements (TE) (or mobile elements) are DNA sequences that have the ability to move themselves to other parts of the genome within the same cell[21]. They are known to exist in all known genomes and can be divided into 2 main categories: *DNA Transposons* and *Retrotransposons*. Even when TE movements increase genome diversity they have also being implicated in diseased states.

The DNA Transposons are common in prokaryotic cells, but can also be found in insects, worms and humans. They mostly move in a *cut and paste* approach but they remain in the vicinity of their original position. These sequences are normally called Insertion Sequence (IS) and commonly are between 700bp and 1500bp long. Figure 2–3 shows the mechanism on how a donor DNA is detached from the transposon by the help of the transposase enzyme and how it reattaches itself to the target DNA. Between 2%-3% of the human genome consist of DNA transposons.

The retrotransposons accounts 42% of the human genome and have the capability to move around in a *copy and paste* approach. As shown in Figure 2–4 retrotransposons are transcribed and then translated into the reverse transcriptase enzyme which generates complementary DNA (cDNA) based on the transposon RNA template. This cDNA is then inserted back into the genome in a different position than its original position.

Retrotransposons are divided further into 2 categories: Long Terminal Repeats (LTR) and Non-LTR. LTR Transposons are abundant in eukaryotes organisms, but at least in the human genomes most of them are inactive (can't move around anymore)[22]. Non-LTR transposons on the other hand are divided into two main categories: Short Interspersed Nuclear Elements (SINE) and the Long Interspersed Nuclear Elements (LINE). LINE elements are divided into L1, L2 and L3 but only L1 elements are still active in the human genome. L1 accounts for around 17% of

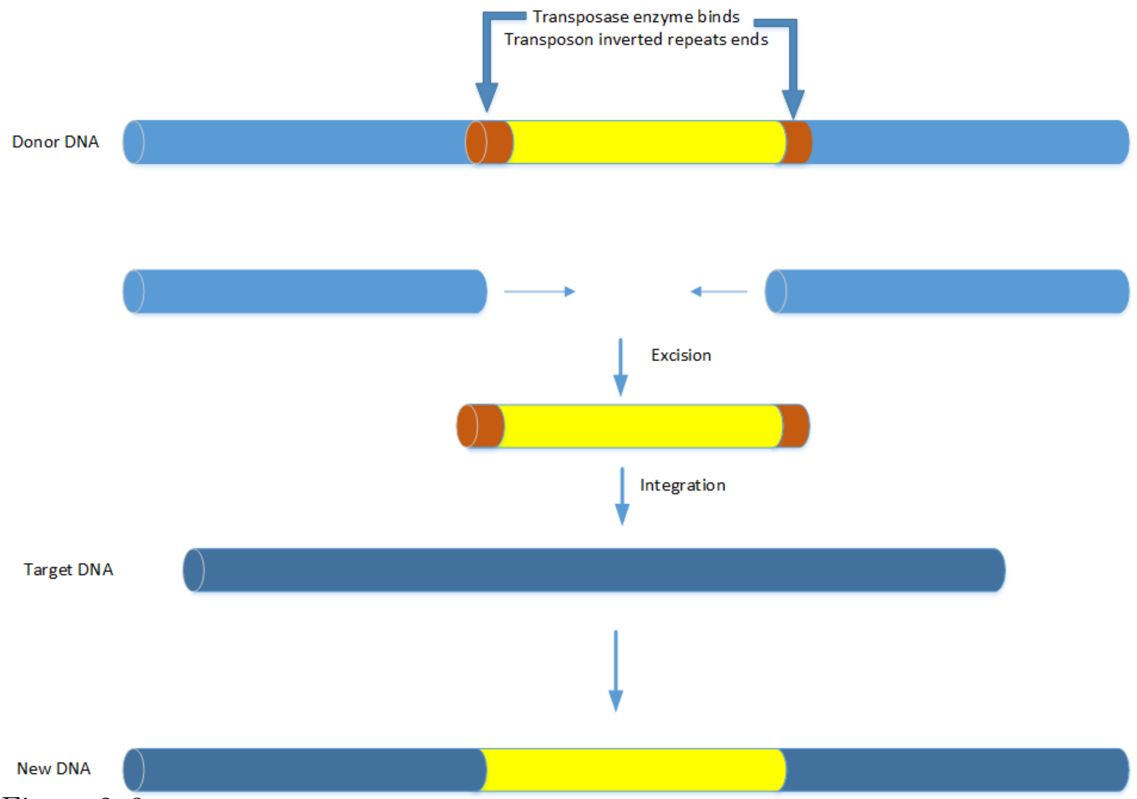


Figure 2–3: **DNA Transposons work by making a *cut and paste* approach and moving with the help of certain proteins to another part of the genome**

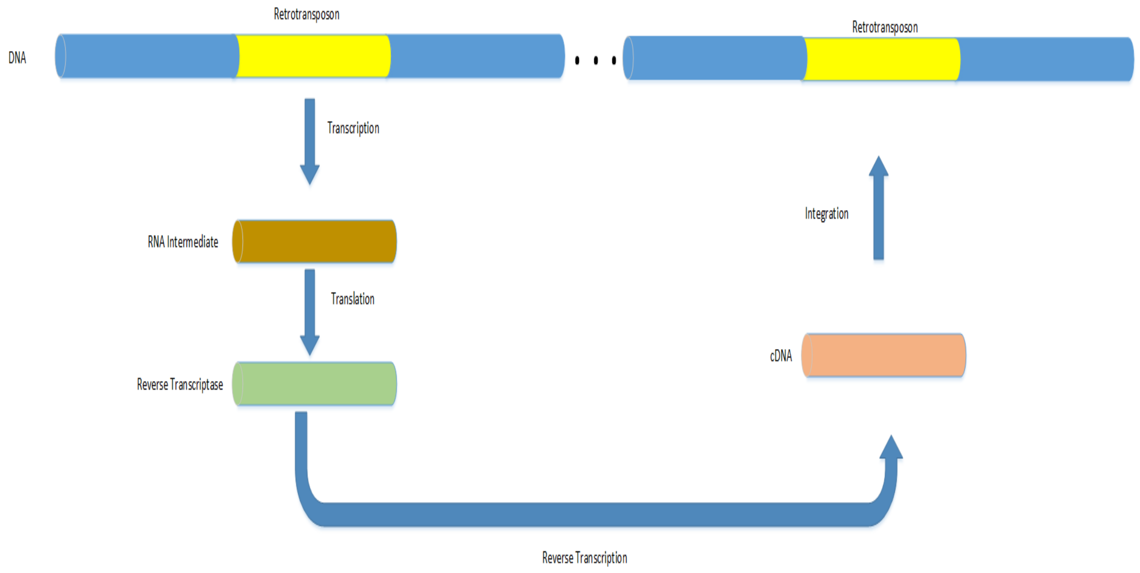


Figure 2–4: **Retrotransposons copied themselves by generating the reverse transcriptase enzyme after translation. These enzymes help to produce cDNA which is then inserted by the transposon on another position on the genome**

the human genome and even when the vast majority of them are inactive, around 80-100 of these sequences have been estimated to be moving around the genome[23] with an estimated average length of 7,000bp. L1 events are predicted to trigger 1 in 1000 diseases producing insertions in humans.

Transpose Elements Detection Complexity

For optimal detection of TE elements in a genome, the sequence identified as a TE (e.g. L1 sequence) needs to be aligned across the whole genome. This process computational complexity is $\mathrm{O}(NM)$ where N is the size of the genome and M the size of the L1 sequence (around 7,000bp). In the human domain, since the genome is about 3.2Gbp, therefore optimal sequence alignments becomes impractical due the highly unbalanced lengths between the sequences involved. Based on these issues, heuristic approaches are used today to detect TE (specifically L1) locations in the genome. This research will focus on TE detection based on a library-based search. That means that TEs were already identified by another tool and the goal is to search for those TEs in a genome. The section below will give a detailed overview of the tools available for TE genome searches.

Current Tools

The following paragraphs provide a detailed overview of the tools currently used to search for TEs in genomes. First BLAST tool will be discussed in detailed and then the Cross Match and N-HMMER tools will be briefly discussed.

BLAST. The Basic Local Alignment Search Tool (BLAST) is by far the most used tool on the sequence alignment and sequence comparison domains[24]. BLAST is based on the concept that a satisfactory alignment between two sequences consist of a set of short length exact matches. BLAST divides the query sequences into a list of short words (or K-mers) and query these K-mers into a previously indexed reference database. Each K-mer consist of a word of length K which is a subsequence of the query sequence. As shown on Table 2–2 the length of the set of K-mers is determined by $L - K + 1$ where L is the sequence length.

K=4	**K=6**	**K=8**
ACTC	ACTCGA	ACTCGATG
CTCG	CTCGAT	CTCGATGC
TCGA	TCGATG	TCGATGCT
CGAT	CGATGC	CGATGCTC
GATG	GATGCT	GATGCTCA
ATGC	ATGCTC	ATGCTCAA
TGCT	TGCTCA	TGCTCAAT
GCTC	GCTCAA	GCTCAATG
CTCA	CTCAAT	
TCAA	TCAATG	
CAAT		
AATG		

Table 2–2: **Example of different K-mers sets for query sequence ACTCGATGCT-CAATG**

Since reference databases are indexed also by K-mers of the reference sequences, a linear search is quickly performed to match queried K-mers to references. However, the K value selected have implications on the results of the algorithm. A lower K will increase the quantity of searches, the amount of overlapping and will make more difficult DNA repeats identification. On the other hand, larger K values will increase the amount of searches and computational resources, but will help in repeat detection.

Once all K-mers have been selected, a scoring matrix is used to produce a measure for all pairs. No gaps (insertions or deletions) are assumed on this step and scores are calculated based only on matching letters. All the scores for the pairs are sorted and only the ones higher than a *threshold* T are selected for further analysis. This list of chosen scores is called High Scoring Pairs (HSP) and this process is often call *seeding*. Next step BLAST goes into the K indexed database to get the reference matches for the HSP.

At this point BLAST starts expanding the obtained seeds on both sequences: query K-mers and reference subsequences of length K on both directions and a new *threshold* T is calculated. If the score increases with the extension, then it continues extending until one of the following conditions is reached: (1) Scores drops X from its maximum achieved, (2)Cumulative score reached zero or less or (3) end of one of the sequences is reached.

Above process is a description for the core BLAST algorithm. As it can be seen, it is very fast since it is not doing a full sequence alignment. It does a linear search between the query and the reference and only use scoring matrices as a way to measure sequence comparisons. There is no guarantee that the hits obtained by BLAST are the best ones since it does not take into account gaps and there could be better alignments that were not selected in the HSPs. However, BLAST also provides a statistical model that allow its users to perceive the statistical significance of the returned hits. This will be explained in detail the below.

BLAST Statistical Significance. It has been proved that a Smith-Waterman local alignment without gaps follows an Extreme Value Distribution (EVD). An EVD is often used to model the smallest or largest value among a large set of independent, identically distributed arbitrary values representing measurements or observations, in this case ungapped Smith-Waterman scores[25]. Based on the generalized probabilistic theory for EVDs, a Type-I (or Gumbel) distribution can be used to obtain the probability P of a score S equal or greater than x for two sequences of size N and M. This can be expressed as:

$$\begin{gathered} P(S \geq x) = 1 - e^{-e^{-\lambda(x-v)}} \\ where: v = \frac{KN'M'}{\lambda} \end{gathered} \tag{2.4}$$

K and λ are estimated statistical parameters that are dependent on the substitution matrix used, sequence composition and gap penalties. As it can be seen, the probability assumes an exponential decay. Since BLAST's HSP are based in subsequences and not overall sequence, N' and M' are sequences with effective lengths that take into account alignment positions. They are estimated by:

$$M' \approx M - \frac{ln(KMN)}{H}$$
$$(2.5)$$
$$N' \approx N - \frac{ln(KMN)}{H}$$

H is the average score per alignment for two random sequences. Since BLAST already knows its databases (they are already indexed), on most implementations λ, H and K values are pre-calculated by the tool.

Using Equation 2.4 BLAST is able to calculate the expected score E of a match which is the number of times an unrelated random sequence can get an score S which can be higher then x against the set of sequences stored on a database D. This expected score is obtained from:

$$E \approx 1 - e^{P(S>x)D} \qquad (2.6)$$

The expected score or *E-value* is used by biologists to determine a significance of a selected HSP and it is reported by BLAST on all them which are above T. Even when BLAST is an heuristics method that does not guarantee the best alignment, its statistical significance on results is one of the best features that has been added to the sequence comparisons field and explains why BLAST is embraced by the scientific community.

Cross Match. Cross Match is an efficient implementation of Smith-Waterman algorithm that claim to have reduce the number of machine instructions per similarity matrix cell[26]. It relies on SWAT tool which scans a database and provides statistical significance on K-mer hits[27]. Based on those hits, then a constrained Smith-Waterman is executed on highlighted sites.

N-HMMER. N-HHMER uses known HMMER framework to search for TEs in a sequence[28]. HMMER is based on the Hidden Markov Model (HMM) that applies probabilistic inference methods to the sequence alignment problem. Even when it is based on probabilistic model, HMM provides gap based scoring based on the query profile and calculates the signal of homology based on the most dominant HMM algorithm.

2.3.2 Genome Assembly Process

The genome assembly process aims at generating a consensus sequence from the reads produced by a NGS system as shown on Figure 2–2. Reconstruction algorithms are grouped in two broad classes, namely De Novo assembly, and mapping to a known reference genome. The De Novo assembly process is used when there are no known or consensus genome available for the organisms under study[29]. However, whenever the sampled specimen is known to be a variant or closely related to a species whose genome has been previously sequenced, the assembly process is done by alignment (or mapping) with the known genome, also called reference genome. Indeed, a reference genome sequence is a DNA template assembled by scientists as a representative example of the specie genetic information. Thus, in these cases, the assembly process orders and maps the reads to the reference genome, to produce a consensus sequence. Since the count and length of chromosomes varies across species, the length of the reference genome also will vary by species. In the case of the human genome, the reference genome is about 3 billion nucleotides (or base pairs).

Reference Genome Assembly Complexity

The naïve approach to a genome assembly is to simply map all possible reads against the reference genome. In this implementation, a sequence alignment is performed against the read under study and the reference genome, as is shown on Figure

2–2. For each read, the assembly process must figure out the best position for that read in the genome. As explained on section 2.2.4, the computational complexity for optimal local alignment is $O(RMN)$ where R is the number of reads, M the average read length and N is the size of the reference genome. Since each read may contain insertions, deletions or mutations at the same position in the genome an exact match is not always possible. As with the TE problem described on above section, the time and space complexity of an optimal local alignment is deemed too high to handle reads assembly, thus, heuristics assembly algorithms are used. Subsection 2.3.2 will present a detailed overview of the most common genome assemblers used today by the scientific community

Current Tools

BLAT. The BLAST-Like Alignment Tool (BLAT) algorithm uses the heuristics of popular BLAST algorithm to align sequenced reads to a reference genome and produce a consensus DNA genome string [30]. As BLAST[24] and FASTA[31] this algorithm is based on the heuristic principle in which each major alignment is a combination of small exact matches. To implement this heuristics, BLAST indexes the query sequence into a set of K-words or K-tuple. The set of K-tuples represent all possible subsequences of length K of the query sequence. A search is performed against different Databases and the best matches between the K-tuple and the Databases sequences are then analyzed in more detail. BLAT is similar in spirit, but with a few differences. In BLAT, the query sequence is not indexed, only the reference genome is indexed. Also, in BLAST an extension is performed when one or two overlapping or close K-tuples provide together a high scoring alignment (at least greater than a prefixed threshold T). In BLAT, extensions may be performed on any number of K-tuple (not just two).

SeqMap. SeqMap is another tool for mapping short reads into a reference genome to generate a consensus sequence[32]. SeqMap relies on splitting each read into parts assuming the number of possible substitutions between the read and the reference genome is known. It is based on the Pigeonhole principle: if X substitutions are present and the length of the read is M, then the read may be partied in pieces of length $\frac{M}{2X}$ to ensure that at least 2 pieces may have exact matches. These exact matches could then be easily looked up against the reference genome and provide an index to the query read segment.

MAQ. The Mapping and Assembly with Quality (MAQ) alignment method indexes the first 28 base pairs of each read, since these are typically the read segment with best quality[33]. It uses the same hash technique and pigeonhole principle of SeqMap. The main difference is that it fragments only the first 28 nucleotides into 6 hash indexes, which combine different 8bp seed templates in the form of 11110000, 00001111, 11000011, 00111100, 11001100, and 00110011 where nucleotides in a position represented by 1 will be indexed while those at positions represented by a 0 are not. For each template, each read is indexed based on the template nucleotide mapping. As seen by each template, not all nucleotides on the segment will be indexed. It depends on the current template. Once a hash table is created for each template on each read, a set of 28-tuples is produced from the reference genome and compared against the template hashes. If there is an exact match between the read templates index and the reference genome index that is considered to be a hit. Now, once a hit is found, the read quality scores on the mismatches nucleotides (including those beyond the 28 bp size) is added. The best two higher scores produced for each read are then stored together with the reference position. If a read obtains the same quality score on two positions, the algorithm chooses the smaller hash index, as a way to achieve pseudo-randomness to the selection.

Bowtie. Bowtie is a memory efficient tool that aims at aligning short DNA reads to produce a consensus sequence[34]. It relies on the Burrows-Wheeler transformation and the Full-Text Minute-Space (FM) Index on which the reference genome is represented as a permutation of characters with some auxiliary data structures. By using this transformation and auxiliary data, each nucleotide in a read is looked up into the permutations transformation. Due to the special arrangement produced by the Burrows-Wheeler transformation, whenever a suffix is expanded on the read, the range of the lookup matrix is reduced. The method finds all possible exact matches between a read and the reference genome. Since the suffix scanning is performed in $\mathrm{O}(N)$ time, the algorithm incorporates pre-calculated nucleotide tallies of the reference transformation to reduce the scan into constant time by adding very little space. A problem arises when reads are not exact matches. In this case, Bowtie performs sequence backtracking based on nucleotides quality values. Low quality values suggest that the sequenced nucleotide could be a mismatch. In such a case, the algorithm does a nucleotide substitution in that position, and continues the matching process forward. Since the search is greedy, the first match is the one selected. If no match is found, the algorithm keeps on backtracking based on a previously established backtracking policy. Unlimited backtracking is controlled by different indexing technologies like double indexing and MAQ-like search, however there is also a backtracking limit in place. Due to the backtracking and the6 greedy approaches, Bowtie could not guarantee an optimal alignment and it is considered a heuristic algorithm. However, due to its fast execution it is currently one of the most widely used assemblers.

SSAHA. The Sequence Search Alignment by Hashing Algorithm (SSAHA) that uses an extensive hashing approach to complete fast searches against the reference genome[35]. The reference genome is pre-processed by breaking it into consecutive non-overlapping k-tuple. Each k-tuple is then hashed as a 24 bit key. The list for that key contains all the positions on the reference genome that have that k-tuple. For each read, a set of new k-tuple is then generated (this time they are consecutive and overlapping) and a new hash table is created for each read with the position of the reference genome that matched the read k-tuple. Then, the offset of the read k-tuple is removed and each index is sorted accordingly. Once sorted, the list order contains each read k-tuple exact match in the reference genome and its specific position in the reference.

This chapter presented the computational challenges for unbalanced local sequence alignments and showed how these constraints are relevant to at least two very important problems in biology. It was shown how these problems are handled today by heuristics approaches that sacrificed accuracy by speed. In the next chapter, a new algorithm designed for unbalanced sequences that provides near optimal alignments is proposed.

3. Thesis Objectives

1. Design and implement a new optimal sequence alignment technique that uses Smith-Waterman algorithm at its based but that can reduce execution time on highly dissimilar sequence lengths
   (a) Specific objectives
      i. Mathematical Correctness - algorithm must be able to detect optimal solution
      ii. Execution Times - algorithm must be considerably faster than common Smith-Waterman implementations on highly dissimilar sequence lengths.
   (b) Metrics
      i. Algorithm accuracy results (scores and alignment positions) will be compared against known Smith-Waterman implementations using same inputs.
      ii. Algorithm executions times will be compared against known Smith-Waterman implementations for same inputs.
2. Develop a benchmark suite that can compare new algorithm accuracy against known tools preferred by the scientific community.
   (a) Specific objectives
      i. Benchmark new algorithm results against BLAST[24] tool.
      ii. Benchmark new algorithm results against Bowtie[34] tool.
   (b) Metrics
      i. Use BLAST and Bowtie returned position and obtain an score against that position.

4. Proposed Algorithm

An algorithm that performs an *improved search for a local alignment* (ISLA) and uses a probability to assess the quality of the approximation will be proposed on this chapter. The method finds approximate Smith-Waterman (SW) solutions faster than a direct application of SW, and is scalable. ISLA achieves this by exploiting two properties of an alignment of a short sequence X and a significantly larger sequence Y. The first is a *localization property*, which reduces computations to segments of Y of length $|X|$ instead of the whole Y. The second is the existence of a method to search for the best approximation to the exact solution. The method allows a significant amount of parallel computations as well.

4.1 Algorithm Overview

ISLA uses all the exact matches between X and Y to build a profile of matches for finding segments of Y whose alignment with X has a good chance of being a Local Alignment optimal solution, but not to construct the solution. Each entry in the profile of matches is an upper bound of the number of matches in the local alignment. ISLA starts with a selection of a segment of Y with the highest match profile value and then computes a local alignment using SW against that segment. The score, number of matches and length of the alignment are extracted and used to initialize the search space reduction. The latter is performed with an iterative method, which performs one SW computation at each step. If a higher score is found, the method reduces the number of Y-segments of interest, by eliminating entries on the profile of matches. ISLA computes SW on each of the remaining segments and returns the ones with the highest score.

4.2 Definitions

This section will document some formal definitions that will be used in the rest of the chapter. These definitions are the building blocks for the propositions that will be developed later.

4.2.1 Local Alignment

The first definition is of a local alignment between sequence X and Y, which will be defined with the symbol Λ, in specifically $\Lambda(X, Y)$.

4.2.2 Length of an alignment

The number of columns in a local alignment or alignment length L is defined by:

$$L = m + s + g, \tag{4.1}$$

where m will be the quantity of matches, s the quantity of substitutions and g the quantity of gaps inserted on the resulting alignment. In the worst case scenario, a resulting alignment length between sequences X and Y can be up to $|X| + |Y|$, but in practice we see lengths in the range of $|X| < L < |Y|$ in our case of interest where $|Y| \gg |X|$.

4.2.3 Substitution Matrix

As explained on section 2.1.1 the quality of a local alignment is assessed with a *scoring scheme*, which is a pair (S, f) of a *substitution matrix* and a *gap penalty* map. In DNA alignments $S = [S(\alpha, \beta)]$, $\alpha, \beta \in \{A, C, G, T\}$ is defined as

$$S(\alpha, \beta) = \begin{cases} r, \text{ if } \alpha = \beta \text{ and} \\ \omega, \text{ otherwise,} \end{cases} \tag{4.2}$$

where $r > 0$ is the reward for a match and $0 < \omega < r$ is the cost of a character substitution or mismatch. In turn, the gap penalty map is defined as

$$f(g) = \begin{cases} \gamma + (g-1)\eta, \text{ if } g > 0; \\ 0, \text{ otherwise,} \end{cases} \tag{4.3}$$

where g is defined (as in equation 4.1) as the length of the gap sequence, this is the number of gap symbols between two consecutive characters. On a perfect match between 2 sequences g is zero. The constant γ is defined as the cost of the first gap insertion between symbols. It is also known as *open gap penalty*. The other constant η is the cost of subsequent gap insertions or *extended gap penalty*. In the particular case where $\eta = \gamma$ the gap penalty map is called *linear*. Otherwise, is said to be *affine*.

The defined ranges for each of these constants are $0 < \gamma < r$ and $0 < \eta \leq \gamma$. From now on, this DNA scoring scheme will be referred as (S, f), with S as in (4.2) and f as in (4.3).

4.2.4 Score of an Alignment

The score of and alignment between a sequence X and a sequence Y with respect to a given DNA scoring scheme is defined as

$$sc\left(\Lambda(X,Y)\right) = r \cdot m - \omega \cdot s - \sum_{i=1}^{q} f(g_i). \tag{4.4}$$

Here q is the number of gap sequences inserted in the alignment, while m and s are the number of character matches and substitutions as defined on 4.1. In local alignments, the score is required to be positive. Thus, a formal definition of local alignment between X and Y is the 2-row array that satisfies the following conditions:

1. The first column of the alignment is a match;
2. No gap symbol is aligned with a gap symbol; and

3. $sc(\Lambda(X,Y)) > 0$.

4.3 Propositions

To show ISLA mathematical correctness, some propositions will need to be defined first. This section will establish these propositions and will use the definitions presented on section 4.2

Proposition 4.3.1. *Let* $\alpha = \min\{\omega, \gamma\}$. *Then,*

$$L < m\left(1 + \frac{r}{\alpha}\right). \tag{4.5}$$

Proof. Since $sc(\Lambda(X,Y)) > 0$ we have

$$\begin{aligned} 0 &< r \cdot m - \omega \cdot s - \gamma \cdot g \\ &\leq r \cdot m - \alpha(s+g). \end{aligned} \tag{4.6}$$

Thus,

$$s + g < \frac{r}{\alpha} m. \tag{4.7}$$

By replacing $s+g$ in L with the bound (4.7) the following relationship is obtained:

$$\begin{aligned} L &= m + s + g \\ &< m + \frac{r}{\alpha} m = m\left(1 + \frac{r}{\alpha}\right). \end{aligned} \tag{4.8}$$

Corollary 4.3.1. *All local alignments that start with a match between a character in* X *and* Y_j *in* Y *are a local alignment between* X *and* $Y[j : j + m \cdot (1 + (r/\alpha)]$. *In particular, all local alignments between* X *and* Y *are a local alignment between* X *and* $Y[j : j + |X| \cdot (1 + (r/\alpha)]$

Proof. Direct consequence of (4.3.1) and the fact that $m \leq |X|$.

Proposition 4.3.2. *Let* $\alpha = \min\{\omega, \gamma\}$ *and* $\beta = \max\{\omega, \gamma\}$. *Then the score to match ratio of an alignment is bounded by mappings:*

$$\phi_\alpha(u) = -\alpha u + (r + \alpha), \text{and} \tag{4.9}$$

$$\phi_\beta(u) = -\beta u + (r + \beta) \tag{4.10}$$

Proof. The ratio between the score and the matches is a measure of similarity. From equation 4.4 and dividing the whole equation by m the following relationship is obtained:

$$\frac{sc}{m} = r - \frac{\omega \cdot s - \gamma \cdot g}{m}; \tag{4.11}$$

It can be seen the ratio approaches its maximum r if s and g are small and m is large. Since $\alpha = \min\{\omega, \gamma\}$ and $\beta = \max\{\omega, \gamma\}$ and $1 \leq u < 1 + r/\beta$, then we can establish the score ratio upper and lower bounds for the alignment by:

$$\phi_\alpha(u) = -\alpha u + (r + \alpha), \text{and} \tag{4.12}$$

$$\phi_\beta(u) = -\beta u + (r + \beta) \tag{4.13}$$

It is worth remarking $\phi_\alpha(1) = \phi_\beta(1) = r$, and $\phi_\beta(u) < \phi_\alpha(u)$ for all $1 < u < 1 + \frac{r}{\beta}$. The restriction $u < 1 + r/\beta$ is necessary since $\phi_\beta(u) \leq 0$ for $u \geq 1 + r/\beta$ and thus, is meaningless as lower bound for a positive rational. Figure 4–1 shows how relationships 4.12 and 4.13 behave with these boundaries.

Proposition 4.3.3. *Let L be the length of a local alignment with score sc, and m, the number of matches in this alignment. Then,*

$$\phi_\beta\left(\frac{L}{m}\right) \leq \frac{sc}{m} \leq \phi_\alpha\left(\frac{L}{m}\right). \tag{4.14}$$

Proof. From the definition of sc we get

$$r \cdot m - \beta(s + g) \leq sc \leq r \cdot m - \alpha(s + g), \tag{4.15}$$

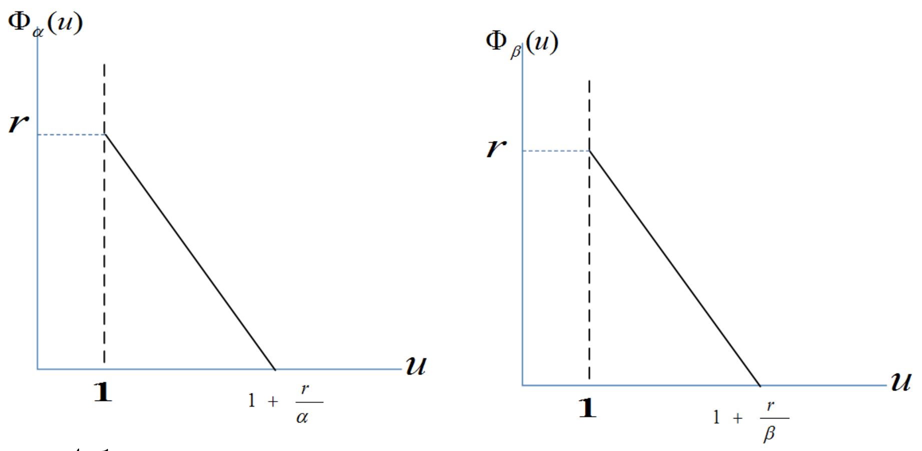


Figure 4–1: ϕ_α **and** ϕ_β **mappings as defined in relationships 4.12 and 4.13**

By replacing $s+g$ with $L-m$ and using a simple algebraic manipulation we get (4.14).

The set of all possible score-to-matches ratios does not admit a total order. The next proposition proves the existence of a partial order.

Proposition 4.3.4. *Let sc_1 and sc_2 be scores of local alignments of lengths L_1 and L_2, and number of matches m_1 and m_2; respectively. If*

$$\frac{L_2}{m_2} < \phi_\beta^{-1}\left(\frac{sc_1}{m_1}\right), \tag{4.16}$$

then

$$\frac{sc_1}{m_1} < \frac{sc_2}{m_2} \tag{4.17}$$

Proof. Since ϕ_β is decreasing, when applied to 4.16 we get

$$\phi_\beta\left(\frac{L_2}{m_2}\right) > \frac{sc_1}{m_1}. \tag{4.18}$$

But

$$\phi_\beta\left(\frac{L_2}{m_2}\right) \le \frac{sc_2}{m_2}. \tag{4.19}$$

We establish next, an explicit relations between matches and scores of local alignments.

Proposition 4.3.5. *Let α be as in Proposition 4.3.1, r as defined in equation 4.2, L as defined in 4.1 and m be the number of matches in $sc(\Lambda(X,Y))$. Then,*

$$m \geq \frac{sc(X,Y) + \alpha \cdot L}{r + \alpha} \tag{4.20}$$

Proof. Since $L = m + s + q$ as defined on 4.1, we can also say that $L - m = s + q$. We also know the score is $sc(\Lambda(X,Y)) = r \cdot m + w \cdot s + q \cdot \gamma$ but since it needs to be greater than zero, we use α to obtain:

$$sc(X,Y) \geq r \cdot m + \alpha(s + q) \tag{4.21}$$

By combining L with $sc(X,Y)$ the following is obtained:

$$sc(\Lambda(X,Y)) \geq r \cdot m + \alpha(L - m) \tag{4.22}$$

which can be re-arranged to obtain:

$$m \geq \frac{sc(X,Y) + \alpha \cdot L}{r + \alpha} \tag{4.23}$$

Corollary 4.3.2. *Let $sc(\Lambda(X, Y[j : j + \kappa \cdot m]))$ be a score with m matches, and let*

$$t_{low} = \frac{b + \alpha \cdot L}{r + \alpha}, t_{high} = \frac{b + \beta \cdot L}{r + \beta} \tag{4.24}$$

Then if $m \leq t_{low}$, $sc(\Lambda(X, Y[j : j + \kappa \cdot m])) \leq b$.

We refer to number t in (4.24) as a *threshold* for the number of matches.

Proposition 4.3.6. *Assume that $\Lambda(X, Y[j_1 : j_1 + \kappa \cdot m])$ has m matches. Let m_{j_i}, $i = 1, ..., r$, be a sequence of positive matches between X and substrings $Y[j_i :$*

$j_i + |X|]$, *such that for each* i, $2 \leq i \leq r$ $m_j = 0$, for $i_{i-1} < j < j_i$. *Define the recursion:*

Base: $A_1 = S_1 = m_{j_1}$.

Recursion: for $2 \leq i \leq r$, *if* $S_{i-1} + \gamma + (j_i - j_{i-1} - 1)\eta > 0$ *set* $S_i = S_{i-1} + m_{j_i}$ *and*

$$A_i = \begin{cases} A_{i-1} + m_{j_i}, \text{ if } A_{i-1} + m_{j_i} < |X|; \\ |X|, \text{ otherwise.} \end{cases} \tag{4.25}$$

If $S_{i-1} + \gamma + (j_i - j_{i-1} - 1)\eta \leq 0$, *stop.*

Let $U(j_1) = \max A_i$. *Then,* $m \leq U(j_1)$.

Assume that b is a lower bound for $msc(X, Y)$ and t is the threshold in 4.24. Then, by Corollary 4.3.2, we eliminate all $U(j) < t$, where $1 - |X| \leq j \leq |Y| + |X| - 1$.

4.4 Probability Model

As shown in Figure 4–2 the graph of the set of pairs $\{(u, \phi_\alpha(u) - \phi_\beta(u)) : 1 \leq u < 1 + \frac{r}{\beta}\}$ is a straight line with origin $(1, 0)$ and end $(1 + \frac{r}{\beta}, r)$. The area below this line is

$$A = \frac{1}{2}\Big(\frac{r^2(\beta - \alpha)}{\beta^2}\Big). \tag{4.26}$$

Even when there is a possibility of scores in the $1 + \frac{r}{\alpha}$ area, those will not be of interest since the amount of matches will be low.

Since each score-to-matches ratio of interest falls within this area, we define the *probability of existence of a higher score-to-match ratio* in the sense of proposition 4.3.4 as

$$P\Big(\rho > \frac{sc}{m}\Big) = \frac{1}{A}\Big(\phi_\alpha\Big(\phi_\beta^{-1}\Big(\frac{sc}{m}\Big)\Big) - \frac{sc}{m}\Big)\Big(\phi_\beta^{-1}\Big(\frac{sc}{m}\Big) - 1\Big), \tag{4.27}$$

where ρ = *score-to-match ratio*, is a random variable. Figure 4–3 shows the relationship between the areas of the whole score-to-match ratio versus the area of the obtained alignment. The rate between these two areas produces equation 4.27.

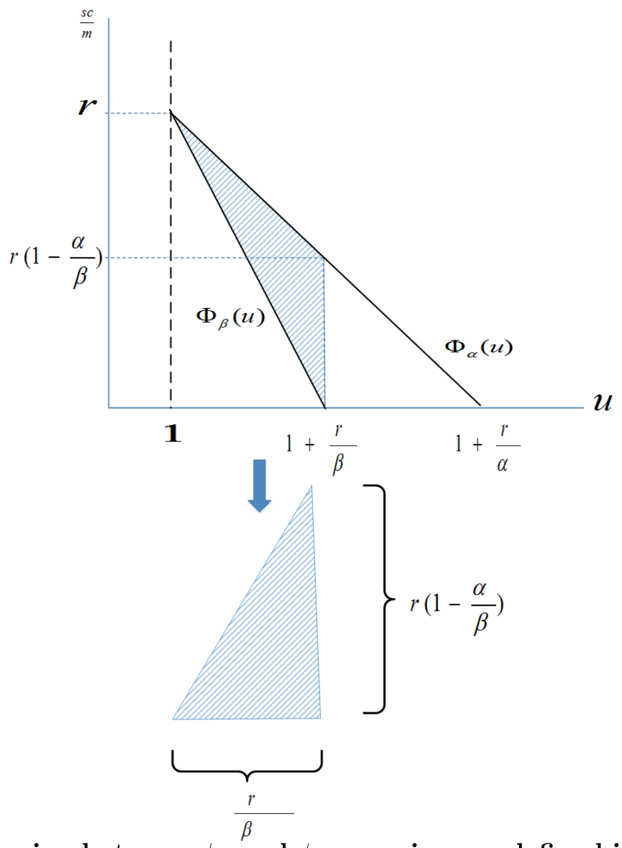


Figure 4–2: **Overlapping between ϕ_α and ϕ_β mappings as defined in relationships 4.12 and 4.13**

Now, by defining a new random variable $\sigma = \textit{score of the alignment of } X \textit{ and } Y$, for a computed $sc(X, Y)$ with m matches and length L, we set the probability for $sc(X, Y)$ to be the *optimal score* as

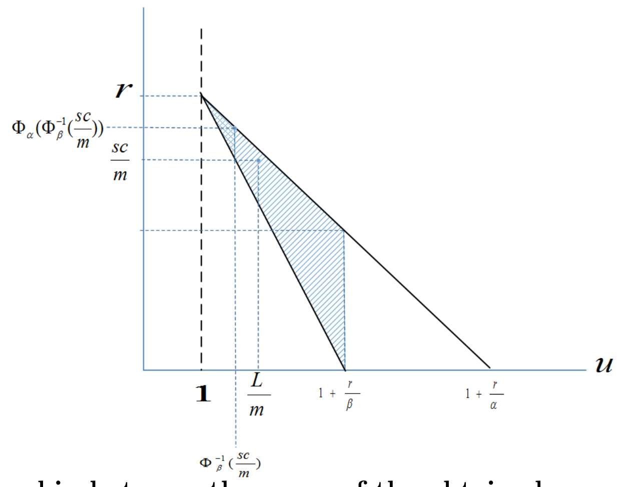
$$P_{\text{op}}\Big(sc(X, Y)\Big) = P\Big(\rho > \frac{sc(X, Y)}{m}\Big) \tag{4.28}$$

It is worth noticing that $P_{\text{op}}(sc_{X,Y}) \geq 0$ and equals 0 if and only if $sc_{X,Y} = |X| \cdot r$ which is an exact match.

4.5 Methodology

4.5.1 Match Count Vector

ISLA algorithm starts with a match count vector or *mCount*. This vector is calculated based on a well known cyclic convolution algorithm[36]. This section



r
$\Phi_\alpha(\Phi_\beta^{-1}(\frac{sc}{m}))$
$\frac{sc}{m}$
u
1
$\frac{L}{m}$
$1+\frac{r}{\beta}$
$1+\frac{r}{\alpha}$
$\Phi_\beta^{-1}(\frac{sc}{m})$

Figure 4–3: **Relationship between the areas of the obtained score-match ratio versus the area from Equation 4.26**

provides a detailed explanation on how this was achieved for two DNA sequences. Same approach can be used for proteins or mRNA sequences.

Assuming X and Y are the following sequences:

$$X = AGCT$$

$$Y = CGTAGCTCTA$$

On position (or index) zero of sequence Y (or Y_0) we have:

```
AGCT-------
CGTAAGCTCTA
```

A similarity match can be found by simply counting the symbols that are the same on each position of the sequence Y. For example, when $x_1 =' G'$ there is a match (since y_1 is also 'G'), so the similarity matching of Xinto Y at position zero (or $Z[0]$) is 1. On Z[1] there are zero matches and if the process continues the maximum symbol matches between sequence X and sequence Y is found on position $Z[4]$ with a count of 4 symbols matching. Since $|X|$ is also 4 that implies an exact match of X in Y at position 4 (y_4). This function can be defined more properly as:

$$Z[n] = \sum_0^{M-1} f(X[n], Y[(n+k) mod N])$$

where:

$$X[n] = x_n$$

$$Y[n] = y_n$$

$$f(x,y) = \begin{cases} 1, \text{ when x = y} \\ 0, \text{ otherwise} \end{cases}$$

The above approach will give all similarities for each position of Y and the maximum is the best symbol match of X into Y (or vice-versa). However, the computational resources needed is $\mathrm{O}\left(MN\right)$ or $\mathrm{O}\left(M^2\right)$ since sequence X needs to be padded to make its length M. The same result can be accomplished by creating a function for each nucleotide on each sequence to convert into numeric vectors.

$$f_G(x) = \begin{cases} 1, \text{ when x = 'G'} \\ 0, \text{ otherwise} \end{cases}$$

$$f_A(x) = \begin{cases} 1, \text{ when x = 'A'} \\ 0, \text{ otherwise} \end{cases}$$

$$f_T(x) = \begin{cases} 1, \text{ when x = 'T'} \\ 0, \text{ otherwise} \end{cases}$$

$$f_C(x) = \begin{cases} 1, \text{ when x = 'C'} \\ 0, \text{ otherwise} \end{cases}$$

Using these functions numeric vectors can be created for each symbol of the nucleotide alphabet G, A, T, C

$X_G = x_{G0}x_{G1}....x_{G(M-2)}x_{GM-1}$

$x_{Gi} = f_G(x_i)$

$X_A = x_{A0}x_{A1}....x_{A(M-2)}x_{AM-1}$

$x_{Ai} = f_A(x_i)$

$X_T = x_{T0}x_{T1}....x_{T(M-2)}x_{TM-1}$

$x_{Ti} = f_T(x_i)$

$X_C = x_{C0}x_{C1}....x_{C(M-2)}x_{CM-1}$

$x_{Ci} = f_C(x_i)$

$0 \leq i \leq M-1$

Based on our previous example:

$$X = AGCT$$

$$X = CGTAGCTCTA$$

X_A will be 1000 for sequence X and Y_A will be 0001000001 for sequence Y. This manipulation can also be applied to the Y sequence to obtain Y_G,Y_A, Y_T and Y_C respectively. Using these numeric vectors the count of symbol 'A' matches for each position can be obtained by:

$$Z_A[0] = X_A[0] \cdot Y_A[0] + X_A[1] \cdot Y_A[1] + ... + X_A[M-1] \cdot Y_A[M-1]$$

$$Z_A[1] = X_A[0] \cdot Y_A[1] + X_A[1] \cdot Y_A[2] + ... + X_A[M-1] \cdot Y_A[0]$$

In its generic form the function will be become:

$$Z_A[n] = \sum_{m=0}^{M-1} X_A[m] \cdot Y_A[(n+m) mod M]$$

$$0 \leq n \leq M-1$$

The same functions can be defined for all other letters.

$Z_C[n] = \sum_{m=0}^{M-1} X_C[m] \cdot Y_C[(n+m) mod M]$

$Z_T[n] = \sum_{m=0}^{M-1} X_T[m] \cdot Y_T[(n+m) mod M]$

$Z_G[n] = \sum_{m=0}^{M-1} X_G[m] \cdot Y_G[(n+m) mod M]$

By simply adding all matches for all four nucleotides on each position the total match for each position is obtained.

$Z[n] = Z_G[n] + Z_A[n] + Z_T[n] + Z_C[n]$

$0 \leq n \leq M-1$

Thus, $maximum(Z)$ is the maximum number of symbol matches between sequence X and Y. The index of that maximum will be the position in Y when that maximum happened.

At this point the computation still $\mathrm{O}\left(M^2\right)$ and the problem still exist. However when we look into the generic form of the approach:

$Z[n] = \sum_{m=0}^{M-1} X[m] \cdot Y[(n+m) modM]$

it can be seen it is very similar to a circular convolution calculation. By defining Y as its reverse, a convolution equation can be obtained:

$Y_R = Y_{M-1}Y_{M-2}...Y_1Y_0$

Thus, now we have:

$Z[n] = \sum_{m=0}^{M-1} X[m] \cdot Y_R[(n-m) modM]$

Since this is the same as a circular convolution definition, a discrete fast fourier transform (FFT) can be used:

$Z = IFFT(FFT(X) * FFT(Y_R))$

Thus now we can obtain the symbol similarity vector using a complexity of:

$4 * [2 * \mathrm{O}\left(MlogM\right) + \mathrm{O}\left(MlogM\right) + \mathrm{O}\left(M\right)]$

The complexity needs to be multiplied by 4 since the calculation needs to be performed for each nucleotide symbol.

4.5.2 Percentile trimming

Once the *mCount* (or M vector) is calculated, it is then trimmed to remove low m positions that have small chances of containing the best match or optimal SW alignment. This filtering is based on a statistical percentile and it just assigns to zero all values below certain threshold. Since the match count depends on the query sequence and the reference sequence, this value is calculated dynamically and it is unique for each M vector. Figure 4–4 shows 4 different short sequences boxplots of M vectors that were produced against the same long sequence. ISLA percentile is set to default on the 3^{rd} quartile so the lower $\frac{3}{4}$ of the matches are changed to zero. Relationship 4.29 shows the calculation the new vector M' based on previous match-count vector M. This step can also be visualized on figure 4–5

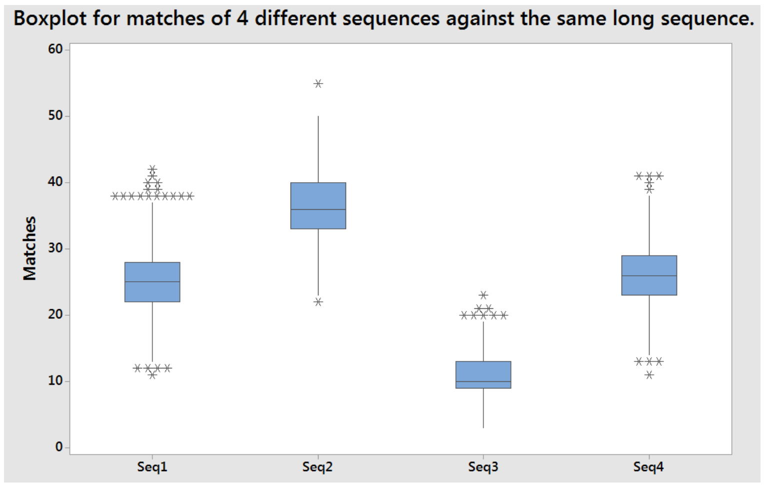


Figure 4–4: **Boxplots of matches of 4 different short sequences against the same long sequence**

$$M' = \begin{cases} m, m \geq percentile, m_i \in M \\ 0, \text{ otherwise} \end{cases} \tag{4.29}$$

4.5.3 Matches Profile

Once M' is calculated ISLA algorithm produces a profile for all the matches remaining after percentile filtering. A simple process, called *pMatches*, transforms M' into a list $\mathscr{M}$ of triples

$$\mathscr{M} = \langle (j, m_j, d_j) : j = 1, ..., |M'| \rangle; \tag{4.30}$$

where for each $j = 1, ..., |M'|$, j is the index in M' with m_j matches and a calculated penalty of d_j. d_j is obtained by:

$$d_j = \gamma + (j + 1 - j + i)\eta \tag{4.31}$$

d_j is simply the cost of inserting a sequence of gaps between the m_j and the next nonzero entry m_{j+i} in M'. If $m_{j+1} \neq 0$, then $d_j = 0$ since there will be no penalty between consecutive matches. However, if $m_{j+1} = 0$ then m_{j+2} will be checked to

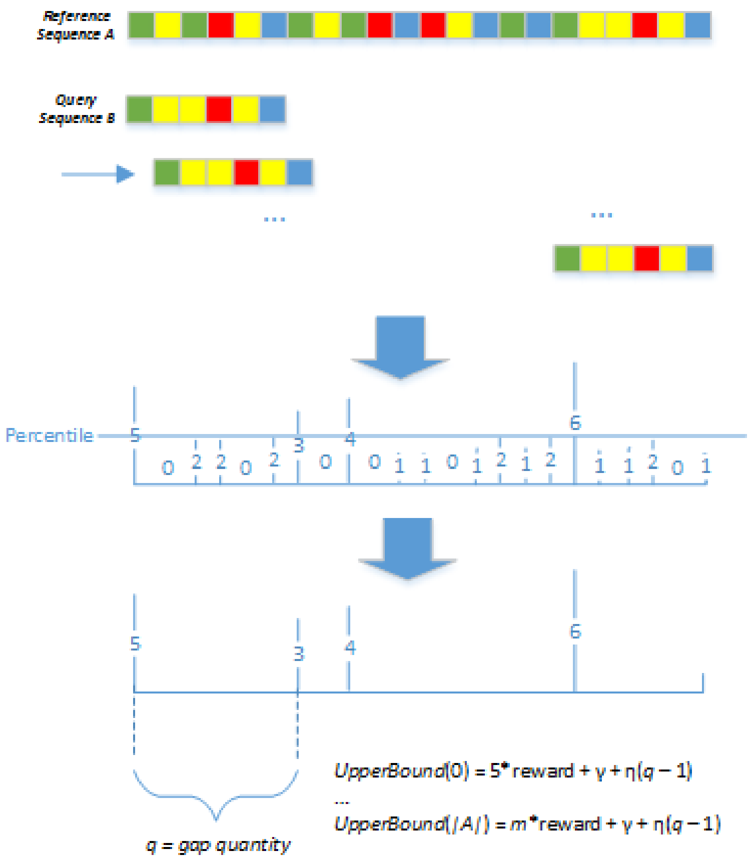


Figure 4–5: **ISLA percentile filtering approach. It removes non significant matches from mCount and applies a score projection**

calculate the number of gaps. The process will continue until $m_{j+i} > 0$ is found and the quantity of gaps and overall penalty for m_j is calculated. This process is bounded by $\mathrm{O}(|Y||X|)$ but the constant are just small fractions of Smith-Waterman's since m_j itself is bounded by $|X|$ and there is no need to continue looking for gaps beyond that distance. The space complexity is bounded by $\mathrm{O}(|Y|)$ since only M' and $\mathscr{M}$ of size $|Y|$ are used.

The objective for the $\mathscr{M}$ list is to provide enough information for each position on M' so an score bound can be calculated.

4.5.4 Score Projection

Using proposition 4.3.6 which give us an score projection based on matches and proposition 4.3.5 a score projection vector is calculated using the generated $\mathscr{M}$ vector. This projected score vector (or PS) contains an estimated scored value based on the matches and penalties obtained on $\mathscr{M}$. As with the match profile calculation on sub section 4.5.3 the score projection is calculated for all ps_j elements of vector PS. For each position ps_j the triple obtained from $\mathscr{M}$ is used in the form of:

$$(j, m_j, d_j) = \mathscr{M}_j \tag{4.32}$$

$$ps_j = \sum_{i=j}^{i<|X|} (r \cdot m_j - d_j) \tag{4.33}$$

Even when the calculation of ps_j may continue until $|X|$, the projection will stop if at some point $ps_j < 0$. In that case ps_j will not be part of PS. Only final calculated $ps_j > 0$ will be included on PS. The final vector PS will contain a list of tuples with the position j and the projected score ps_j. PS is sorted in descending order using ps_j as sorting key. As in the match profile calculation, this process is bounded by $\mathrm{O}(|Y||X|)$ with small constants, space complexity is also bounded by $\mathrm{O}(|Y|)$.

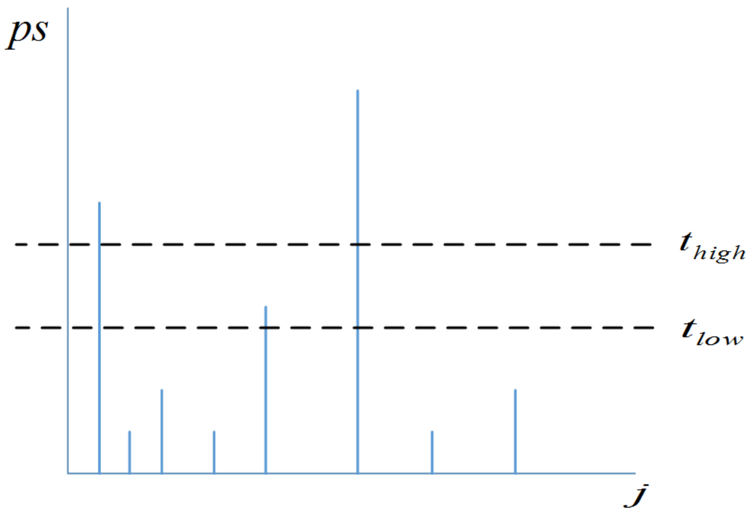


Figure 4–6: t_{high} **and** t_{low} **thresholds helps to trim further the** PS **vector by removing lower projected scores**

4.5.5 Threshold Calculation

Once PS vector is obtained and sorted in descendant order based on projected scores, its first element contains the higher ps_j. Using this first score, a SW is performed against X and a subsequence of Y with a starting position $\bar{j}$ (associated with with the higher ps_j) up to $(1 + \frac{r}{alpha}) \cdot |X|$ as established on proposition 4.3.1.

Also based on proposition 4.3.5 and using equations defined on (4.24), thresholds t_{low} and t_{high} are calculated and PS vector is trimmed further. The initial SW score and alignment length is used to calculated the thresholds. First t_{high} threshold is applied and if the resulting $PS \neq \emptyset$ then PS remains. However, if $PS = \emptyset$ after t_{high} is applied, then it t_{low} is applied and will be used.

4.5.6 Iterations

The iterations in ISLA start with the computation of a new value of

$$score = sc(\Lambda(X, Y[\bar{j} : \bar{j} + \kappa \cdot PS(\bar{j})] \tag{4.34}$$

where $\bar{j} \leftarrow \text{argmax}(PS)$ and $k = (1 + \frac{r}{\alpha})$. If the current iteration produces a better alignment than before, both thresholds (t_{low} and t_{high}) are updated and the vector PS is reduced by deleting all $PS(j) < t_{high}$, if any, together with $PS(\bar{j})$. This process continues until:

1. $\mathscr{U}$ is empty
2. no new values of b are produced.
3. Max iteration limit is reached.

Algorithm 4.5.6 shows the basic concept of ISLA. First M vector is calculated using cyclic convolution. This M vector is trimmed by making zero all elements below a percentile and producing M'. A profile for each match is obtained from M' which is used to calculate $\mathscr{M}$. After $\mathscr{M}$ is determined, a score projection vector PS is produced. Threshold t_{high} and t_{low} are calculated and all elements in the PS below t_{high} are removed. If the resulting vector is empty, the t_{low} threshold is used instead. The iteration process will continue until one or more of the exit criteria defined above are reached. Once an alignment is obtained a probability of the chances of a better score based on the number of matches is provided.

Next chapter will provide implementation details on ISLA and the benchmarks obtained.

Algorithm 1 ISLA Algorithm pseudocode

```
 1: procedure ISLA($X$, $Y$, $(S, f)$,$maxTries$,$perc$)
 2:     $alignmentList \leftarrow \emptyset$
 3:     $M \leftarrow$ mCount$(X, Y)$
 4:     $M' \leftarrow percentileTrim(M, perc)$
 5:     $\mathscr{M} \leftarrow$ pMatches($M'$,(S, f))
 6:     $\bar{j} \leftarrow argmax(\mathscr{M})$
 7:     $(score, L, m, \Lambda) \leftarrow SmithWaterman(X, Y[\bar{j} : \bar{j} + \kappa \cdot |X|]$,(S, f))    ▷ $\Lambda$ is the local alignment
 8:     $alignmentList = \{\Lambda\}$
 9:     $\alpha = min(S, f)$
10:     $\beta = max(S, f)$
11:     $t_{low} \leftarrow (score + \alpha|L|)/(r + \alpha)$    ▷ Calculates lower threshold
12:     $t_{high} \leftarrow (score + \beta|L|)/(r + \beta)$    ▷ Calculates higher threshold
13:     $PS \leftarrow$ ScoreProjection$(\mathscr{M})$
14:     $PS \leftarrow PS - \{PS(j) : PS(j) < t_{high}\}$    ▷ Removes bounds less than $t_{high}$
15:     if $PS = \emptyset$ then    ▷ Check for contents
16:         $PS \leftarrow PS - \{PS(j) : PS(j) < t_{low}\}$    ▷ No contents, using $t_{low}$
17:     end if
18:     while $PS \neq \emptyset$ and $cuurTries \leq maxTries$ do
19:         $\bar{j} \leftarrow \text{argmax}\, PS$
20:         $(newScore, L, m, \Lambda) \leftarrow SmithWaterman(X, Y[\bar{j} : \bar{j} + \kappa \cdot |X|]$,(S, f))
21:         if $newScore > score$ then
22:             $alignmentList \leftarrow \emptyset$
23:             $alignmentList = \{\Lambda\}$
24:             $\mathscr{S} \leftarrow (newb, \Lambda)$
25:             $score \leftarrow newScore$
26:             $t_{low} \leftarrow (score + \alpha|L|)/(r + \alpha)$    ▷ Calculates lower threshold
27:             $t_{high} \leftarrow (score + \beta|L|)/(r + \beta)$    ▷ Calculates higher threshold
28:             $PS \leftarrow PS - \{PS(j) : PS(j) < t\}$    ▷ Removes bounds less than $t_{high}$
29:             $P = getProb(m, score, (S, f))$    ▷ Calculates probability of finding a better score
30:         else if $newb = b$ then
31:             $alignmentList = alignmentList + \{\Lambda\}$
32:         end if
33:         delete $PS(\bar{j})$
34:         $currentTries = currentTries + 1$
35:     end while
36:     return $alignmentList$
37: end procedure
```

5. Implementation

The algorithm described on chapter 4 was implemented in a Symmetric Multi-Processing (SMP) system using the Python[37] programming language. Algorithm parallelization opportunities were exploited and different memory efficiency techniques were used to gain better performance and efficient resources usage. This chapter will explain in detail the algorithm implementation, its results and benchmarks when compared to popular BLAST and Bowtie applications.

5.1 Parallelism

The original focus of this research was to be able to obtain a new algorithm for local alignments that used Smith-Waterman but that was fully parallelized without accuracy loss. That goal was achieved and its results published[38]. The resulting algorithm FAMA (for Fast and Accurate Mapping Assembly) was aimed at processing short reads produced by a NGS system and that needs to be mapped to a known consensus sequence. Even though FAMA achieved 100% Smith-Waterman precision and it is highly scalable, its executions times (while a lot better than sequential processing) are still prohibitive to common biologist researchers and it depends on a High Performance Cluster (HPC) architecture. The FAMA algorithm was developed in C programming language and using Message Passing Interface (MPI) as the communication framework. It was a complex algorithm since it required a vast amount of genome subsequences data transfers between master and workers and the communication scheme was complicated from the programming stand point.

The rest of this section will explore the parallelization approaches used on ISLA and the reasoning behind it based on the lessons learned by FAMA.

5.1.1 Python

After FAMA, it was decided to move to an easier programming paradigm and *python* was selected. Python provides tremendous programing flexibility without the burdens of higher level languages. Even when it is known that python is an interpreted language and it is assumed to execute slower when compared against compiled languages such as C and C++, all the benchmarks performed pushed that theory away.

The author executed benchmarks for computational biology common transactions such as reading a FASTA or FASTQ files, performing Smith-Waterman alignments and even FFTW comparisons. Except for the fftw case, all other benchmarks ran faster on python than on the C code implemented by the author. The reason for this is that even when most of the python wrappers are python code, their low-level modules are developed in C and compiled with optimizations for the target system. For example, even when a person can develop a simple Smith-Waterman algorithm in C, it can't compete with python's *swalign* which uses a lot of the numerical libraries at its core, is open source and a lot of developers are constantly improving it. Moreover, Python setups and configurations scripts have advanced a lot in recent years, and most of the time, the flags and compile optimizations used on the configuration scripts, will take a lot of time for a single developer to learn and implement them.

5.1.2 Shared Memory

Operating System Shared Memory was selected as the vehicle for shared data across workers. Figure 5–1 presents an overview of the ISLA shared memory approach. The algorithm relies on independent processing as much as possible and only relies on communication between master and workers just to transfer inputs and results. In the case of the long sequence (which may be a whole genome), communication of chromosomes or scaffolds between master and processes was prohibitively slow due the time penalty and the memory limitations. Since the algorithm was developed on a SMP system even when each process has its own memory space, they share the same quantity limitation. For example, if a 340Mbp chromosome is transfered to all processes, that space needs to be replicated on all processes on the same system's physical memory, which is a major waste of space and communication time. This was the primary reason why the shared memory approach was used for the long sequence.

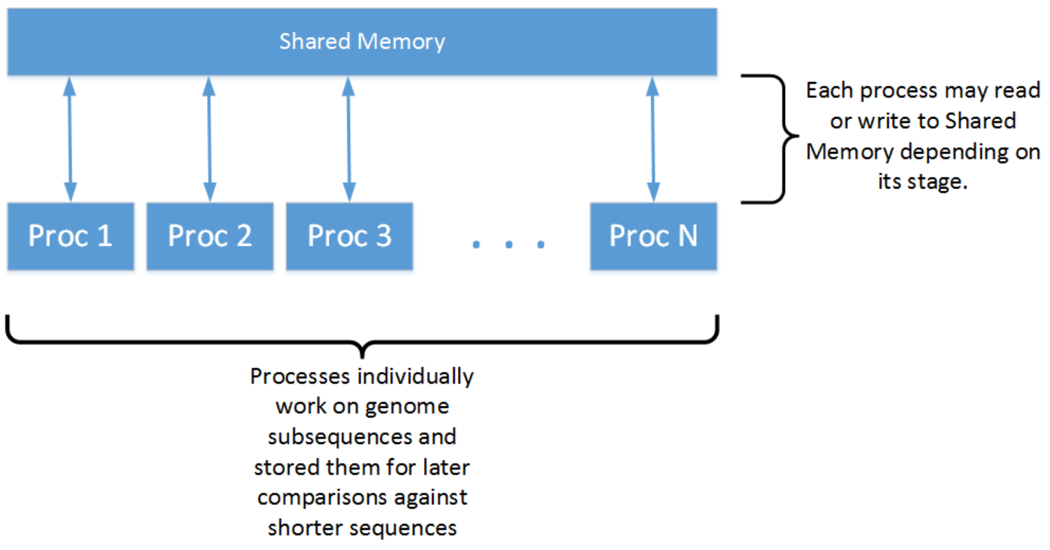

Shared Memory
Each process may read or write to Shared Memory depending on its stage.
Proc 1
Proc 2
Proc 3
. . .
Proc N
Processes individually work on genome subsequences and stored them for later comparisons against shorter sequences

Figure 5–1: **ISLA Algorithm parallel approach. All processes handle each short sequence independent of each other and communicate back results.**

5.1.3 Python Parallelization Framework

Due the flexibility of Python, there exists several solutions for parallel processing. Only on the SMP space more than 10 modules exist, a lot more for cluster architectures and even a few for cloud computing[39].

Even when ISLA algorithm is not attached to any parallelization architecture (e.g. SMP, HPC or cloud), due to the computational resources constraint a SMP approach was selected. For SMP, two Python solutions were evaluated and one selected.

The first solution evaluated was **Parallel Python**. Parallel Python provides a job oriented architecture that can be used for SMP and HPC architectures [40]. It provides a very intuitive and powerful framework which relies on job submission to a *job server*. This *job server* is initialized with the desired number of processors and it handles all requests transparent from users. The job submission takes the function to be executed, its arguments and since each process runs in its own context a list of dependent modules and methods need to be specified at job submission. The submission can also take a callback function which is called with specified parameters at process completion.

Parallel Python worked great for homogeneous loads where all processes shared the same amount of work. However, that is not the case for ISLA algorithm since each short sequence's best alignment computation is dependent on different variables related to sequences comparisons. The first implementation of ISLA was using Parallel Python, but the load balancing control was adding more complexity to the algorithm than what was desired. A callback mechanism was implemented where functions maintained control of states by using global variables once a processes ended. However, it didn't work on some instances. Moreover, the biggest limitation of parallel python was the inability to have query timeouts where the job submitter may query the child process results even if the results are not ready. If the timeout is reached, then the submitter will know the process is not ready yet to provide results.

Due the load balancing complexities encountered with Parallel Python, the final ISLA algorithm implementation is based on the **Python Multiprocessing** module. Python Multiprocessing provides the same capabilities as Parallel Python but with a more robust Application Programing Interface (API). It not only provides processing parallelism, but it also has the data parallelism functionality where a single function can be automatically mapped to several inputs. It also provides basic process control mechanism as semaphores and queues. However, the most important functionality that convinced the author to use the framework is the *timeout* capability. The timeout framework included as part of the API lets the main scheduler to query each process individually even if the process is not ready to answer. This provides great flexibility for unbalanced processes since there is no problem if some of them completed early but others take more time. Algorithm Pseudocode 5.1.3 shows the main queue implementation and how sequences are propagated to different processes.

5.1.4 Genome Processing

Genome processing is also processed in parallel using the same mechanism shown on algorithm 5.1.3. ISLA algorithm takes a reference file which may contain from one to several sequences depending on the organism. The genome (which is normally in a FASTA file format) is divided into individual chromosomes and/or scaffold sequences and these are processed independently of each other. Since the genome is stored in Shared Memory, as explained on section 5.1.2, each genome subsequence is then processed and stored by each process individually (Figure 5–2). Each process uses an unique naming schema to avoid two processes writing to the same space. The name schema is derived from the subsequence name stored on the FASTA file. ISLA assumes that there can't be two different sequences with the same name on a single FASTA file.

5.2 Cyclic Convolution Implementation

The cyclic convolution (explained on section 4.5.1) is accomplished by calculating FFTs for each nucleotide vector. To be able to perform these computations as fast as possible the implementation uses FFTW. FFTW is a C subroutine library for computing the discrete Fourier transform (DFT)[41]. FFTW is one of the most

Algorithm 2 Main Queue pseudocode which uses Python Multiprocessing module

```
procedure MainQueue(SequenceList,totalProcs)
    pool = Pool(totalProcs)
    poolList = $\emptyset$
    currSeq = 0
    for sequence in SequenceList do
        job = pool.submit(processingFunc,sequence)
        pooList = pooList + job
        currSeq = currSeq + 1
        if currSeq == totalProces then                ▷ All processes used
            break
        end if
    end for
    onQueue = 1
    finished = 0
    while onQueue do
        index = 0
        for job in procList do
            try:
                result = job.get(timeout=1)
                processResult(result)
                finished = finished + 1
            except TimeoutError:
                continue
            try:
                sequence = SequenceList.next()
                job = pool.submit(processingFunc,sequence)
                procList[index] = job
                currSeq = currSeq + 1
            except StopIteration:
                procList = procList - result
            if finished $\geq$ currSeq then
                onQueue = 0
                break
            end if
            index = index + 1
        end for
        index = 0
    end while
end procedure
```

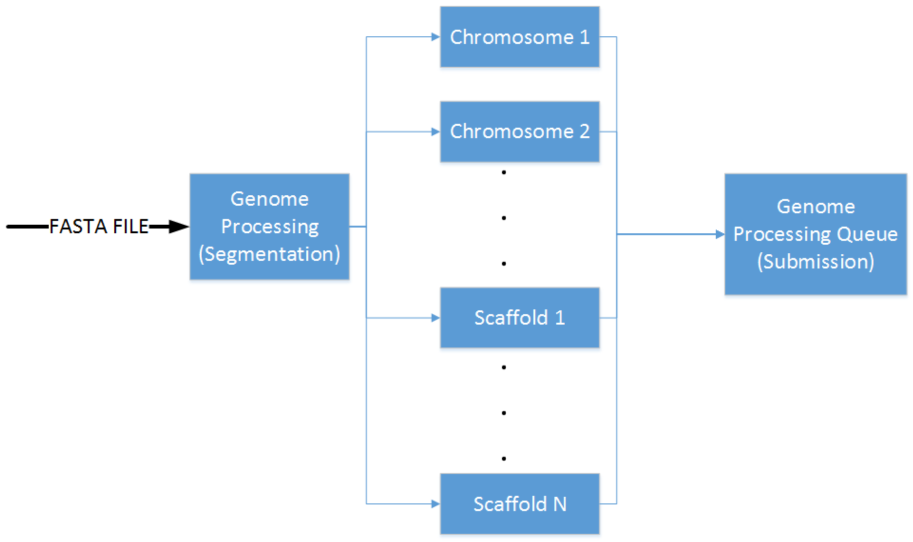

FASTA FILE
Genome Processing (Segmentation)
Chromosome 1
Chromosome 2
Scaffold 1
Scaffold N
Genome Processing Queue (Submission)

Figure 5–2: **Genome processing parallization. As FASTA formatted file is broke in individual sequences and those are then submitted for processing.**

efficient implementations available since it performs runtime queries to the running platform, executes small benchmarks and adapts itself dynamically to perform as fast as possible. FFTW has been ported to all major platforms and python wrappers had been generated for it. This implementation uses Python FFTW (pyFFTW)[42].

The pyFFTW is used to calculate the forward (normal) and backward (inverse) for both: the query sequence (shortest sequence) and the reference sequence (longest sequence). Both sequences need to be converted to frequency domain, but the main challenge was on the longest sequence. In case of a common chromosome or long scaffold, it can easily contain hundred of millions of base pairs.

While the ISLA implementation transforms each genome subsequence into the frequency domain in the same order as they are contained in the FASTA file, each process does not follow the same order. The reason for this is that if a very large chromosome or scaffold is going to be processed concurrently, the different match count vectors generated by each process will have a significant impact on memory. This normally resulted in Out of Memory scenarios. To avoid this, ISLA sorted randomly the genome subsequence so each process is working against different subsequences at the same time.

5.3 Data Used

The data used on all the benchmarks and results on this project is a Stickleback fish (*Gasterosteus aculeatus*) genome downloaded from the Ensembl genome browser for vertebrate genomes[43]. The reason for selecting that specific genome is because it was used by the author on another assembly project which targeted mapping reads produced by NGS to a set of similar organisms and the Stickleback was one of them. The Stickleback genome is approximately 460 Mb in length and contains 22 pairs of chromosomes (groups) including a mitochondrial chromosome and an additional 1,822 unplaced supercontigs.

Due execution time constraints, the results discussed on section 5.5 are not based on the whole genome. For accuracy measures a single scaffold (scaffold_1653) of the genome was used, and for accuracy benchmarks against ISLA, BLAST and Bowtie a super contig called groupVI was used. The scaffold *scaffold_1653* contains 1924bp and is used as benchmark comparisons between full Smith-Waterman algorithms. The super config *groupVI* is a 32Mbp sequence and it is used for long sequence accuracy and execution times.

5.4 Environment

5.4.1 Hardware

All benchmarks and implementation were executed in a HPE BL920 Gen8 system with 480 logical cores and 1 TB of RAM[44]. The system consist of 8 individual blades that are connected and configured via an common backplane. The system firmware allows it to be configured based on specific user needs, up to 8 individual systems or 1 system that includes all 8 blades. In the case of this research, all 8 blades were configured as a single system. Each blade contains 2 Intel's x86_64 EX CPUs, and each physical processor contains 15 cores (30 cores per blade). Since the system has hyper-threading enabled, each blade contains 60 logical cores and thus 480 overall.

The algorithm is total agnostic to this specific configuration, and its only input which is relevant to the architecture is the maximum process count on which 480 is used. However, since the 1TB is shared across all process, there were cases where the amount of process used were configured to be less to avoid Out of Memory scenarios.

5.4.2 Programming Environment

The operating system on the server is SUSE Linux Enterprise Server 12 and no changes was required except for the installation of python needed modules.

The python version used was 2.7 but there was no single requirement for it. It just happens it was the version installed on the system. The following table shows a list of the python modules used on the ISLA implementation.

Module	**Version**	**Description**
multiprocessing	(part of python 2.7)	Provides ISLA parallelization API
numpy	1.8.0	Numerical Library for Python. All vectors in ISLA are numpy vectors.
pyFFTW	0.10.4	FFTW Python wrapper, used for initial match calculation
BioPython	1.67	Used for sequence file readings (FASTA and FASTQ)
SharedArray	1.0	Used to store numpy vectors in shared memory
swalign	0.033	Python SmithWaterman implementation

Table 5–1: **Python modules used by ISLA implementation**

5.5 Results

The following section will cover the ISLA algorithm implementation results. The section is divided into 3 main area of interest: (1) Algorithm Accuracy, (2) Algorithm Execution Times and (3) Benchmarks against known tools. On the first subsection the validation of ISLA mathematical correctness is shown by comparing ISLA results with Smith-Waterman algorithm results, the second section will compare ISLA algorithm execution times with Smith-Waterman execution times and third will provide benchmarks between ISLA algorithm accuracy, Bowtie assembler and BLAST tools.

5.5.1 Algorithm Accuracy

Since ISLA is an iteration based algorithm, its accuracy increases based on the number of iterations. As shown on Figure 5–3, while the number of iterations keep increasing, its accuracy keeps growing until it reaches a 100%. Accuracy is measured by comparing ISLA results with full SW alignment results. To have a hit, the final score obtained by ISLA should be the same as the score produced by SW running against the whole long sequence search space.

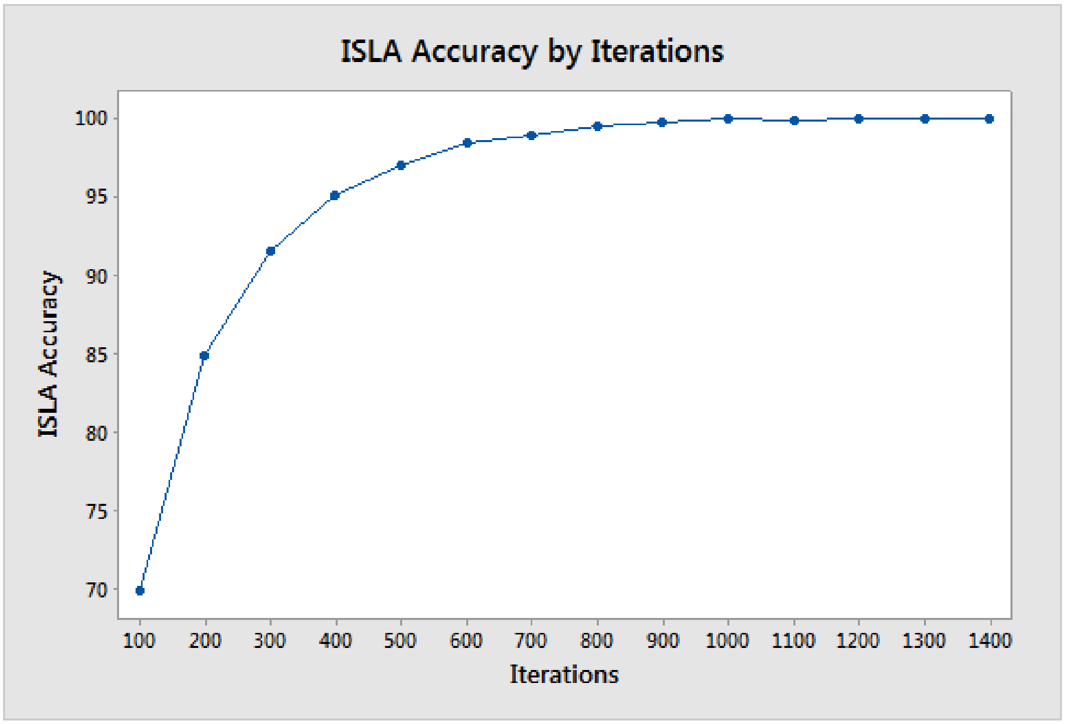


Figure 5–3: **ISLA Accuracy by iterations.**

These results were obtained using *scaffold_1653* and a Illumina set of 1500 reads obtained from the same organism. The scoring matrix used a reward of 2, and a substitution penalty of -1. The open gap parameter used was -1 with an extended gap of -0.5.

As expected, an increase in iterations affects ISLA execution times. Figure 5–4 shows the increase in execution time by the increase in iterations. These are the same runs that provide the accuracy results shown on Figure 5–3 and using the same SW parameters.

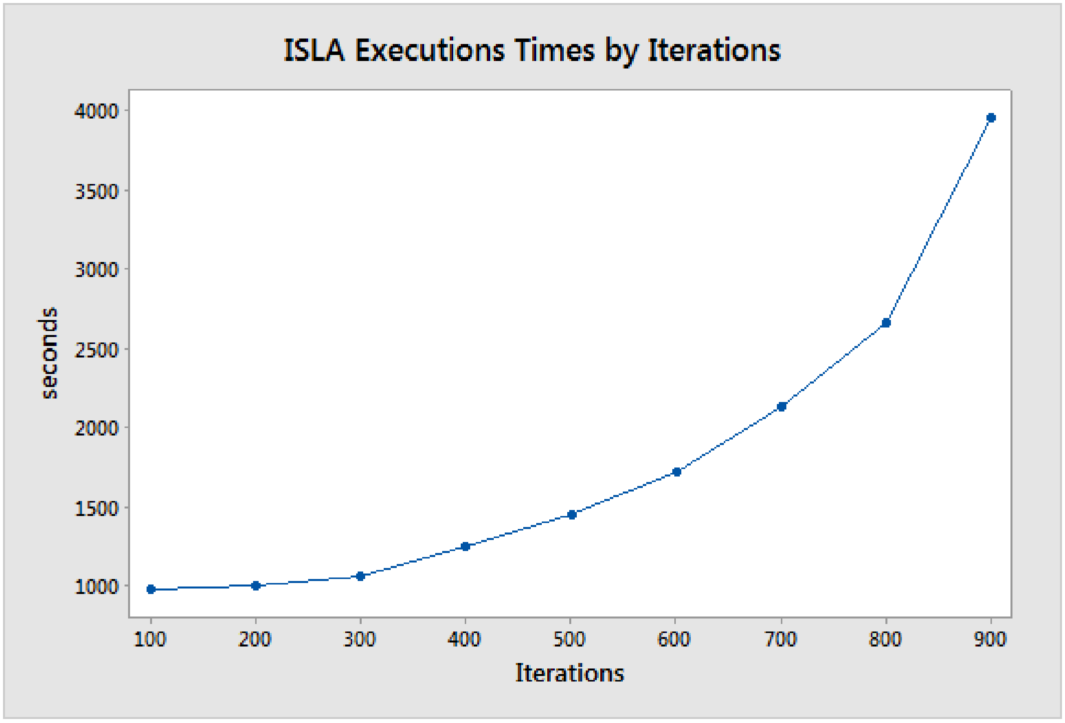


Figure 5–4: **ISLA execution times by amount of iterations**

5.5.2 Execution Times

Table 5–2 shows the execution times of different implementation runs of Smith-Waterman algorithm. All implementations use the same SW algorithm provided by python *swalign* module as their base. The runs used 16 random generated sub-sequences from the super contig groupVI contained on the genome discussed on section 5.3. Appendix A.2 shows the bash script used to generate those reads from the specific reference file. The script was used to generate 16 short sequence of sizes between 35bp and 50bp. These were obtained from random positions on the super contig. For each one of those subsequences random mutations were added. Mutations include insertions, deletions and SNPs. Each subsequence had at most 2 of these mutations at random positions in the subsequence.

Implementation	**Execution Time**
Sequential Smith-Waterman	∼3 days
Parallel Smith-Waterman	∼8 hours
ISLA (200 iterations)	∼15 minutes

Table 5–2: **Execution times comparisons**

The Sequential SW implementation processed all 16 reads in a sequential way using a single processor. The Parallel SW implementation process all subsequences concurrently. This implementation was the one that had the constraint of using a small number of short sequences since the long sequence (32Mbp) in this case was replicated across all nodes and the more subsequences processed, the more overall memory it needs, reaching an Out Of Memory scenario even with small quantity of subsequences. The ISLA implementation used 200 iterations. For these runs, accuracy was not measured.

5.5.3 Benchmarks

On this section an analysis is performed between ISLA, Bowtie and BLAST results. As explained on sections 2.3.1 and 2.3.2, Bowtie and BLAST are two of the most used tools on the sequences comparison domain. Both tools rely on reference sequence(s) indexing and exact matches search is performed in linear time. Even when their results are based on regions of high homologies instead of alignments they are preferred by the scientific community due to its fast execution times.

For these benchmarks the probability of finding a better match (as described in section 4.4) is used against the results obtained from Bowtie and BLAST. Since Bowtie and BLAST do not provide alignments per se, an SW algorithm is executed based on Bowtie and BLAST returned positions and probabilities are calculated based on its results. These probabilities will be compared to the optimal alignment probability and a distance will be obtained. These distances are plotted on histograms. In theory, an implementation obtaining same results as an optimal Smith-Waterman, will contain a single histogram bin located near zero. This will mean there are zero distances between the probabilities calculated by the algorithm under study and the optimal Smith-Waterman alignment.

The short and long sequences used for these benchmarks are created synthetically based on the script included on Appendix A.2. This script generates a random long sequence and generates subsequences based on a short/length ratio. For all these benchmarks the ratio used was 0.01, so for a reference long sequence of 1,000,000bp subsequences of sized 10,000bp were produced. All subsequences were obtained based on random positions on the long sequences, and all of them contained a mutation rate. As mentioned before, mutations are nucleotides insertions, deletions and SNPs. These mutations are selected randomly and inserted in random positions in the subsequence. Five hundred subsequences were generated in total, and the same subsequences are used against the three methods. The benchmark varies the mutation rate and shows algorithm results based on the following rates: 1%, 5%, 10%, 15% and 20%.

Figure 5–5 shows the results comparisons for a 1% mutation rate. As observed, all 3 methods show almost optimal results since the distance between the optimal SW alignment probability and their results is mostly zero. As per the results, BLAST and Bowtie were not able to provide best position on only 3 of the 500 reads. ISLA was able to provide the optimal positions on all of the reads. Since ISLA achieved optimal results on only 50 iterations, no further runs with more iterations were performed.

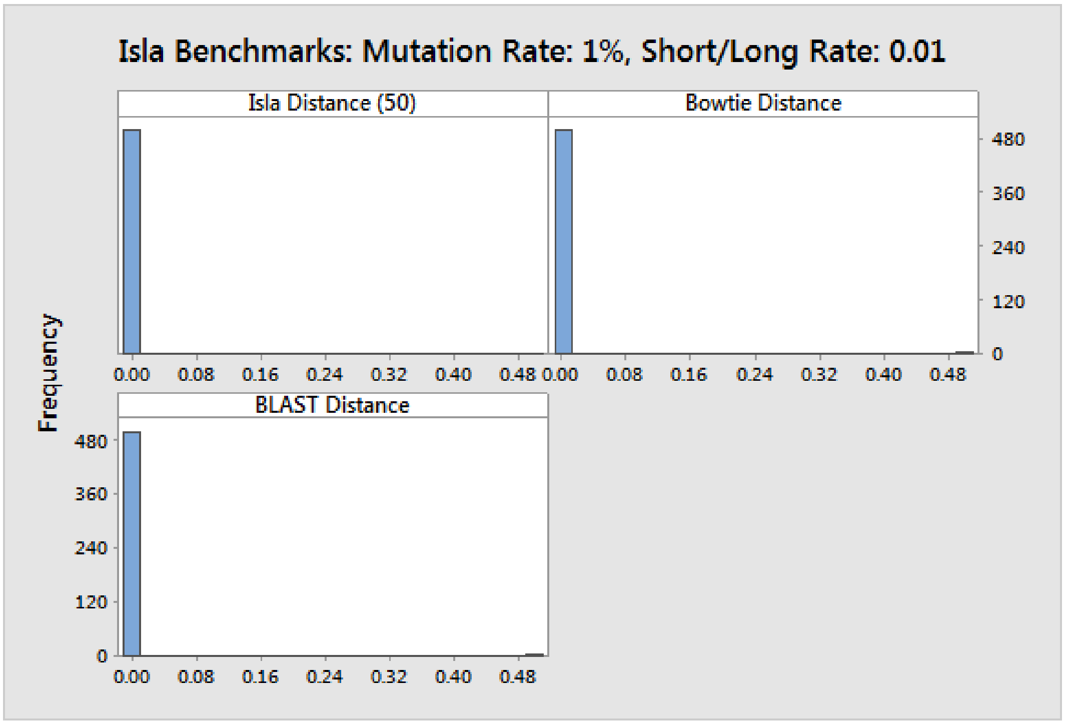


Figure 5–5: **ISLA, BLAST and Bowtie Benchmarks for a 1% mutation rate**

Figure 5–6 shows ISLA results using 5% mutation rate. As expected, an increment in iterations produced an increase in accuracy. ISLA's results with 50 iterations produced 347 optimal alignments, while with the 600 iterations increased to 436. On the other hand, figure 5–7 shows the results for Bowtie and BLAST for the same reads. As it can be seen, Bowtie was able to get optimal positions on 399 of the reads, and BLAST achieves an impressive 456 optimal positions.

Figure 5–8 shows ISLA results for subsequences with a 10% mutation rate. The trend continues where an increase in iterations yields better accuracy and thus better probability distances. For 50 iterations, ISLA was able to achieve 275 optimal positions and 387 with 600 iterations. On Figure 5–9 BLAST and Bowtie results can be observed. Here, it can be seen how Bowtie starts fading out on results, and a new bin near 1 start forming. This bin gathers all subsequences that didn't produce a Bowtie hit. Bowtie only produced 159 optimal positions, while BLAST produced

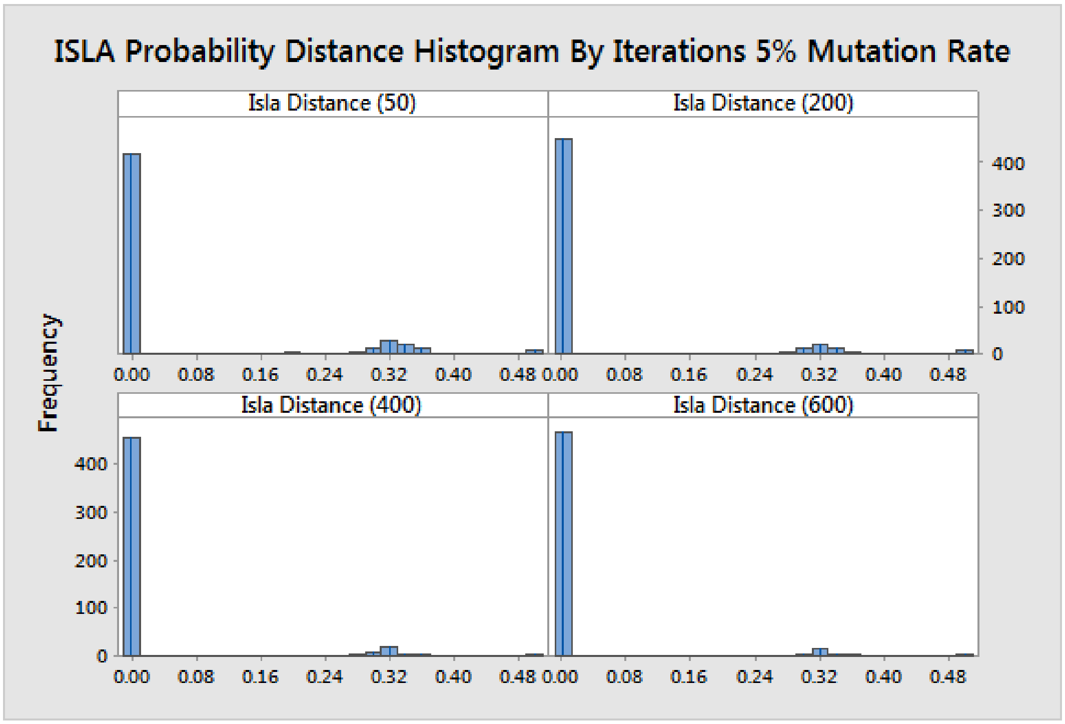


Figure 5–6: **ISLA probability distances for 5% Mutation Rate**

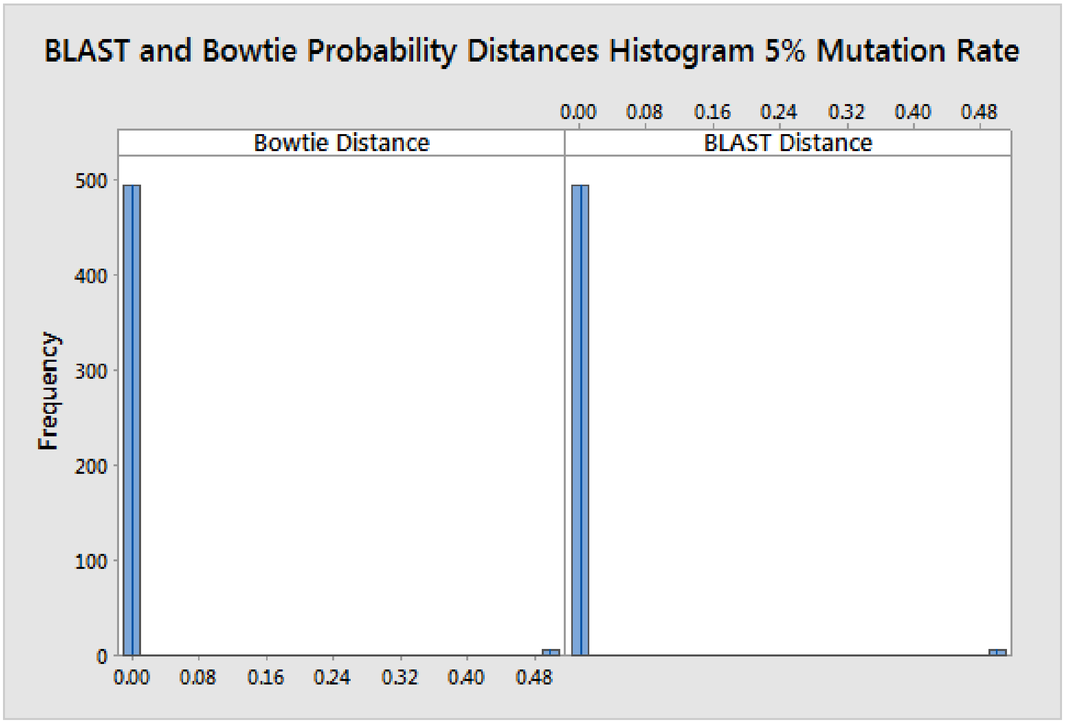


Figure 5–7: **BLAST and Bowtie probability distances for 5% Mutation Rate**

387 (same as ISLA with 600 iterations). Around 10 subsequences didn't produce a BLAST hit and that shows the small bin near 1.

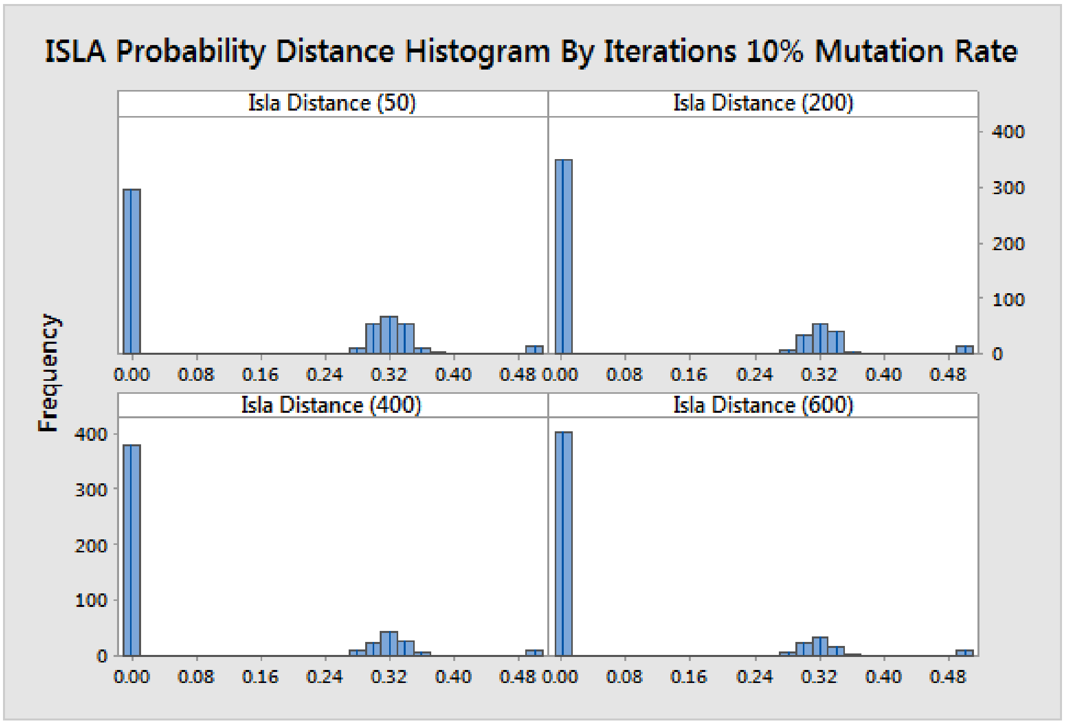


Figure 5–8: **ISLA probability distances for 10% Mutation Rate**

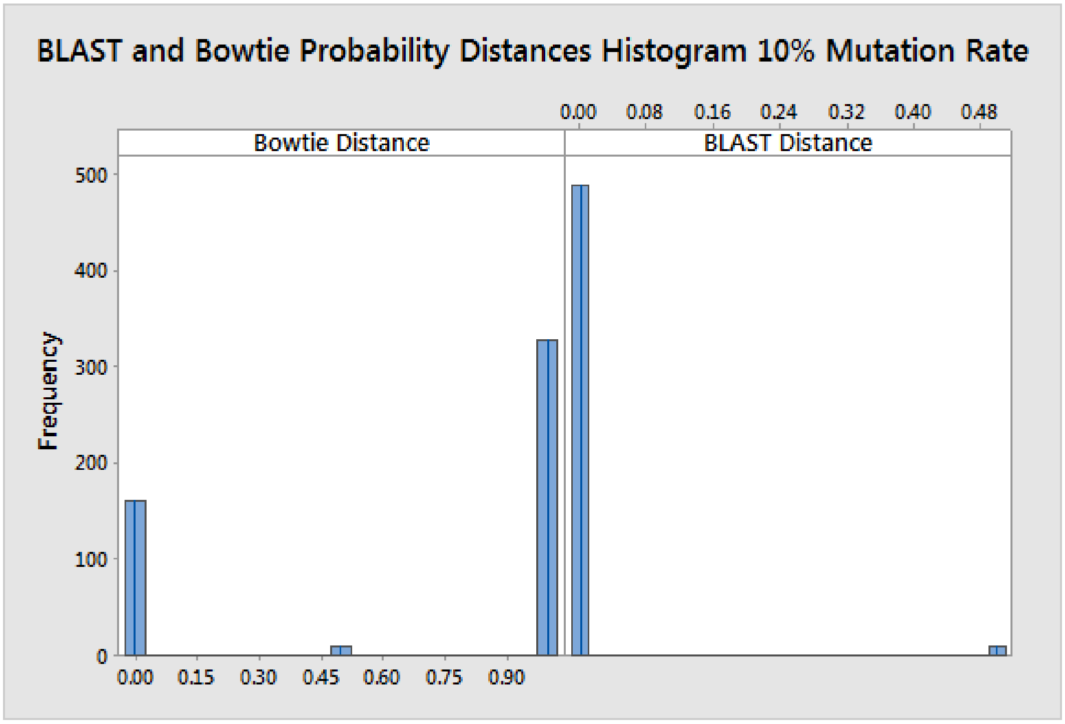


Figure 5–9: **BLAST and Bowtie probability distances for 10% Mutation Rate**

Figure 5–10 shows the ISLA results for 15% mutation rate. Here the optimal positions ranged from 260 with 50 iterations to 376 with 600 iterations. As seen on figure 5–11 Bowtie was only able to find optimal positions for 162 subsequences, while BLAST found 320. However, as per the histogram it seems the number is near 500 but that is because the bin is adding all other close to zero distances, which are about 150 of them. It can also be seen that Bowtie was not able to provide valid positions for most of the subsequences.

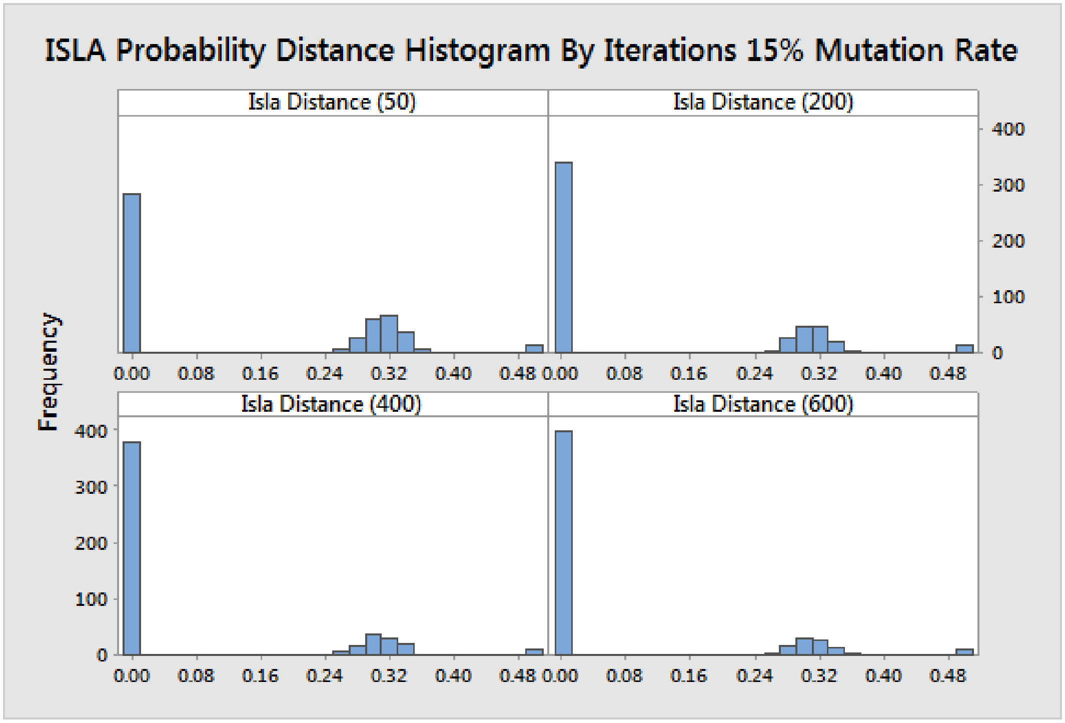


Figure 5–10: **ISLA probability distances for 15% Mutation Rate**

Finally, figure 5–12 shows the results for the ISLA runs for the 20% mutation rate subsequences. Optimal positions ranged from 229 with 50 iterations to 353 with 600 iterations. From figure 5–13 it can be seen that Bowtie was only able to find 5 optimal positions while BLAST achieved 180 optimal positions.

5.5.4 Discussion

As results shown above established, ISLA algorithm achieved better accuracy when compared with Bowtie and BLAST tools. Even when the execution times of BLAST and Bowtie were still considerably faster than ISLA, ISLA was able to achieve magnitudes of speed faster when compared to optimal Smith-Waterman

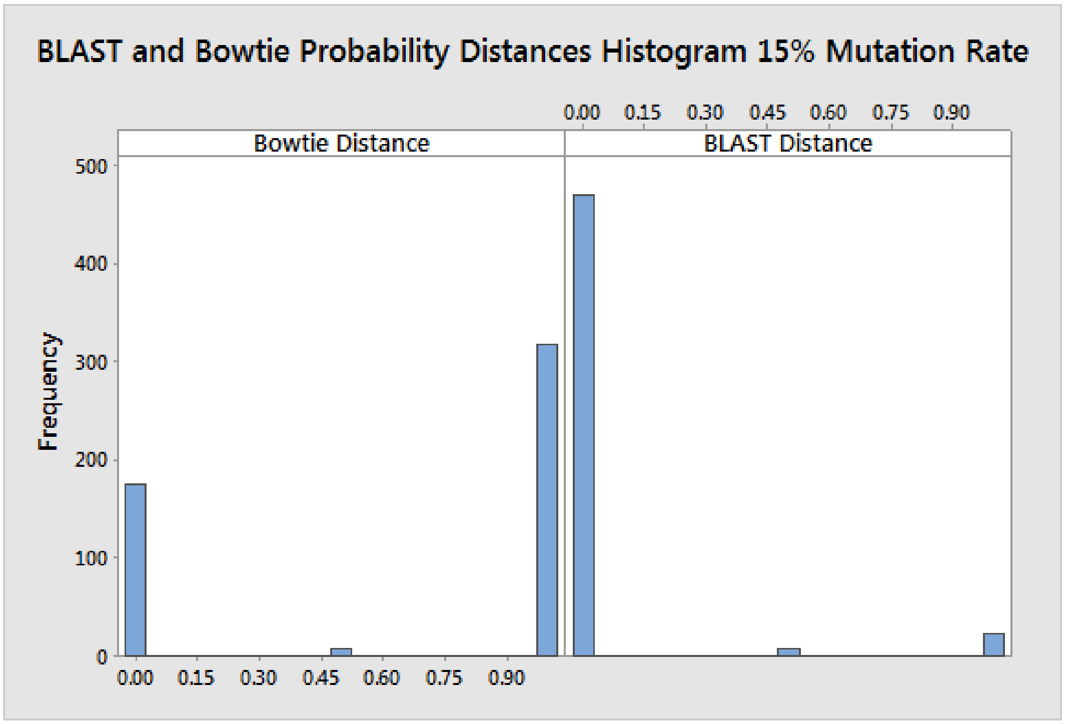


Figure 5–11: **BLAST and Bowtie probability distances for 15% Mutation Rate**

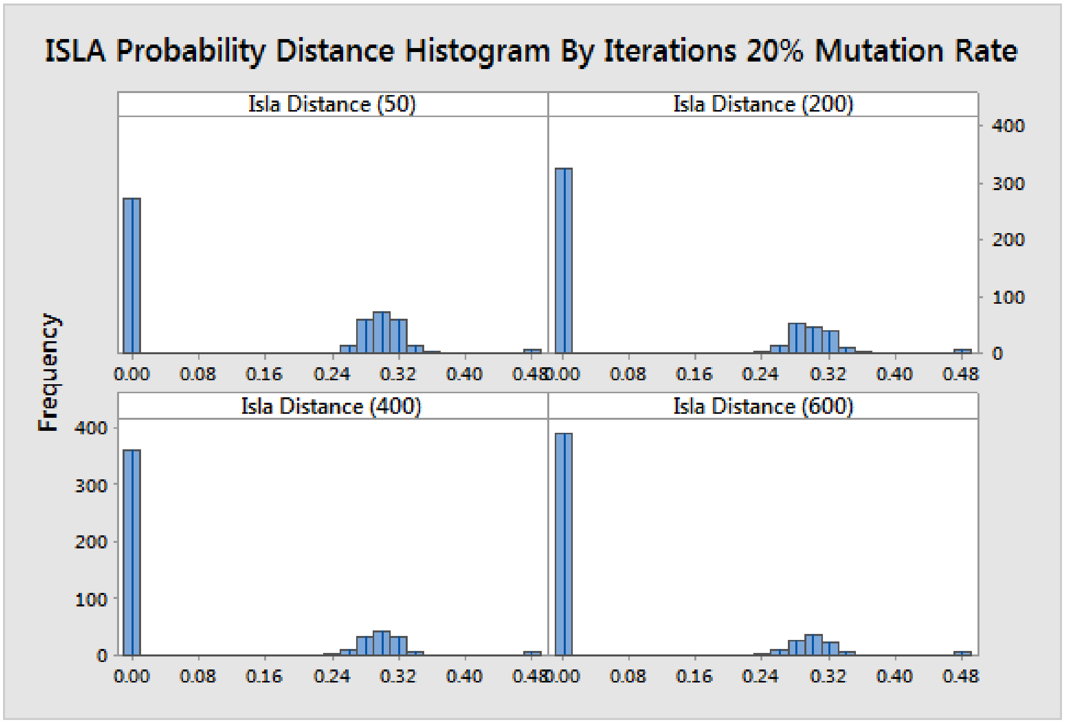


Figure 5–12: **ISLA probability distances for 20% Mutation Rate**

alignments. Even when the computational complexity of ISLA is still O (NM), its

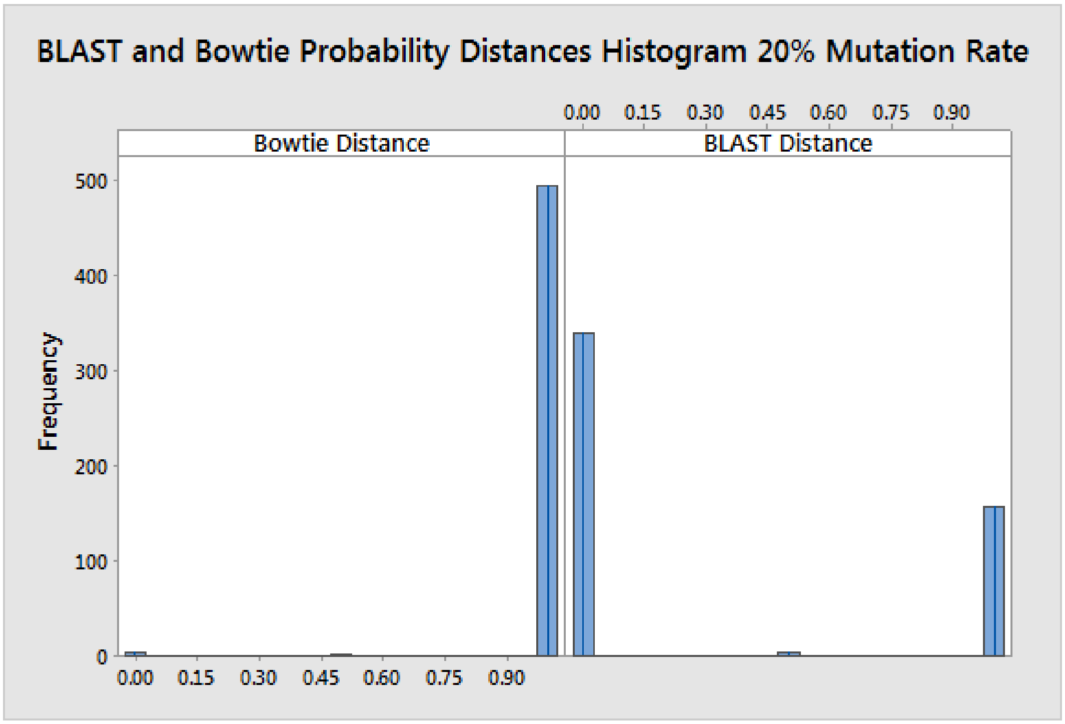


Figure 5–13: **BLAST and Bowtie probability distances for 20% Mutation Rate**

constants are a small fraction of normal Smith-Waterman algorithm since the search space is reduced considerably.

5.6 Reproducible Research

The Reproducible Research Standard (RRS) is a scientific community effort to increase cooperation and leverage across researchers and universities. Its main goal is to free the scientific work from copying and reuse restrictions, with attribution. The general guidelines of RRS are:

1. Release media components (text, figures) under the Creative Commons Attribution License (also known as CC-BY).
2. Release code components under MIT license or similar.
3. Attribution license on selection and arrangement.
4. All Data must be released under CC0.

As part of this research we are aiming on being RRS fully compliant where we can be.

Licensing. ISLA code and associated tools are released under MIT licensing as established on [45]. This frees the scientific community from copying and reusing the code while UPR maintains the copyrights.

Data. As explained on section 5.3, the source data used for gathering ISLA results is already public domain. However, to be compliant with the RRS all charts and results presented in this thesis should be surrendered into Creative Commons Attribution License. Even when the author does not have any problem with that, the contents of this thesis is automatically Copyrighted material from the University of Puerto Rico, thus there will be a conflict with the university.

6. Conclusion and Future Work

6.1 Conclusion

A new and novel algorithm was presented that is able to achieve high accuracy, while decreasing the computational time of optimal local alignments. This algorithm results were compared to the preferred tools used by the scientific community and data shows it can achieve equal or better precision. Algorithm also provides a novel probabilistic model that can show insights on alignment certainty.

Even when the author is satisfied with the results there are still areas for research and improvements that can be achieved. Next section expand on two open areas that can be exploited to optimize even further the current implementation.

6.2 Future Work

6.2.1 Unbalanced Alignments

Even when the ISLA algorithm focused on highly unbalanced sequences, there is no clear definition on what the term *highly* means. ISLA can produce an optimal score between two sequences, one of 1000bp and another of 100bp, but the algorithm overhead will add some much latency to the calculation that it does not make sense to even try it. However, what is the *cut-off* value for ISLA? How can the algorithm can determine if it makes sense to use ISLA or simply run normal Smith-Waterman?

Some benchmarks performed by the author shows that for small long-to-short length ratios, ISLA algorithm execute 50 times slower than a simple SW implementation. It will be good to find such a ratio threshold and don't try to use ISLA for all short sequences, just for the ones that satisfy certain length ratio.

6.2.2 Probability Model Enhancements

The novel probability model presented in this research can be used to understand if the current alignment is close to a perfect match (low probability of finding something better), or if the alignment based on the matches found may be discarded since there is a high probability of finding something better. Even when this metric provides new insights that were not available before, it will be good to think how it may be improved to take into account the current sequences. Fox example, the best possible alignment between two sequences can still yield a high probability of finding something better, but it is not because the obtained alignment is low quality. In cases where there is simply poor matching between the two sequences, probability will be high but it is not due the lack of good alignment. There is an area of opportunity to enhance the current model to at least provide some insights with the probability that can provide a guideline based on the matching quality between the sequences under study.

7. Ethical Considerations on Genome Sequencing

This chapter deals with the ethical considerations in the area of DNA sequencing and sequence alignments as part of medical treatments. It explores the current contended issues in the field and briefly discuss the different point of views in the area.

7.1 Incidental Findings

Due the decreasing costs on genome sequencing, medicine practitioners are now increasingly incorporating the procedure as part of their diagnostics tools. This clinical sequencing (it maybe be a whole genome or just exomes) is often ordered in multiple medical situations that can help the practitioner get insights on rare diseases, provide individualized treatment, screenings during pregnancy, asses possible drugs responses based on gene functions and other diseases predisposition. Since there exists an increasing number of genes and gene variants that have been correlated to known diseases or risks, a single sequencing can lead to secondary discoveries that are not related to the primary medical interest. These secondary findings are known as *incidental findings* and they are a major source of ethical concern in the medical community.

In 2012 the American College of Medical Genetics and Genomics (ACMG) published a statement defining a new set of guidances on how to handle patient sequencing procedures. The ACMG argues their main objective behind the new protocols was to set a professional standard for laboratory best practices. On these recommendations, the ACMG indicated a minimum list of conditions (57 specified genes) that should be checked and reported by laboratories when doing common clinical sequencing. Moreover, the guidelines indicate that these findings should be reported *without validating patient preferences*[46]. The recommendations also establishes that the reporting should not be contingent on patient's age, thus the parents of a child being clinically sequenced does not have a say on incidental findings. This specific guideline triggered an ongoing ethical discussion on patients rights and privacy. One group in the medical community argue that genome sequencing is just a single test and individual genes or variations checks should not be considered as discrete tests. If these screenings are not tests, then there is no need for distinct patient consent[47]. The other group contend that these guidelines are the same as testing without consent and that the ACMG recommendations is an abrupt change on patient autonomy. Their point is that the subject has a right of not to know and also has the right to make a decision which can be different from what the medical practitioner might choose [48].

Even when our thesis work may speed up the genome assembly process, and as a consequence lower the costs of the test itself, the ethical debate on reporting (or not) incidental findings will continue regardless of our work.

7.2 Code Source

What it is being considered one of the most relevant ethical issues in Bioinformatics is the case of open/closed source software. It is argued that all sequencing software should be open source since it encourages scrutiny from the scientific community and encourage replication and improvement of algorithms. On the other hand, corporations that spend big part of their R&D budgets on trying to reach a technology edge that can differentiate themselves from their competitors are not willing to give that away.

This research try to adhered to the Research Reproducibility Standard (RRS) and thus all the algorithm source code is available to the scientific community under the MIT licensing which will allows researchers to use it, modify it while giving respective attribute to authors.

7.3 Legal Issues

In this section legal ethical issues affecting the sequencing of data are briefly covered. It also discussed how these issues will be addressed under this proposed research.

7.3.1 Open Data

One of the issues that are being debated by the scientific community on recent years is the availability of new sequenced data. In 2000 Science Magazine had an agreement with the Celera Genomics company where it allowed Celera to made available their human genome sequence from its web site instead of common sequence database consortiums as GenBank[19], MBL and DDBJ. Until that point it was required that for all DNA/Protein related scientific publications to be considered by Science should use sequences commonly available on these databases. If a new sequence was generated, then that sequenced needed to be added to them. Under the agreement, even when the access was granted to the genome data to every interested party, they were not able to distribute it further.

This change in direction opened the door to a interesting debate between biologists regarding open data. Biologists were concerned that limiting the access to data could restrict their ability to easily compare and aligned new data based on a common infrastructure and that all sequenced data should be open and should be accessed with minimal legal encumbrances.

This research encourages RRS protocol and all the new data produced will be under the Creative Commons (CC0) licensing. However, no new sequencing data was produced as part of this research.

7.3.2 Gene Patents

It is estimated that more than $4,000$ genes of the $22,000$ genes in the human genome have U.S. patents against them. This specific number accounts for a polarizing debate on the scientific community. One part of the debate is concentrating on the companies right to a return of investment (ROI) after months or years studying certain health conditions. On the other hand, it is argued that certain genes are part of the human body and they are owned by humanity and not a single company.

APPENDICES

A. Bash Scripts Used

A.1 Random Sub-Sequence Generator

```
#!/bin/bash
#Copyright (c) <2016> <University of Puerto Rico at Mayaguez>
#
#Permission is hereby granted, free of charge, to any person
   obtaining a copy of this software and associated
   documentation files (the "Software"), to deal in the
   Software without restriction, including without limitation
    the rights to use, copy, modify, merge, publish,
   distribute, sublicense, and/or sell copies of the Software
   , and to permit persons to whom the Software is furnished
   to do so, subject to the following conditions:

#The above copyright notice and this permission notice shall
   be included in all copies or substantial portions of the
   Software.

#THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
   KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
    WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
   PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
   OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
    OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
    SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#__author__ = "Wilfredo Lugo"
#__copyright__ = "Copyright 2016, University of Puerto Rico
   at Mayaguez"
#__credits__ = ["Wilfredo Lugo"]
#__license__ = "MIT"
#__version__ = "1.0.1"

referenceFile=$1
exactsFile=$2
snpFile=$3
delFile=$4
insFile=$5
totalReads=16
minReadSz=$6
maxReadSz=$7
maxSNP=$8
maxDEL=$9
maxINS=${10}
echo "Generating $totalReads random reads against:
   $referenceFile"
totalReferenceLen=`cat $referenceFile | grep -v '>' | tr -d "
   \n" | wc -m`
everything=`cat $referenceFile| grep -v '>'|tr -d "\n"`
echo -n "">$exactsFile
```

```
for seed in ‘shuf -i 0-$totalReferenceLen -n $totalReads‘
do
    length=‘shuf -i $minReadSz-$maxReadSz -n 1‘
    read=${everything:$seed:$length}
    echo "@RandomSubSequence_${seed}_${length}" >>
       $exactsFile
    echo "$read"  >> $exactsFile
    echo "+"  >> $exactsFile
    quality=‘cat /dev/urandom | tr -dc ’<>?/:;!@#$%^&*()-_+a-
       zA-Z0-9’ | fold -w $length | head -n 1‘
    echo $quality >> $exactsFile
done

echo "Generating $totalReads reads with $maxSNP snps"
echo -n "">$snpFile
for seed in ‘shuf -i 0-$totalReferenceLen -n $totalReads‘
do
    length=‘shuf -i $minReadSz-$maxReadSz -n 1‘
    read=${everything:$seed:$length}
    temp=‘echo $read | grep N‘
    while [ $? != 1 ]
    do
        echo "N found on read $read"
        seed=‘shuf -i 0-$totalReferenceLen -n 1‘
        read=${everything:$seed:$length}
        temp=‘echo $read | grep N‘
    done
    snpCount=0
    while [ $snpCount -lt $maxSNP ]
    do
        rndPos=‘shuf -i 1-$length -n 1‘
        letter=${read:$rndPost:1}
        newLetter=""
        if [[ $letter = ’A’ ]]
        then
            newLetter=‘cat /dev/urandom | tr -dc "GTC" | fold
                -w 1 | head -n 1‘
        elif [[ $letter = ’T’ ]]
        then
            newLetter=‘cat /dev/urandom | tr -dc "GAC" | fold
                -w 1 | head -n 1‘
        elif [[ $letter = ’G’ ]]
        then
            newLetter=‘cat /dev/urandom | tr -dc "TAC" | fold
                -w 1 | head -n 1‘
        elif [[ $letter = ’C’ ]]
        then
            newLetter=‘cat /dev/urandom | tr -dc "TAG" | fold
                -w 1 | head -n 1‘
        elif [[ $letter = ’N’ ]]
        then
            newLetter=‘cat /dev/urandom | tr -dc "TAGC" |
               fold -w 1 | head -n 1‘
        fi
        let snpCount=snpCount+1
        read=‘echo $read | sed s/./$newLetter/$rndPos‘
    done
    echo "@RandomSubSequence_${seed}_${length}_${maxSNP}_SNP"
        >> $snpFile
    echo "$read">> $snpFile
    echo "+"  >> $snpFile
    quality=`cat /dev/urandom | tr -dc '<>?/:;!@#$%^&*()-_+a-
       zA-Z0-9' | fold -w $length | head -n 1`
    echo $quality >> $snpFile
done

echo "Generating $totalReads reads with $maxDEL deletions"
echo -n "">$delFile
for seed in `shuf -i 0-$totalReferenceLen -n $totalReads`
do
    length=`shuf -i $minReadSz-$maxReadSz -n 1`
    read=${everything:$seed:$length}
    temp=`echo $read | grep N`
    while [ $? != 1 ]
    do
        echo "N found on read $read"
        seed=`shuf -i 0-$totalReferenceLen -n 1`
        read=${everything:$seed:$length}
        temp=`echo $read | grep N`
    done
    newLength=$length
    delCount=0
    while [ $delCount -lt $maxDEL ]
    do
        let newLength=newLength-1
        rndPos=`shuf -i 1-$newLength-1 -n 1`
        let delCount=delCount+1
        read=${read:0:$rndPos}${read:$rndPos+1}
    done
    echo "@RandomSubSequence_${seed}_${newLength}_${maxDEL}
       _DEL" >> $delFile
    echo $read >> $delFile
    echo "+"  >> $delFile
    quality=`cat /dev/urandom | tr -dc '<>?/:;!@#$%^&*()-_+a-
       zA-Z0-9' | fold -w $newLength | head -n 1`
    echo $quality >> $delFile
done

echo "Generating $totalReads reads with $maxINS insertions"
echo -n "">$insFile
for seed in `shuf -i 0-$totalReferenceLen -n $totalReads`
do
    length=`shuf -i $minReadSz-$maxReadSz -n 1`
    read=${everything:$seed:$length}
    temp=`echo $read | grep N`
    while [ $? != 1 ]
    do
        echo "N found on read $read"
        seed=`shuf -i 0-$totalReferenceLen -n 1`
        read=${everything:$seed:$length}
        temp=`echo $read | grep N`
    done
    newLength=$length
    insCount=0
    while [ $insCount -lt $maxINS ]
    do
        rndPos=`shuf -i 0-$newLength-1 -n 1`
        newLetter=`cat /dev/urandom | tr -dc "TAGC" | fold -w
            1 | head -n 1`
        read=${read:0:$rndPos}${newLetter}${read:$rndPos}
        let newLength=newLength+1
        let insCount=insCount+1
    done
    echo "@RandomSubSequence_${seed}_${newLength}_${maxINS}
       _INS" >> $insFile
    echo $read >> $insFile
    echo "+"  >> $insFile
    quality=`cat /dev/urandom | tr -dc '<>?/:;!@#$%^&*()-_+a-
       zA-Z0-9' | fold -w $newLength | head -n 1`
    echo $quality >> $insFile
done
```

A.2 Benchmark Generation Automation

```
#!/bin/bash
#!/bin/bash
#Copyright (c) <2016> <University of Puerto Rico at Mayaguez>
#
#Permission is hereby granted, free of charge, to any person
   obtaining a copy of this software and associated
   documentation files (the "Software"), to deal in the
   Software without restriction, including without limitation
    the rights to use, copy, modify, merge, publish,
   distribute, sublicense, and/or sell copies of the Software
   , and to permit persons to whom the Software is furnished
   to do so, subject to the following conditions:

#The above copyright notice and this permission notice shall
   be included in all copies or substantial portions of the
   Software.

#THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
   KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
    WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
   PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
    OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
   OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
    OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
    SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#__author__ = "Wilfredo Lugo"
#__copyright__ = "Copyright 2016, University of Puerto Rico
   at Mayaguez"
#__credits__ = ["Wilfredo Lugo"]
#__license__ = "MIT"
#__version__ = "1.0.1"
yLen=100000
ratio=0.01
totalReads=500
readsQFile=reads.fastq
readsAFile=reads.fasta
refFile=ref.fasta
bowTieDBID=randRef
mutationRatio=0.20
islaRetries=50
outputFile=${yLen}_${ratio}_${mutationRatio}_${islaRetries}_$
   {totalReads}.csv
xLen=`echo "($yLen*$ratio)/1" | bc`


Y=`cat /dev/urandom | tr -dc 'AGTC' | fold -w $yLen | head -n
    1`
currRead=0
echo ">Random_Y_${yLen}_${ratio}" > $refFile
echo "$Y" >> $refFile
echo "$xLen"
let maxSeed=yLen-xLen
echo -n "" > $readsQFile
echo -n "" > $readsAFile

function getRandomMutation
{
    subSeq=$1
    #echo "before selection"
    selection=`shuf -i 1-3 -n 1`
    #echo "Selection: $selection"
    if [ $selection -eq 0 ]
    then
        echo "EXACT"
        #
        # Read remains Exact
        #
        mutatedRead=$subSeq
        mutationSuffix="EXACT"
        newLength=$xLen
    elif [ $selection -eq 1 ]
    then
        maxINS=`echo "($xLen*$mutationRatio)/1" | bc`
        #echo "MaxINS: $maxINS"
        newLength=$xLen
        insCount=0
        read=$subSeq
        while [ $insCount -lt $maxINS ]
        do
            rndPos=`shuf -i 0-$newLength-1 -n 1`
            newLetter=`cat /dev/urandom | tr -dc "TAGC" |
               fold -w 1 | head -n 1`
            read=${read:0:$rndPos}${newLetter}${read:$rndPos}
            let newLength=newLength+1
            let insCount=insCount+1
        done
        mutatedRead=$read
        mutationSuffix="${maxINS}_INS"
    elif [ $selection -eq 2 ]
    then
        maxDEL=`echo "($xLen*$mutationRatio)/1" | bc`
        #echo "MaxDEL: $maxDEL"
        newLength=$xLen
        #echo "DEL Original Length: $newLength"
        read=$subSeq
        delCount=0
        while [ $delCount -lt $maxDEL ]
        do
            let newLength=newLength-1
            let myLength=newLength-1
            rndPos=`shuf -i 1-$myLength -n 1`
            let delCount=delCount+1
            read=${read:0:$rndPos}${read:$rndPos+1}
        done
        mutatedRead=$read
        mutationSuffix="${maxDEL}_DEL"
        #echo "DEL Final Length: $newLength"
    elif [ $selection -eq 3 ]
    then
        maxSNP=`echo "($xLen*$mutationRatio)/1" | bc`
        #echo "MaxSNP: $maxSNP"
        newLength=$xLen
        read=$subSeq
        snpCount=0
        while [ $snpCount -lt $maxSNP ]
```

```
            do
                rndPos=`shuf -i 1-$newLength -n 1`
                letter=${read:$rndPost:1}
                newLetter=""
                if [[ $letter = 'A' ]]
                then
                    newLetter=`cat /dev/urandom | tr -dc "GTC" |
                       fold -w 1 | head -n 1`
                elif [[ $letter = 'T' ]]
                then
                    newLetter=`cat /dev/urandom | tr -dc "GAC" |
                       fold -w 1 | head -n 1`
                elif [[ $letter = 'G' ]]
                then
                    newLetter=`cat /dev/urandom | tr -dc "TAC" |
                       fold -w 1 | head -n 1`
                elif [[ $letter = 'C' ]]
                then
                    newLetter=`cat /dev/urandom | tr -dc "TAG" |
                       fold -w 1 | head -n 1`
                fi
                let snpCount=snpCount+1
                read=`echo $read | sed s/./$newLetter/$rndPos`
            done
            mutatedRead=$read
            mutationSuffix="${maxSNP}_SNP"
        fi
}

while [ $currRead -lt $totalReads ]
do
    seed=`shuf -i 0-$maxSeed -n 1`
    subSequence=${Y:$seed:$xLen}
    getRandomMutation $subSequence
    echo "@Random_X_From_${seed}_${currRead}_${mutationSuffix
       }" >> $readsQFile
    echo "$mutatedRead" >> $readsQFile
    echo "+" >> $readsQFile
    #
    # Generating random quality
    #
    #echo "Quality length: $newLength"
    quality=`cat /dev/urandom | tr -dc '<>?/:;!@#$%^&*()-_+a-
       zA-Z0-9' | fold -w $newLength | head -n 1`
    echo "$quality" >> $readsQFile
    let currRead=currRead+1
    echo "Finished read: $currRead"
done
echo "Generating FASTA file"
../Data/bbmap/reformat.sh in=$readsQFile out=$readsAFile qin
   =33 overwrite=true

echo "Making BLAST Reference Indexing"
../ncbi-blast-2.2.31+-src/c++/ReleaseMT/build/app/blastdb/
   makeblastdb -in $refFile -parse_seqids -dbtype nucl
echo "Making Bowtie Reference Indexing"
../bowtie2-2.2.5/bowtie2-build -f $refFile $bowTieDBID

echo "Running ISLA"
python src/isla.py -l islabench.log -r $refFile -q
   $readsQFile -i $islaRetries > isla.out

echo "Running BLAST"
```

```
../ncbi-blast-2.2.31+-src/c++/ReleaseMT/build/app/blast/
   blastn -query $readsAFile -db $refFile -outfmt "7 qseqid
   sstart" | grep -v "#">blast.out
echo "Running Bowtie"
../bowtie2-2.2.5/bowtie2 -a -x  $bowTieDBID -q -U $readsQFile
    --very-sensitive > bowtie.out
echo "Creating output $outputFile"
function processRead
{
    seq=$1
    bowtieHit=`cat bowtie.out | grep $seq | wc -l`
    blastHit=`cat blast.out | grep $seq | wc -l`
#     if [ $bowtieHit -eq 0 -o $blastHit -eq 0 ]
#     then
#         echo "No consensus hit found for read $seq"
#         continue
#     fi
    #readSeq=`python getReadFromFile.py $readsAFile $seq`
    optimalProb=`python SmithWaterman.py $refFile $readsAFile
        $seq Random_Y_${yLen}_${ratio} 0 end`
    echo "Optimal Prob: $optimalProb"
    islaProb=`cat isla.out | grep $seq | head -n 1 | awk -F",
       " '{print $5}'`
    islaIndex=`cat isla.out | grep $seq | head -n 1 | awk -F"
       ," '{print $3}'`
    echo "ISLA Index=$islaIndex"
    #
    # Processing Blast Results
    #
    blastIndex=`cat blast.out | grep $seq | awk -F" " '{print
        $2}'`
    let blastIndex=blastIndex-1
    blastEnd=`echo "($blastIndex+($xLen*3))/1"| bc`
    echo "BLAST Index=$blastIndex, BLAST End: $blastEnd"
    blastSubSeq=${Y:$blastIndex:$blastEnd}
    blastProb=`python SmithWaterman.py $refFile $readsAFile
       $seq Random_Y_${yLen}_${ratio} $blastIndex $blastEnd`

    #
    # Processing Bowtie Results
    #
    bowtieIndex=`cat bowtie.out | grep $seq | awk -F" " '{
       print $4}'`
    let bowtieIndex=bowtieIndex-1
    bowtieEnd=`echo "($bowtieIndex+($xLen*3))/1"| bc`
    echo "Bowtie Index=$bowtieIndex, Bowtie End: $bowtieEnd"
    bowtieSubSeq=${Y:$bowtieIndex:$bowtieEnd}
    bowtieProb=`python SmithWaterman.py $refFile $readsAFile
       $seq Random_Y_${yLen}_${ratio} $bowtieIndex $bowtieEnd
       `
    #bowtieProb=`python SmithWaterman.py $bowtieSubSeq
       $readSeq`
    echo $seq,$islaIndex,$blastIndex,$bowtieIndex,
       $optimalProb,$islaProb,$bowtieProb,$blastProb>>
       $outputFile
}

maxConcurrent=85
totalProcs=0
```

```
echo "Read Name,ISLA Position,BLAST Position,BowTie Position,
   Optimal Probability,ISLA Probability,Bowtie Probability,
   BLAST Probability">$outputFile
for seq in `cat $readsAFile | grep ">"`
do
    seq=`echo $seq | awk -F">" '{print $2}'`
    echo "Processing read: $seq"
    processRead $seq &
    let totalProcs=totalProcs+1
    if [ $totalProcs -ge $maxConcurrent ]
    then
        echo "Sleeping to let other processes finish"
        sleep 600
        totalProcs=0
    fi
done
```

References

[1] J. D. Watson and F. H. Crick. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, April 1953.

[2] A. W. F. Edwards and L. L. Cavalli-Sforza. *Reconstruction of evolutionary trees*, volume 6, pages 67–76. Systematics Association Publ., London, 1964.

[3] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. Biological sequence analysis: probabilistic models of proteins and nucleic acids, 1998.

[4] Erwin Chargaff. Calculated composition of a 'messenger' ribonucleic acid. *Nature*, 1962.

[5] F. H. Crick. On protein synthesis. *Symposia of the Society for Experimental Biology*, 12:138–163, 1958.

[6] Stephen P. Jackson and Jiri Bartek. The DNA-damage response in human biology and disease. *Nature*, 461(7267):1071–1078, October 2009.

[7] Michael R. Stratton, Peter J. Campbell, and P. Andrew Futreal. The cancer genome. *Nature*, 458(7239):719–724, April 2009.

[8] S. Wells. *The Journey of Man: A Genetic Odyssey.* Random House Publishing Group, 2012.

[9] R. Frankham, J.D. Ballou, and D.A. Briscoe. *Introduction to Conservation Genetics.* Cambridge University Press, 2010.

[10] J.M. Butler. *Forensic DNA Typing: Biology, Technology, and Genetics of STR Markers.* Forensic DNA Typing Series. Elsevier Science, 2005.

[11] Jan-Peter Nap, Peter L. J. Metz, Marga Escaler, and Anthony J. Conner. The release of genetically modified crops into the environment. *The Plant Journal*,

33(1):1–18, January 2003.

[12] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5(suppl 3):345–351, 1978.

[13] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, November 1992.

[14] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[15] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[16] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of Next-Generation Sequencing Systems. *Journal of Biomedicine and Biotechnology*, 2012:1–11, 2012.

[17] U. K. Laemmli. Cleavage of Structural Proteins during the Assembly of the Head of Bacteriophage T4. *Nature*, 227(5259):680–685, August 1970.

[18] Richard Williams, Sergio G. Peisajovich, Oliver J. Miller, Shlomo Magdassi, Dan S. Tawfik, and Andrew D. Griffiths. Amplification of complex gene libraries by emulsion PCR. *Nat Meth*, 3(7):545–550, July 2006.

[19] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. GenBank. *Nucleic Acids Res*, 31(1):23–27, January 2003.

[20] Rasko Leinonen, Ruth Akhtar, Ewan Birney, Lawrence Bower, Ana Cerdeno-Tárraga, Ying Cheng, Iain Cleland, Nadeem Faruque, Neil Goodgame, Richard Gibson, Gemma Hoad, Mikyung Jang, Nima Pakseresht, Sheila Plaister, Rajesh Radhakrishnan, Kethi Reddy, Siamak Sobhany, Petra T. Hoopen, Robert Vaughan, Vadim Zalunin, and Guy Cochrane. The European Nucleotide

Archive. *Nucleic Acids Research*, 39(suppl 1):D28–D31, January 2011.

[21] Haig H. Kazazian. Mobile elements: drivers of genome evolution. *Science (New York, N.Y.)*, 303(5664):1626–1632, March 2004.

[22] Henry L. Levin and John V. Moran. Dynamic interactions between transposable elements and their hosts. *Nat Rev Genet*, 12(9):615–627, September 2011.

[23] Christine R. Beck, José Luis L. Garcia-Perez, Richard M. Badge, and John V. Moran. LINE-1 elements in structural variation and disease. *Annual review of genomics and human genetics*, 12:187–215, September 2011.

[24] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[25] S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, March 1990.

[26] Phil Green. Swat and cross match tools. http://www.phrap.org/phredphrap/general.html, 1996. Accessed on 2016-10-23.

[27] Yifeng Li and Hesham H. Ali. *SWAT: A New Spliced Alignment Tool Tailored for Handling More Sequencing Errors*, pages 927–935. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[28] Travis J. Wheeler and Sean R. Eddy. nhmmer: Dna homology search with profile hmms. *Bioinformatics*, 29(19):2487–2489, 2013.

[29] Daniel R. Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5):821–829, May 2008.

[30] J. J. Kent. BLAT - the BLAST-like alignment tool. *Genome Research*, 12(4):656–664, April 2002.

[31] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States*

of America, 85(8):2444–2448, April 1988.

[32] Hui Jiang and Wing Hung H. Wong. SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics (Oxford, England)*, 24(20):2395–2396, October 2008.

[33] Heng Li, Jue Ruan, and Richard Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851–1858, November 2008.

[34] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25–10, March 2009.

[35] Z. Ning, A. J. Cox, and J. C. Mullikin. SSAHA: a fast search method for large DNA databases. *Genome research*, 11(10):1725–1729, October 2001.

[36] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1 edition, May 1997.

[37] Python Software Foundation. Python official site. https://www.python.org/. Accessed on 2016-11-01.

[38] Wilfredo Lugo and Jaime Seguel. A fast and accurate parallel algorithm for genome mapping assembly aimed at massively parallel sequencers. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '15, pages 574–581, New York, NY, USA, 2015. ACM.

[39] The Python Wiki. Parallel processing and multiprocessing in python. https://wiki.python.org/moin/ParallelProcessing. Accessed on 2016-11-06.

[40] Vitalii Vanovschi. Parallel python software. http://www.parallelpython.com/. Accessed on 2016-11-06.

[41] M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, February 2005.

[42] Henry Gomersall. Pyfftw documentation. https://hgomersall.github.io/pyFFTW/. Accessed on 2016-11-01.

[43] Ensembl. Stickleback: Gasterosteus aculeatus. http://www.ensembl.org/Gasterosteus_aculeatus/Info/Index. Accessed on 2016-11-06.

[44] Hewlett Packard Enterprise. Hp bl920s gen8 system quickspecs. https://www.hpe.com/h20195/V2/getpdf.aspx/c04383189.pdf?ver=1. Accessed on 2016-11-06.

[45] Open Source Initiative. The mit license. https://opensource.org/licenses/MIT. Accessed on 2016-11-06.

[46] Robert C. Green, Jonathan S. Berg, Wayne W. Grody, Sarah S. Kalia, Bruce R. Korf, Christa L. Martin, Amy L. McGuire, Robert L. Nussbaum, Julianne M. O/D́aniel, Kelly E. Ormond, Heidi L. Rehm, Michael S. Watson, Marc S. Williams, and Leslie G. Biesecker. ACMG recommendations for reporting of incidental findings in clinical exome and genome sequencing. *Genetics in Medicine*, 15(7):565–674, 2013.

[47] A. L. McGuire, S. Joffe, B. A. Koenig, B. B. Biesecker, L. B. McCullough, J. S. Blumenthal-Barby, T. Caulfield, S. F. Terry, and R. C. Green. Ethics and Genomic Incidental Findings. *Science*, 340(6136):1047–1048, May 2013.

[48] S. M. Wolf, G. J. Annas, and S. Elias. Patient Autonomy and Incidental Findings in Clinical Genomics. *Science*, 340(6136):1049–1050, May 2013.